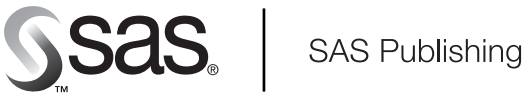


Technical Report: S2-107 XBUF Caching Feature in SYSTEM 2000® Software

Release 11.6 under IBM OS

The Power to Know.®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 1989. *Technical Report: S2-107 XBUF Caching Feature in SYSTEM 2000® Software, Release 11.6 under IBM OS*. Cary, NC: SAS Institute Inc.

Technical Report: S2-107 XBUF Caching Feature in SYSTEM 2000® Software, Release 11.6 under IBM OS

ISBN 1-55544-159-9

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

1st printing, 1989

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

Contents

Preface

This technical report, *S2-107 XBUF Caching Feature in SYSTEM 2000® Software, Release 11.6 under IBM OS,* discusses dual logging for data base files and cache areas that you can set up in 31-bit addressable memory or on high-speed disk devices. XBUF caching offers supplemental buffers to the SYSTEM 2000 I/O buffers.

This report is divided into eight chapters.

Chapter 1, INTRODUCTION, gives an overview of the XBUF features.

Chapter 2, FRONT-END CACHING MACROS, discusses the simple, front-end macros that you can use to set up dual logging, simulated caching, and real cache areas. These macros use default settings for the XBUF parameters.

Chapter 3, INVOKING XBUF SOFTWARE, shows you how to invoke the XBUF software with the XBUF execution parameters.

Chapter 4, THE CACHE MACRO, explains all options, parameters, and subparameters available in the basiC CACHE macro. You can use this macro if you want to customize the cache areas at your site by setting your own values for the XBUF options and parameters. The CACHE macro is an alternative to the front-end macros discussed in Chapter 2.

Chapter 5, DATA BASE FILE EXTENSION AND XBUF CACHING, describes file extension and how it pertains to caching.

Chapter 6, XBUF CONSOLE COMMANDS, discusses console commands that you can use to request XBUF statistics (real or simulated) and to reset parameter values while XBUF is executing.

Chapter 7, XBUF MAIN MEMORY, describes the various tables that XBUF sets up internally.

Chapter 8, XBUF MESSAGES, lists messages that you might receive when you assemble the front-end macros and messages that can appear while XBUF is executing.

Chapter 1
Introduction

Release 11.6 offers an Extended Buffer (XBUF) Manager feature, which allows you to use several kinds of caching techniques. XBUF caching is available for single-user and Multi-User jobs under OS. Cache areas pertain only to data base files, not scratch files or other temporary files. XBUF intercepts only BDAM I/O, not all data base I/O. (Since SYSTEM 2000 software uses sequential I/O for save and restore operations, they take place without affecting cache areas.)

Simple, front-end XBUF macros are available for

- dual logging of data base files
- caching data base files to secondary buffers in 31-bit addressable memory
- caching data base files to a high-speed disk device
- modeling cache activity
- collecting statistics about SYSTEM 2000 buffer pool and XBUF cache activities.

The simple XBUF macros let you try caching without having to learn about all the XBUF options. They use the default settings of parameters.

Chapter 2, "Front-End Caching Macros" shows you how to start using the various XBUF features quickly and easily with the front-end macros. Later, if you want to tailor caching for special applications at your site, see Chapter 4, "The CACHE Macro." That chapter contains background material on caching and describes the more complex CACHE macro, along with its many options.

Chapter 6, "XBUF Console Commands" discusses the XBUF console commands for displaying caching activities, such as the number of reads, writes, and drops in the various cache areas. In addition, you can display and reset most of the CACHE macro options during a Multi-User session.

DUAL LOGGING

XBUF allows you to specify one or more data base files for dual logging. When dual logging is activated, XBUF maintains two identical copies of the selected files. If the primary copy is lost, you can continue to process the data base by substituting the second copy for the primary copy. Dual logging is especially appropriate for data base Files 1, 7, and 8. These files are relatively small, but the information contained in them is essential for data base recovery.

The XBDUAL Macro on page 2-2 describes how to set up dual logging with the XBDUAL macro. You can try dual logging even if your computer site does not have the high-speed devices, usually associated with caching operations.

31-BIT ADDRESSABLE BUFFERS

XBUF allows you to create and use secondary buffers in 31-bit addressable memory. Here again, XBUF caching techniques offer advantages that do not require any extra high-speed devices.

SYSTEM 2000 software can access data in 24-bit addressable memory only. Theoretically, 24-bit addressable memory contains 16 megabytes of storage. But actually, only 4 to 6 megabytes are available for the buffers in storage. If you need to access a record that is not in a buffer pool, at least one I/O operation must be performed to bring the data into memory.

XBUF can expand the buffering of data base files to memory that can be addressed by 31 bits. Thus, up to 2048 megabytes are theoretically available to hold data without the system having to reference disk data sets. The data that XBUF keeps in 31-bit addressable memory (above the 24-bit memory) cannot be used directly by SYSTEM 2000 software. However, the XBUF instructions that move the data to the SYSTEM 2000 area are significantly faster than the I/O operations necessary to bring data from a disk to a buffer pool.

For details about using 31-bit addressable buffers, see **The XBMEMORY Macro** on page 2-4.

CACHING TO HIGH-SPEED DISK DEVICES

If your computer site has high-speed disk devices, you can use the front-end macros to define a cache file on a selected device. For details about associating specific data base files in high-speed cache files, see **The XBCACHE Macro** on page 2-5.

COLLECTING AND DISPLAYING STATISTICS

Dual logging, 31-bit addressable memory, and cache files are not always applicable to a SYSTEM 2000 data base. Depending on the mix of transactions that are unique to each SYSTEM 2000 production environment, you may not notice any improvement with XBUF.

A significant option of XBUF addresses the question of what XBUF caching would achieve at your site. Without using any physical resources, XBUF can simulate its own operation and collect statistics on how it would have performed if you had specified real dual logging, 31-bit buffers, or high-speed disk caching. You can then issue XBUF console commands to display these statistics and use this information to study the performance of your SYSTEM 2000 jobs or to decide whether to actually use XBUF.

Requesting XBUF I/O statistics with console commands can help you decide whether your application is I/O bound. The symptom is an I/O rate approaching the maximum for your disk devices, as published by the disk vendor. For example, a 3375 disk can typically service from 10 to 50 reads per second, depending on the arm movement required and the average record size. The fixed-head portion of a 3350 has no moving arm and can deliver 4096-byte records to a mainframe computer (running MVS) at approximately 120 records per second.

Suppose your model shows that important applications were driving disk data base files at, say, 600 reads-per-minute or more. You could expect that XBUF cache areas would cost you no more in terms of CPU and memory and would speed up the whole system.

On the other hand, if disks are being driven at only a few dozen reads-per-minute, you do not have a disk I/O bottleneck. Moving data base files (or a software-selected part of them) to a cache area improves elapsed time required for reads, but it does not help your applications. You must look elsewhere in your system for performance problems.

For details about how to simulate caching, see **The XBSTAT Macro** on page 2-8. The XBUF timing (I/O) monitor is activated any time you specify XBUF=YES or STAT in your execution parameters. Also, you can turn timing on and off with the CACHE macro and XBUF console commands.

Chapter 2

Front-End Caching Macros

Six simple, front-end macros for XBUF caching are discussed in this chapter.

- The XB macro starts and ends a set of cache specifications.
- The XBDUAL macro sets up dual logging for a data base file.
- The XBMEMORY macro defines a cache area in 31-bit addressable memory.
- The XBCACHE macro defines a cache file on a high-speed disk device.
- The XBDD macro specifies the DDname of a data base file to be cached.
- The XBSTAT macro simulates caching.

These front-end macros let you use XBUF caching without your having to learn much about how caching works internally. For example, you can try dual logging or 31-bit memory quickly, using default settings for most of the basic XBUF options. The defaults and assumptions are presented in **Assumptions and Default Settings** on page 2-9. You cannot mix these front-end macros with the complex CACHE macro discussed in Chapter 4, "The CACHE Macro."

Note: before coding these macros, you should be familiar enough with IBM Assembler language to know about column restrictions, spacing, capitalization, and so forth. Also, you should know how to assemble and link the resulting table into a load library.

THE XB MACRO

The XB macro starts and ends a sequence of front-end XBUF macros.

```
XB |START
   |END
```

where

START starts generating the XBUFTBL table of control blocks.

END finishes generating the XBUFTBL table.

The XB START macro must be the first macro in a sequence of front-end macros, and the XB END macro must be the last.

Example The example below illustrates using the XB macro to designate three files for dual logging: File 1, File 7, and File 8 for the EMPLOYEE data base.

```
XB START
   XBDUAL=EMPLOYE1,BLKSIZE=4060
   XBDUAL=EMPLOYE7,BLKSIZE=4060
   XBDUAL=EMPLOYE8,BLKSIZE=4072
XB END
```

THE XBDUAL MACRO

The XBDUAL macro allows you to specify a data base file for dual logging. With dual logging, XBUF maintains two identical copies of a specified data base file. If the system or disk crashes, destroying one copy, the other copy can be easily substituted by changing the DDname in the JCL or by renaming the copy to the name of the original. Good candidates are data base Files 1, 7 (the Update Log), and 8 (the Rollback Log). These files are crucial for automatic recovery of a data base.

```
XBDUAL DDNAME=filename[,BLKSIZE=blksize]
```

where

filename is the DDname of the data base file that is to be dual logged.

The DDname must follow the conventions for data base file names. It is formed from the first seven characters of the data base name and a suffix of 1 through 8 in the eighth character. Replace embedded blanks and trailing blanks with Xs. For more details about data base file names, see Section 4.5.2 in the *SYSTEM 2000 DEFINE Language Manual.*

blksize is the block size for the data base file that is to be dual logged. The value of blksize must be an acceptable SYSTEM 2000 page size. The default is 2492. Note: the block size for File 8 must be 12 bytes larger than the largest block size for Files 1 through 6. This parameter is optional, because you can specify the block size in your JCL; if you specify the option, it overrides the JCL. (See the *SYSTEM 2000 Product Support Manual* for acceptable page sizes.)

Note: if XBUF=STAT in the execution parameters, the number of blocks to be modeled (simulated records) for the dual logged file is set to 999,999 by default. When XBUF=YES, the number of blocks is computed using the primary space allocated when the file is formatted for actual caching. The primary space for the dual log must be equal to or greater than the space for the data base file (PRIM+(15*SEC).

You can specify one or more XBDUAL macros in a sequence of front-end macros. Each XBDUAL macro specifies only one data base file to be dual logged.

The XBDUAL macro sets up the dual log DDnames by mapping the last character of the data base file names (1 through 8) into the last character of the dual log DDnames (A through H, respectively). For example,

Data Base DDname	Dual Log DDname
DBNAMEX1	DBNAMEXA
DBNAMEX2	DBNAMEXB
DBNAMEX3	DBNAMEXC
DBNAMEX4	DBNAMEXD
DBNAMEX5	DBNAMEXE
DBNAMEX6	DBNAMEXF
DBNAMEX7	DBNAMEXG
DBNAMEX8	DBNAMEXH

Since XBUF copies the data base file to the dual log every time the data base is physically opened, you may want to consider methods for keeping data bases open for an entire Multi-User session. (For details, contact SAS Technical Support Services.)

Example This example shows how to set up dual logging for File 1, File 7, and File 8 in the INSURANCE data base.

The following XBDUAL macros cause data base files INSURAN1, INSURAN7, and INSURAN8 to be copied to the dual log data sets INSURANA, INSURANG, and INSURANH, respectively. Thereafter, all updates made to INSURAN1, INSURAN7, and INSURAN8 are also made to INSURANA, INSURANG, and INSURANH.

Notice that if the largest block size for Files 1 through 6 is 4060, the block size for File 8 (the Rollback Log) must be 4072.

```
XB START
   XBDUAL DDNAME=INSURAN1,BLKSIZE=4060
   XBDUAL DDNAME=INSURAN7,BLKSIZE=4060
   XBDUAL DDNAME=INSURAN8,BLKSIZE=4072
XB END
```

THE XBMEMORY MACRO

The XBMEMORY macro allows you to specify an in-memory cache area that can utilize 31-bit addressable buffers. You specify a block size and the number of buffers for the cache area. Also, you must code one or more XBDD macros to indicate which data base files should participate in the cache (see **The XBDD Macro** on page 2-7).

If you specify the XBMEMORY macro more than once (for different block sizes or different data base files), more than one in-memory cache area will be set up.

```
XBMEMORY |START,BLKSIZE=blksize,COUNT=count
         |END
```

where

START — starts a set of macros that defines a secondary buffer area in 31-bit addressable memory. START is a positional parameter.

END — ends a set of macros that defined an in-memory buffer area. END is a positional parameter.

blksize — is the block size for the DDnames (specified in subsequent XBDD macros) that are eligible to be cached into this set of in-memory buffers. The value of blksize must be an acceptable SYSTEM 2000 page size. (See the *SYSTEM 2000 Product Support Manual* for acceptable page sizes.)

count — is the number of secondary buffers that should be made available in the in-memory cache area.

You must provide the block size and the number of buffers for each in-memory cache area that you define. You can specify these parameters in either the XBMEMORY START macro or the XBMEMORY END macro or in both. If both, the block size and count in the END statement override the values in the START statement. Also, there must be sufficient 31-bit addressable memory to allocate (blksize x count) bytes.

For example, to define a cache area in 31-bit addressable memory, do the following:

1. Code the XBMEMORY START macro.

2. Code at least one XBDD macro, which gives the DDname of a data base file to cached (see **The XBDD Macro** on page 2-7). If you want more than one data base file in the cache area, code consecutive XBDD macros between the XBMEMORY START and XBMEMORY END macros. You can include Files 1 through 6 for any data base; Files 7 and 8 are good candidates for dual logging only, since they are accessed sequentially.

3. Code the XBMEMORY END macro after the last XBDD macro.

Note: if you specified XBUF=STAT in the execution parameters, the XBMEMORY macro sets up a simulated in-memory cache area, which allows you to display cache I/O statistics.

Example The example below illustrates using the XBMEMORY macro to set up 31-bit addressable buffers in memory for Files 1 through 6 for the EMPLOYEE data base and File 2 for the PERSONNEL data base. Assume that the block size for these files is 4060 and that you want to set up 1,000 buffers in this cache area. Note: you could have specified the block size and count on the XBMEMORY END card.

This cache area will take approximately 4 megabytes of storage, which should not be a problem above the 24-bit line.

```
XB START
   XBMEMORY START,BLKSIZE=4060,COUNT=1000
      XBDD DDNAME=EMPLOYE
      XBDD DDNAME=PERSONN2
   XBMEMORY END
XB END
```

THE XBCACHE MACRO

The XBCACHE macro allows you to define a high-speed disk cache file. You specify the DDname of the cache file and one or more XBDD macros to indicate which data base files belong to the cache file.

If you want to cache data base files that have different page sizes, code several XBCACHE macros, each having a different block size, and assign the various data base files to the appropriate cache file.

```
XBCACHE |START,[BLKSIZE=blksize,][COUNT=count,]XBNAME=xbfilename
        |END
```

where

START — starts a set of macros that defines a disk cache file. START is a positional parameter.

END — ends a set of macros that defined a disk cache file. END is a positional parameter.

blksize — is the exact block size for the DDnames (specified in subsequent XBDD macros) that belong to this cache disk file. The value of blksize must be an acceptable SYSTEM 2000 page size. This parameter is optional because it can be checked when the data set for the cache file is physically opened. You must specify the block size either in the macro or in the JCL; if both places, the block sizes must match. (See the *SYSTEM 2000 Product Support Manual* for acceptable page sizes.)

count — is the number of secondary buffers that should be made available in this disk cache file. This parameter is optional because it can be determined when the cache file is physically opened.

xbfilename — is the DDname of the disk data set that will be used as the cache file.

You must specify the cache file DDname in either the XBCACHE START macro or the XBCACHE END macro. Also, an appropriate DD statement must be included in the single user or Multi-User JCL.

You can specify the block size and number of buffers in either the XBCACHE START macro or the XBCACHE END macro or in both. If both, the block size and count in the END statement override the values in the START statement.

If you do not declare the block size in the JCL, you must specify BLKSIZE and COUNT in the XBCACHE macro. If you declare BLKSIZE in the JCL and in the macro, the block sizes must match. Also, the disk cache file must have enough primary space to allocate (blksize x count) bytes. These requirements cannot be checked until the Multi-User or single-user job points to the data set.

For example, to define a high-speed disk cache file, do the following:

1. Code the XBCACHE START macro.

2. Code at least one XBDD macro, which gives the DDname of a data base file to be cached (See **The XBDD Macro** on page 2-7). If you want more than one data base file in the cache file, code consecutive XBDD macros between the XBCACHE START and XBCACHE END macros. You can include Files 1 through 6 for any data base; Files 7 and 8 are considered good candidates for dual logging only, since they are accessed sequentially.

3. Code the XBCACHE END macro after the last XBDD macro.

If XBUF=STAT in your execution parameters, the XBCACHE macro sets up simulated caching for the cache area, which allows you to display cache I/O statistics. Note: if you have a fast disk drive, you can easily try the XBCACHE macro for setting up disk cache files. However, as mentioned earlier, these simple macros assume many default settings. After you try the XBCACHE macro, you may want to read the remaining chapters in this technical report. They show you how to code the basic CACHE macro directly and describe how to set and reset the various parameters and options.

Example The example below illustrates using the XBCACHE macro to set up a disk cache file for Files 2, 4, 5, and 6 of the SALES data base along with File 5 and File 6 of the CARS data base. Assume that the page size for these files is 4060 and that you want to set up 1200 buffers in this cache file. FAST1 is the DDname for the cache file.

```
XB START
   XBCACHE START
      XBDD DDNAME=SALESXX2
      XBDD DDNAME=SALESXX4
      XBDD DDNAME=SALESXX5
      XBDD DDNAME=SALESXX6
      XBDD DDNAME=CARSXXX5
      XBDD DDNAME=CARSXXX6
   XBCACHE END,BLKSIZE=4060,COUNT=1200,XBNAME=FAST1
XB END
```

THE XBDD MACRO

The XBDD macro specifies a data base file to be cached in 31-bit addressable buffers or on a high-speed disk. The XBMEMORY macro and the XBCACHE macro each require at least one XBDD macro. (For dual logging, you specify one data base file directly in the XBDUAL macro so the XBDD macro is not needed.)

```
XBDD DDNAME=filename
```

where

filename — is the DDname of a data base file that is eligible for caching in 31-bit addressable memory or on a high-speed disk file.

You can specify either seven or eight characters in the DDname. For seven characters, XBUF assumes the characters represent a valid data base name (possibly padded with Xs) and sets up Files 1 through 6 by appending the numbers 1 through 6 to the given DDname. An eight-character DDname designates one specific data base file. Files 7 and 8 are not allowed in the XBDD macro; they are good candidates for dual logging only.

The DDname must follow the conventions for data base file names. That is, the DDname should consist of the first seven characters of the data base name and, optionally, a suffix of 1 through 6 in the eighth character. Replace embedded blanks and trailing blanks with Xs. For more details about DDnames, see Section 4.5.2 in the *SYSTEM 2000 DEFINE Language Manual.*

You must code at least one XBDD macro for each XBMEMORY or XBCACHE macro. If you give more than one XBDD macro in a sequence, all files specified by the DDnames belong to the same cache, and their block sizes must match the cache file block size. See the XBMEMORY macro, discussed in **The XBMEMORY Macro** on page 2-4 and the XBCACHE macro, discussed in **The XBCACHE Macro** on page 2-5.

Example The example below illustrates using the XBDD macro with the XBMEMORY macro. Data base Files 2 and 4 (the Index tables) for the CARS data base will belong to a cache area in 31-bit addressable memory.

```
XB START
   XBMEMORY START . . .
      XBDD DDNAME=CARSXXX2
      XBDD DDNAME=CARSXXX4
   XBMEMORY END . . .
XB END
```

THE XBSTAT MACRO

The XBSTAT macro allows you to simulate cache activity. You can model a caching area and analyze file activity for the various data base files. The BLKSIZE and COUNT parameters are the equivalent of the XBMEMORY macro, but only the control blocks are acquired (not the caching area). Control blocks take a minimal amount of memory.

When you use the XBSTAT macro, the XBUF I/O monitor is activated but actual caching is not performed. (The XBUF execution parameter must be set to either REAL or STAT.) Multi-User console commands let you display various I/O statistics for the simulated cache area (see Chapter 6, "XBUF Console Commands"). Also, when XBUF I/O monitoring is turned on, you can display data base file I/O rates as well as buffer pool information with MUSTATS console commands. For details, see the DBN= and POOLS console commands described in *Technical Report: S2-105 Changes and Enhancements to SYSTEM 2000 Software, Release 11.6 under IBM OS and CMS.*

The XBSTAT macro is often used to model caching of data base files that have a specific block size. In this case, give very large values to the COUNT and SCOUNT parameters so that the size of the simulated cache area is not a limiting factor.

```
XBSTAT BLKSIZE=blksize,COUNT=count,SCOUNT=scount
```

where

blksize	is the page size that is eligible to be cached in this simulated caching area. The value of blksize must be an acceptable SYSTEM 2000 page size. (See the *SYSTEM 2000 Product Support Manual* for acceptable page sizes.)
count	is the number of pages that could theoretically fit in this simulated caching area.
scount	is the maximum number of DDnames that can belong to this simulated caching area.

DDnames of data base files are assigned to the simulated caching area when a data base is opened, on a first-encountered-first-assigned basis, up to the maximum number of files specified in scount. You can specify one or more XBSTAT macros between the XB START and XB END macros.

Example The example below illustrates using the XBSTAT macro to request file activity statistics for the first twenty- four data base files that are opened. The simulated caching area will contain 1,000 blocks with block size 4060.

```
XB START
   XBSTAT BLKSIZE=4060,COUNT=1000,SCOUNT=24
XB END
```

ASSUMPTIONS AND DEFAULT SETTINGS

Following are brief descriptions of several assumptions and default settings in effect when you code the front-end XBUF macros. For more details about these options, see the CACHE macro discussed in Chapter 4, "The CACHE Macro."

Timing By default, XBUF timing is turned on so you can display various I/O statistics with console commands. The TIME parameter allows you to turn timing on and off. TIME=ON is the default. The SECI parameter is closely related to timing. SECI determines the interval (in seconds) at which all rates and counters are recomputed. If you are watching the rates very closely through repeated console commands, you want the rates updated frequently. The default for SECI is 10 seconds. (See **XBUF Timing Parameters: TIME and SECI** on page 4-10 for details on computing rates.)

Swapping Periodically, XBUF analyzes its own performance to see whether it can improve its throughput by swapping a disk file in or out of active caching for awhile. By default, SWAP=YES activates swapping for all cache files set up by the front-end macros. The SWAP parameter allows you to turn swapping on and off. Four other parameters are associated with swapping: CNTIN, LTDROPC, HTDROPC, and RRAT. They indicate the timing interval between swap analyses and a range of acceptable drops-per-minute rates between which no swaps are performed. The defaults are CNTIN=500, LTDROPC=400, HTDROPC=600, and RRAT=75. (See **Swapping Disk Files in Cache Areas** on page 4-11 for more details.) Dual logged files do not participate in swapping.

Runtype A runtype option of REAL, STAT, or NONE allows you to specify different kinds of modeling, monitoring, and real I/O to take place in the same system. The runtype option is available at three levels in the basic CACHE macro: the global level, the specific cache file level, and the data base file level. XBUF honors an order of precedence among the levels (see **Priority of runtypes** on page 4-20). The default is REAL for all caches set up with the front-end macros except for the simulated caches implemented by the XBSTAT macro. If the XBUF execution parameter equals STAT, all cache areas specified in your front-end macros become models and the caching is simulated.

Mapping The multiple entry (me) group option determines how XBUF maps data base file pages into a cache area. By default, XBUF operates at the individual record level. That is, one cache file record is acquired for each data base file page that needs to be kept in cache. If the multiple entry option is greater than 1, XBUF allocates cache space in groups of records. This option provides a way to trade precision of cache space management for main memory space. The smaller the value, the more main memory XBUF needs in order to keep track of the cache area, and the more accurately XBUF handles paging. (For more details, see **Option 1: Multiple Entry Groups** on page 4-16.)

Favoritism You can specify low and high priority percentages for each data base file in a cache area. These options determine how much space a disk file can occupy in the cache and thus allow you to favor one or more specific data base files. Priorities can be reset while Multi-User is executing. By default, the data base files in each cache area are treated equally. (See **Disk File Space Priorities** on page 4-26 for more details.)

XBUF I/O MONITOR

When XBUF software is activated, you can display I/O rates for real or simulated cache areas. The XBUF I/O monitor also monitors data base file and buffer pool I/O rates, which you can display with MUSTATS console commands in a Multi-User environment. You can analyze where (or if) you have a bottleneck due to poor allocation for files, buffers, or devices. You can see whether your system needs more buffers, threads, or scratch pads. You might discover that you have overallocated some of these resources, which causes SYSTEM 2000 software to waste time constantly deciding not to use them. Or you might find you are using everything available; if so, the system would probably perform better if you allocated more resources or set up appropriate cache areas

For details about using XBUF console commands, see Chapter 6, "XBUF Console Commands." For details about MUSTATS console commands, see *Technical Report: S2-105 Changes and Enhancements to SYSTEM 2000 Software, Release 11.6 under IBM OS and CMS.*

TYPES OF ERROR MESSAGES

Messages that occur because of errors in coding front-end macros are listed in **Error Messages from Front-End Macros** on page 8-1. Other messages, which occur during XBUF initialization and at execution time, are also listed in Chapter 8, "XBUF Messages."

Chapter 3

Invoking XBUF Software

ASSEMBLING AND LINKING PROCEDURES

To code the XBUF front-end macros, you must be familiar with IBM Assembler language in regard to column restrictions, spacing, capitalization, and so on. You must also be able to assemble and link the resulting table into a load library. The resulting load module name must be of the format XBUFTBL[suffix]. The optional suffix distinguishes one XBUF configuration from another when you generate several different XBUF load modules.

Physically, XBUF consists of two members of a load library. During execution, the member names are S2KXBUF and XBUFTBL[suffix]. S2KXBUF is executable code, and XBUFTBL is link-edited output from assembling your XBUF macros. Both members must exist in order to use XBUF caching.

For sample JCL, see S2K.R116.TEST.

EXECUTING XBUF SOFTWARE

This section describes the disposition for cache files when executing XBUF and discusses two execution parameters.

Disposition for Cache Files

Cache files and dual log files must have DISP=OLD; otherwise, SYSTEM 2000 Error Code 815 is issued. However, in-memory cache files (MEM and MEMX) do not require any JCL.

SYSTEM 2000 Execution Parameters

Two new execution parameters in Release 11.6 affect XBUF caching. The XBUF parameter turns the XBUF software on or off. The XBUFSUF specifies a suffix that indicates which XBUFTBL load module you want to use when you have assembled several of them. (See the *SYSTEM 2000 Product Support Manual* for details about setting the SYSTEM 2000 execution parameters.)

XBUF Execution Parameter The XBUF execution parameter activates the XBUF software and determines the type of XBUF caching that will be performed unless you specify a different runtype in your macros.

```
XBUF= |NO
      |STAT
      |YES
```

where

NO means the XBUF software is not activated. NO is the default.

STAT activates the XBUF software in a simulated mode where you can issue console commands that display statistics, without allocating any actual cache space or performing any real I/O. You can also model selected cache areas and specific files by setting various runtype options in the basic CACHE macro (see Chapter 4, "The CACHE Macro") or by coding the XBSTAT macro (see **The XBSTAT Macro** on page 2-8). You do not have to purchase or install any equipment for model caches. XBUF=STAT costs as much in CPU cycles and memory space as XBUF=YES. STAT and YES yield the same statistical results.

YES means actual caching takes place unless you specify otherwise for selected cache areas when you code your XBUF macros.

If the XBUF execution parameter equals YES or STAT, SYSTEM 2000 software looks for members S2KXBUF and XBUFTBL[suffix] in its run-time load libraries, allocated via the STEPLIB or JOBLIB DD statements. The delivery tape contains a default XBUFTBL module, which you can use to try XBUF=STAT. However, for XBUF=YES you must allocate real files, and you must code one or more cache macros. The XBUF execution parameter affects all cache areas that you have defined.

If you have created more than one XBUFTBL member, you must indicate which configuration you want to use by specifying the XBUFSUF execution parameter. SYSTEM 2000 software loads the members, and the XBUF software formats the cache files, if necessary, and sets up various tables in memory.

XBUFSUF Execution Parameter The XBUFSUF execution parameter allows you to activate a specific configuration of XBUF cache areas.

```
XBUFSUF=suffix
```

where

suffix is a suffix, appended to the XBUFTBL load module name. The suffix is any valid national character that is allowed in a PDS member name.

Optionally, you can create several configurations of XBUFTBL, each represented by a different module in the load library. In this situation, you must specify the new SYSTEM 2000 execution parameter XBUFSUF. The suffix indicates which configuration to activate. The member name for the XBUFTBL module must be in the format XBUFTBL[suffix].

Chapter 4

The CACHE Macro

BACKGROUND ON CACHING

The classification of electronic recording devices by speed is sometimes called the *hierarchy of memory.* Speed varies greatly among the different media. The hierarchy of memory is outlined below, with the fastest memory at the top of the list.

- main memory
- a group of high-speed, secondary storage devices, such as drums, fixed-head disks, and solid-state disks
- various kinds of ordinary movable-head disks
- tape drives.

Clearly, you should utilize the memory available at your computer site as efficiently as possible. Ideally, you would keep the most frequently accessed data on the fastest memory device. However, in most practical situations, very sophisticated software, firmware, hardware, and staff are required to take the best advantage of the hierarchy of memory available in your environment.

The concept of caching involves some type of buffering and overlapped I/O. This section discusses overlapped I/O in main memory for the CPU registers, IBM caching in main memory, and the relationship between the SYSTEM 2000 buffer manager and supplemental XBUF cache areas.

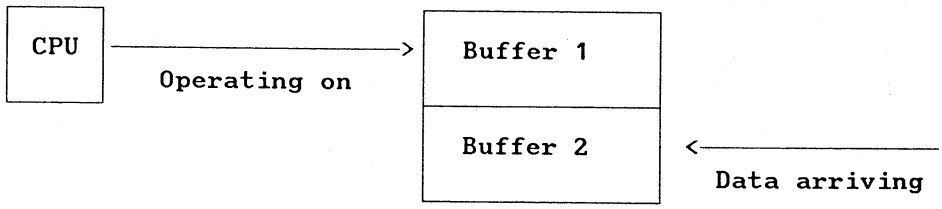
Buffering and Overlapped I/O

As mentioned above, the fastest and most costly memory available in a computer system is main memory, which is quite complicated. Registers within the central processing unit (CPU) provide the most expensive and fastest memory in the machine. Information in main memory must be fetched into the CPU registers before it can be processed arithmetically or logically.

Although data is transferred to the CPU registers in just a few instruction cycles, memory access usually overlaps other activity. That is, the CPU initiates a storage or retrieval operation against main memory and, without waiting for that operation to complete, performs arithmetic and logical operations on the last data it retrieved.

Transferring data from a relatively slow, distant device into a faster, closer device does not need to stop the CPU. The CPU can do other work while new data are arriving. The disadvantage is that the faster recording device must be large enough to hold two sets of data at a time: data that the CPU is actually working with and newer data that are arriving. Buffering isolates the CPU from the slow process of delivering data into its registers.

The illustration below shows a CPU operating on data in an area called Buffer 1, at the same time that Buffer 2 is being filled with data. When the CPU finishes its work with Buffer 1 and Buffer 2 is full, the CPU can work with Buffer 2 and let Buffer 1 fill up with the next group of data. If the CPU were much faster than the arrival process, more than two buffers would provide even greater efficiency.



Buffering is useful whenever data are moved. Buffering accommodates the difference in speeds between devices operating on different levels in the hierarchy of memory. Some IBM access methods provide automatic buffering; others do not. For example, the Basic Direct Access Method (BDAM), which SYSTEM 2000 software uses for data base files, is not automatically buffered; the problem program must manage by itself.

SYSTEM 2000 software has a powerful buffer manager, which retains the most recently referenced data base data in main memory. XBUF (Extended Buffer Manager) provides additional buffering capabilities, and it uses caching techniques. XBUF cache concepts for both IBM and XBUF are discussed next.

IBM Caching

In some cases, a buffer is called a *cache.* For example, main memory commonly has a subdivision called a cache, relatively small and very fast, which serves as a kind of buffer in CPU-memory access. The cache can be 8K or 16K or even larger. The IBM 3033 computer has a 64K cache memory.

The computer tries to predict what part of main memory it needs next and fetches that information into the cache, so that it is quickly available when required. Data actually travel from main memory into the cache and across the memory bus into the CPU registers. The 3033 CPU accesses data in 8-byte (doubleword) units, which is typical.

The buffer inside the 3033 CPU is a doubleword (8 bytes); the 3033 64K cache is organized into 8192 doublewords, each of which can be accessed by the CPU. Once it obtains a doubleword, the CPU usually operates on all of it. The CPU buffer typically contains interrelated or inseparable information. However, the cache is much larger and can contain many different kinds of data, not all of it necessarily pertinent.

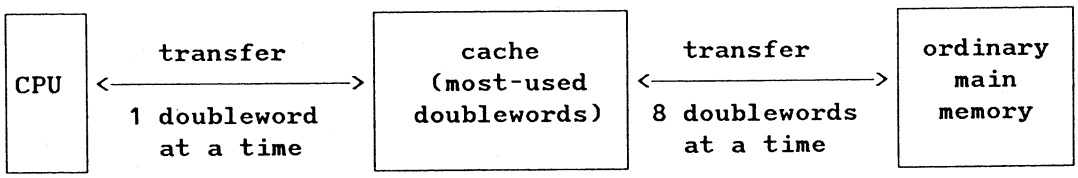
If the CPU incorrectly predicts the next data needed, the cache does not have the proper information when the CPU needs it. The CPU must fetch it from slow main memory into the cache. The computer transfers data from 3033 main memory to cache in increments of eight doublewords, anticipating that the next reference will be to the same doubleword or one of its seven neighbors, readily available in cache.

When eight doublewords are brought into cache, they replace the oldest (or least recently referenced) eight doublewords, which are dropped from the cache. The fetching of seven neighboring doublewords in anticipation of their probable usefulness is called *anticipatory buffering.*

If the CPU updates a doubleword that is in the cache, a *store-through* operation copies the new doubleword from the CPU to the cache and to main memory. If the doubleword is not in cache, only main memory is updated.

Suppose main memory is four million bytes and the cache is sixty-four thousand. If memory access were completely random, the CPU would find the correct doubleword less than two percent of the time. Therefore, the success of the cache depends on memory access not being random. The tendency of successive memory references to be close to one another, so that the CPU can correctly predict them, is called *locality of reference.*

Here is an overview of CPU and memory architecture.



XBUF Caching

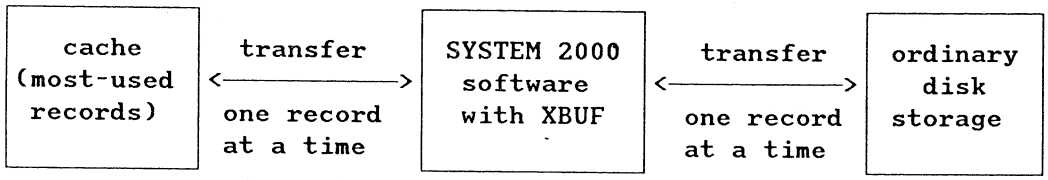
XBUF allows you to take advantage of 31-bit addressable memory for cache areas. You can also set up XBUF cache files on a relatively low-capacity, high-speed device in a system of slower, movable-head disk drives.

XBUF keeps the most active records of specific data base files in the specified cache area. When the SYSTEM 2000 buffer manager wants to read or write a record to secondary storage, XBUF intercepts the I/O request if it is for a data base file that you have associated with a cache area.

For a read operation, XBUF inspects some control blocks and diverts the read to the cache area if the record is there. If the record is not in the cache, it must be retrieved from disk. (Note: the record is not added to the cache at this point.) For a write operation, a store-through operation is performed. The disk is updated and then the cache. If the record is not in the cache, it is added to the cache, replacing the oldest record.

A record can be added to cache in another way. Remember that SYSTEM 2000 software has a buffer manager, which keeps several records in main memory. Eventually, one record too many must be read into main memory, and the contents of the oldest buffer must be replaced (dropped). If XBUF is enabled, the buffer manager calls it to add the contents of the oldest buffer in main memory to the cache.

XBUF cache files stand between the SYSTEM 2000 buffer manager and the operating system access methods.



XBUF caching and IBM memory caching are different. For XBUF caching, a direct path between the cache and the movable-head disks does not exist. Everything flows through SYSTEM 2000 software and XBUF, which reside in the mainframe.

For cache areas designated with real I/O, every update operation writes both to disk and to cache, regardless of whether the record is present in cache. If the record is not in cache, it is added, replacing the oldest record. If the record is in cache, it is updated in place.

XBUF is more efficient when most I/O operations are reads because XBUF turns all disk writes into dual writes, making them much slower than reads. Dual writes are overlapped. Therefore, excessive updating slows the system down to the slower disk speed. Both of the dual writes are initiated before either write is checked for completion.

MUSTATS and XBUF console commands allow you to display reads-per-minute, writes-per-minute, and drops-per-minute for SYSTEM 2000 pool buffers, data base files, and cache records. Remember that a drop occurs for the oldest buffer in a pool or the oldest record in cache when the pool or cache is full and needs to add another record. If writes-per-minute consistently exceed reads-per-minute, perhaps some of the disk files being cached should be removed from cache.

OVERVIEW OF THE CACHE MACRO

The CACHE macro allows you to tailor the type of XBUF processing for your environment. This macro offers options that are quite powerful and complex in allowing you to control caching operations. However, defaults provide a workable configuration for getting started, and you can fine-tune it later.

The CACHE macro allows you to specify relationships between DDnames. That is, you specify a one-to-many relationship between each DDname of a cache file and the corresponding DDnames of data base files that share the cache file. For example, consider the following CACHE macro statement.

```
CACHE (FAST1,PERSONN1,PERSONN4,PERSONN5,PERSONN6),          C
      (FAST2,LIBRARY6,PERSONN2,PERSONN3)
```

This statement generates a Cache Table, which SYSTEM 2000 software loads dynamically at initialization time. It specifies that disk files with DDnames PERSONN1, PERSONN4, PERSONN5, and PERSONN6 should share the available space in the cache file allocated by DDname FAST1. The cache file FAST2 is dedicated to disk files LIBRARY6, PERSONN2, and PERSONN3. This sample CACHE macro statement is complete and correct, but many more options and parameters are also available.

The CACHE macro accepts up to 256 cache files, and each cache file can serve up to 256 disk data base files. Each DDname appearing in the CACHE macro is a real DDname that either is in your run-time JCL or will be dynamically allocated. Therefore, no duplicates are allowed in the CACHE macro DDnames.

Note: you cannot mix the front-end macros with this CACHE macro.

THE CACHE TABLE

As mentioned above, the CACHE macro defines a one-to-many relationship between each cache file and its corresponding data base files. Each cache file has one or more disk files that share the cache file, but each disk file belongs to only one cache file. In a list of DDnames, the first DDname is always for the cache file, and the following DDnames are for the disk data base files.

The CACHE macro generates a Cache Table. XBUF stores the parameters and options in this internal table and uses them in its caching operations. You can reset many of these values during job execution with XBUF console commands (see Chapter 6, "XBUF Console Commands").

The CACHE macro statement below has two cache files; FAST1 and FAST2 are their DDnames. PERSONN1, PERSONN4, PERSONN5, and PERSONN6 are disk file DDnames that share the FAST1 cache file. LIBRARY6, PERSONN2, and PERSONN3 are disk file DDnames that share cache file FAST2.

You can also assign various attributes to cache and disk files in a parenthetical list after the DDname. This example illustrates the format for assigning 32 as an option for cache file FAST1 and two options (40, 100) for disk file PERSONN3. Cache file options are explained in detail in **Cache File Options** on page 4-16. Disk file options are discussed in **Disk File Options** on page 4-26.

```
CACHE (FAST1,(32),PERSONN1,PERSONN4,PERSONN5,PERSONN6),    C
      (FAST2,LIBRARY6,PERSONN2,PERSONN3,(40,100))
```

Cache Table	
Cache Files	Disk Files
FAST1 (32)	PERSONN1 PERSONN4 PERSONN5 PERSONN6
FAST2	LIBRARY6 PERSONN2 PERSONN3 (40,100)

SUMMARY OF THE CACHE MACRO SYNTAX

The general format for specifying CACHE macro parameters is shown below. The "C" in Column 72 is the continuation character, which indicates that the CACHE macro statement continues on the next line.

```
Column 10       Column 16                          Column 72
↓               ↓                                  ↓
CACHE           parameters                         C
```

Parameters is a list of one or more of the CACHE parameters separated by commas. The list of parameters is given below. Defaults are underscored. For clarity here, the delimiter within the parenthetical options is the slash.

```
DEL= |COMMA
     |SLASH
     |PERIOD

DSECT= |NO
       |YES

TIME= |YES
      |NO

SECI=nn

SWAP= |YES
      |NO

CNTIN=jjj

LTDROPC=kkk

HTDROPC=lll

RRAT=mmm

( cacheDDname,  [ ( me / bk / mp / |REAL  / |XBUF1 ) ] ,
                                   |STAT    |XBUF0
                                   |NONE

                diskDDname, [ ( lp / hp / |REAL ) ] , ...
                                          |STAT
                                          |NONE

                SYS, [ ( sysnum / lp / hp / |REAL ) ] )
                                            |STAT
                                            |NONE
```

SUMMARY OF CACHE MACRO FUNCTIONS

The following summary of CACHE macro parameters and options refers you to specific sections of this chapter for specific topics.

CACHE Macro Functions		
Function	Parameters	Page Reference
Changing the delimiter	DEL	4-9
Producing a DSECT	DSECT	4-9
Timing and I/O statistics	TIME SECI	4-10
Swapping disk files in and out of cache	SWAP, CNTIN, RRAT, LTDROPC, HTDROPC	4-11
Setting up cache files	Cache file DDnames in-memory cache MEM and MEMX DDname Cache file options multiple entry groups model block size model file size runtype maptype, dual logging Disk file DDnames data base files SYS DDnames Disk file options space priorities runtype SYS file options	4-14 4-16 4-24 4-26

CHANGING THE DELIMITER: DEL PARAMETER

The DEL parameter allows you to specify a comma, a slash, or a period as the delimiter within the parenthetical options after a DDname. Comma is the default.

```
DEL= |COMMA
     |SLASH
     |PERIOD
```

DEL does not affect CACHE macro syntax outside the parenthetical options following a DDname. That syntax is the domain of the IBM assembler. The Cache Table created by the CACHE macro does not depend on the DEL parameter. Alternate delimiters are provided only for readability.

The example below specifies DEL=SLASH.

```
CACHE DEL=SLASH,                                              C
      (FAST1,(////XBUF1),LIBRARY1,(//STAT),LIBRARY2),         C
      (FAST2,(////XBUF0),PERSONN1),                           C
      (FAST3,(///NONE),SYS,(100///NONE))
```

Notice how much more readable that example is than the following one, which uses the comma as the delimiter.

```
CACHE (FAST1,(,,,,XBUF1),LIBRARY1,(,,STAT),LIBRARY2),         C
      (FAST2,(,,,,XBUF0),PERSONN1),                           C
      (FAST3,(,,,NONE),SYS,(100,,,NONE))
```

PRODUCING A DSECT: DSECT PARAMETER

DSECT=YES produces a DSECT only. This labeled layout of the CACHE macro fields can be helpful when trying to analyze performance or a problem that requires reading a dump.

```
DSECT= |NO
       |YES
```

DSECT=YES bypasses and overrides all other CACHE macro parameters and specifications. The default is DSECT=NO.

XBUF TIMING PARAMETERS: TIME AND SECI

The TIME and SECI parameters determine whether XBUF I/O monitoring is activated and the interval at which rates and counters are calculated.

TIME Parameter

The TIME parameter turns on XBUF timing so that you can display various I/O statistics with console commands.

```
TIME= |YES
      |NO
```

The default for TIME is YES. You can display or reset the TIME parameter dynamically at run time with console commands (see **Displaying and Resetting the TIME Parameter** on page 6-6).

TIME=NO prevents XBUF from constantly issuing the store clock (STCK) instruction and avoids the other overhead necessary for recording I/O rates. If you are not planning to use the console commands, specify TIME=NO, for example, when generating Cache Tables for a single-user jobs, which never allow console commands. For TIME=NO, XBUF skips most of its timing activity, which conserves some CPU cycles. All I/O rates remain equal to zero during execution.

Even if you specify TIME=NO, XBUF still issues enough STCKs to monitor cache file drops-per-minute, so XBUF swap analyses can determine when a cache file is busy. Drops-per-minute is a very small fraction of overall timing activity, which in turn is only a small fraction of cache management.

SECI Parameter

SECI is the minimum interval in which rates are recalculated. The actual interval is determined by the first XBUF read, write, or drop that occurs after SECI seconds have elapsed since the last recalculation. An XBUF read, write, or drop must occur to trigger the recalculation. The default for SECI is 10 seconds.

```
SECI=nn
```

Computing the rates takes some time, and you would not see much difference in the overhead between using, say, SECI=10 instead of SECI=60. Too much other activity is usually taking place in a live MVS system for SECI to impact performance. Still, you do not want to use CPU cycles needlessly. So, if you intend to examine system performance only occasionally with console commands, set SECI equal to 60, its maximum value. (SECI=1 is the minimum value.) Regardless of SECI, the XBUF I/O rates are always calculated and reported in terms of occurrences-per-minute.

You can set SECI with the CACHE macro only. SECI cannot be displayed or reset with console commands.

SWAPPING DISK FILES IN CACHE AREAS

XBUF tables and algorithms are designed to keep the most recently read (not the most recently referenced) records for a designated group of cached disk files. As used here, a record means a page in a data base file. Paging means I/O activity against the cache.

Periodically, XBUF reviews its various counters to see whether the group of disk files being cached needs to be adjusted. Adjustment is done by setting flags in XBUF control blocks that quiesce or activate cache I/O for a disk file. *Swapping* means taking a whole file out of the cache or putting one into the cache. You can control swapping through various parameters in the CACHE macro discussed here.

In a Multi-User environment, you can also display and reset swapping parameters with online XBUF console commands (see **Controlling the Swapping Parameters** on page 6-7).

SWAP Parameter

As mentioned before, XBUF reviews its own performance periodically to see if it can improve throughput for a cache area by swapping one of its disk files in or out of active participation in caching. XBUF swaps the disk files internally the same way that you reset runtypes with console commands. For a disk file that seems to be causing problems, XBUF simply sets the file's runtype to NONE. Later, if the file's I/O patterns become more favorable, XBUF can reset the runtype from NONE to REAL, which turns on actual caching for the file.

XBUF cannot change runtype from NONE to REAL if a console command or the CACHE macro set it to NONE. However, if XBUF swaps the runtype from REAL to NONE, XBUF can consider swapping it back to REAL. This runtype is displayed as NS, which means NONE-system-swapped. Swapping ignores any file with runtype STAT, and it never automatically resets a REAL or NONE runtype to STAT.

```
SWAP= |YES
      |NO
```

SWAP=YES is the default because swapping usually helps. However, you can disable XBUF swapping. To suppress swapping entirely, specify SWAP=NO in the CACHE macro. Swapping can also be turned on and off with console commands (see **Controlling the Swapping Parameters** on page 6-7).

If you are using priorities to show favoritism among data bases, you might discover that swapping takes out the very data base that should benefit most from caching. In this situation you might want to suppress swapping. Swapping does not consider the low and high percentage priorities.

The swap process ignores dual logged files. Because dual logging has only one disk file for each cache file, XBUF does not need to decide which disk files are contributing the most to throughput.

CNTIN Parameter

Swapping is a complex process. If you decide to allow swapping, you can control the interval at which XBUF reviews its performance. Performance might improve if you suppress swapping or if you vary the interval at which XBUF reviews its status.

```
CNTIN=jjj
```

CNTIN means *count interval*, which determines the number of I/Os that XBUF performs for a disk file before doing a swap analysis. CNTIN is an integer from 100 to 32,767, inclusive; the default is 500. XBUF is aware of only I/Os for files that were specified in the CACHE macro or files that it picks up as SYS files. (See **Specifying Disk Files for a Cache File** on page 4-24 for details about SYS files.)

Separate count intervals are maintained for each disk file. If CNTIN = 500, XBUF does a swap analysis after each 500th I/O performed on each disk file. If LIBRARY5 experiences 1,000 I/Os in the same time period that LIBRARY6 experiences 50,000 I/Os, LIBRARY6 goes through swap analysis many more times.

LTDROPC, HTDROPC and RRAT Parameters

XBUF uses LTDROPC, HTDROPC, and RRAT during its swap analysis to see whether a disk file should be swapped in or out. TDROPC means *target drop criteria* and signifies a level of busyness for cache files, as measured by drops-per-minute. LTDROPC is the *lowest target drop criteria* and HTDROPC is the *highest target drop criteria*. Together they indicate a range of acceptable drops-per-minute between which no swaps are done.

```
LTDROPC=kkk

HTDROPC=lll

RRAT=mmm
```

The default for LTDROPC is 400. The default for HTDROPC is 600. LTDROPC and HTDROPC can be displayed and reset dynamically with XBUF console commands (see **Controlling the Swapping Parameters** on page 6-7).

RRAT means read ratio, which lets you set the ratio of reads to total I/O. RRAT separates disk files that are good for caching from those that are not. RRAT is not accessible through console commands; it can be set only with the RRAT parameter in the CACHE macro. The default for RRAT is 75.

The table below shows how XBUF uses drops-per-minute rates during swap analysis.

Drops-per-minute rate is less than LTDROPC	Drops-per-minute rate is between LTDROPC and HTDROPC	Drops-per-minute rate is greater than HTDROPC
Look for a file to swap in. (Cache drop rate is too low.)	Do no swapping. (Cache drop rate is satisfactory.)	Look for a file to swap out. (Cache drop rate is too high.)

Swapping NONE to REAL

XBUF swaps (resets) a disk file runtype to REAL only if the cache file is not dropping pages at a very high rate. A cache file that is already dropping pages rapidly is too busy to handle another disk file; that would only make things worse. LTDROPC is the lowest limit of acceptable drops-per-minute; the default is 400.

Suppose LTDROPC = 400 and the cache file is dropping pages at a rate of 500/minute. XBUF does not reset any disk file runtype to REAL because the cache is already acceptably busy and does not need any more work.

Even if the cache file is not busy, the disk file might not appear to be a good candidate for caching. For example, suppose that most of the last 500 I/Os were updates. You can set the ratio of reads to total I/Os (RRAT) that you feel are acceptable before a file should be swapped from NONE to REAL. RRAT is expressed as a percentage, and the default is 75.

If 400 of the last 500 I/Os were reads, the reads represent 80 percent of the total. XBUF treats this disk file as a suitable candidate for swapping, but another test takes place at that point. XBUF inspects the other disk files in the cache file to see if any have a more favorable read ratio than the current one. If so, the better disk file will come up for analysis soon, and XBUF waits for it. XBUF swaps a disk file to REAL only if it is the best choice among all disk files that have currently been swapped to NONE.

Swapping REAL to NONE

Suppose the cache file is not busy. That is, the drops-per-minute rate does not exceed the value of HTDROPC. XBUF does not swap any disk file runtypes in that cache area to NONE.

The cache is considered busy if the drops-per-minute rate exceeds HTDROPC. If the cache is busy, XBUF tries to swap the least desirable disk file from REAL to NONE; otherwise, XBUF does nothing and waits for the worst file to come up for analysis.

REAL disk files contribute more information to swap analyses than NONE disk files. A fairly accurate calculation of relative desirability for a disk file can be done. XBUF records the number of reads and writes for the disk file and the number of reads, writes, and drops for the cache file associated with the disk file. Reads are good; writes and drops are not good.

Here is the desirability formula:

```
DESIRE  = (INDBRIN * 100) / (INDBDIN + 1)
```

where

INDBRIN is the number of reads-on-cache for this disk file in the last count interval (CNTIN I/Os).

INDBDIN is the number of drops-on-cache for this disk file in the last count interval.

INDBDIN remains 0 until the cache fills up since no dropping or swapping occurs until the disk files begin to compete for cache space. Also, INDBRIN is incremented only when the cache is full and actually dropping pages. XBUF does not check reads-on-cache done in the last CNTIN I/Os.

SETTING UP CACHE FILES

Each cache file is associated with one or more disk data base files. The cache file DDname and its associated disk file DDnames are positional parameters, enclosed in parentheses. The first DDname in the parenthetical list is always the cache file DDname. The succeeding DDnames in the list are disk file DDnames of data base files that share the cache file. (See **Specifying Disk Files for a Cache File** on page 4-24 for details about disk file DDnames.) The delimiter within the parenthetical options is the slash.

```
( cacheDDname,  [ ( me / bk / mp / |REAL  / |XBUF1 ) ] ,
                                   |STAT    |XBUF0
                                   |NONE

                diskDDname, [ ( lp / hp / |REAL ) ] , ...
                                          |STAT
                                          |NONE

                SYS, [ ( sysnum / lp / hp / |REAL ) ] )
                                            |STAT
                                            |NONE
```

The example below illustrates the general syntax of the CACHE macro. Detailed explanations with more examples are given in subsequent sections. LIBRARY1 is a disk file DDname belonging to the cache file FAST1. The parenthetical information (//STAT) is associated with LIBRARY1. The slash is the delimiter; the first two options use default settings, and the third option is set to STAT. LIBRARY2 also belongs to the FAST1 cache file. The FAST2 cache file is for dual logging of PERSONN1. FAST3 is a special SYS cache file, which holds several data base files, as they are encountered in the JCL.

```
CACHE DEL=SLASH,                                                    C
      (FAST1,(////XBUF1),LIBRARY1,(//STAT),LIBRARY2),               C
      (FAST2,(////XBUF0),PERSONN1),                                 C
      (FAST3,(///NONE),SYS,(100///NONE))
```

Cache File DDnames

Specifying cache file DDnames FAST1, FAST2, and so on, is only a coding convention. The word FAST is not required by the system.

XBUF has one restriction for cache file DDnames: if they begin with MEM, they are treated as in-memory buffer requests; they are not external disk file DDnames. (See next topic for details about in-memory cache buffers.)

The only other restrictions for cache file DDnames are the rules established by IBM for valid DDnames. However, you should probably establish a convention that is easy to work with. For example, you could begin all cache DDnames with the letters FAST. To display the status of all cache files whose DDnames begin with FAST, simply enter

```
SEND XBUF D FAST* *
```

This console command would select all FAST cache files for viewing, unless you also had a data base file name starting with FAST. (In that case, you should begin your cache file DDnames with a different prefix, such as XBUF or CACHE.)

If you plan to use console commands very often, avoid the following reserved words as cache file DDnames: TIME, SWAP, TDROPC, LTDROPC, and HTDROPC. These words are not formally restricted because using them does not cause any problem in caching. However, it does prevent you from displaying activity for a cache file with one of those names.

For instance, suppose you have a cache file named TIME and you give the following command to display cache activity:

```
SEND XBUF D TIME *
```

SYSTEM 2000 software displays the setting of the TIME parameter, without checking to see whether a cache file DDname TIME exists.

Note: the DDname TIME would not conflict with disk file DDnames because those must be seven or eight characters, for example, TESTXXX, PERSONN1, or LIBRARY8.

You can also assign optional attributes to a cache file or disk file. These options are positional subparameters enclosed in parentheses after the appropriate DDname. The parentheses distinguish the options from the next DDname. (See **Cache File Options** on page 4-16 for cache file options and **Disk File Options** on page 4-26 for disk file options.)

In-Memory Cache Files: MEM and MEMX DDnames

XBUF allows you to bring a substantial part of even a large data base into main memory and maintain the most active pages there, without excessive CPU overhead. As mentioned earlier, XBUF is designed for large amounts of data, as opposed to the SYSTEM 2000 buffer manager, which works better with smaller volumes. When more than 200 buffers are in use in a given pool, SYSTEM 2000 software spends excessive amounts of CPU time searching its buffers to see which one is the oldest. On the other hand, testing XBUF with over 12,000 buffers showed very little measurable CPU overhead.

The ideal situation would be to give SYSTEM 2000 software only enough buffers to avoid deadlocks, and let XBUF take care of the rest. SYSTEM 2000 buffers are in main memory, and you can set up in-memory cache buffers with the XBMEMORY macro (see **The XBMEMORY Macro** on page 2-4) or with the basic CACHE macro as explained here.

To set up in-memory cache buffers, use the special cache file DDname MEM or MEMX in the CACHE macro. If a cache file DDname begins with the letters MEM, XBUF does not expect to find the DDname in the JCL allocating an external device. Instead, other parameters specify the number and size of in-memory buffers that XBUF should allocate. These buffers are supplements to the SYSTEM 2000 buffers.

In-memory buffers offer another advantage because SYSTEM 2000 software cannot address buffers above 16 meg, even when running under MVS/XA. But XBUF, being somewhat isolated, can address them, if the special cache file DDname begins with MEM or MEMX.

Both MEM and MEMX buffers are acquired with a GETMAIN request to the operating system. However, XBUF asks for MEMX buffers with a special XA parameter, LOC=(ANY,ANY). The LOC parameter authorizes the operating system to satisfy the storage request wherever it can, above or below the 16 meg boundary.

For details about the size and number of in-memory buffers, see **Options 2 and 3: Block Size and File Size** on page 4-18.

CACHE FILE OPTIONS

The cache file DDname is the first subparameter in its list. After a cache file DDname, you can specify five options, enclosed in parentheses. If you want to use defaults for all options, omit the entire set of parenthetical information, which is boldfaced below. The delimiter is the slash.

```
( cacheDDname,  [ ( me / bk / mp / |REAL  / |XBUF1 ) ] ,
                                   |STAT    |XBUF0
                                   |NONE

                diskDDname, [ ( lp / hp / |REAL ) ] , ...
                                          |STAT
                                          |NONE

                SYS, [ ( sysnum / lp / hp / |REAL ) ] )
                                            |STAT
                                            |NONE
```

The five options (subparameters) following the cache file DDname allow you to specify: multiple entry groups, block size, file size, runtype (REAL, STAT, or NONE), and maptype for dual logging (XBUF0 or XBUF1).

The system recognizes an option according to its position within the parentheses. To use defaults for the rightmost options in the list, simply omit them. To use defaults for the leftmost options in the parenthetical list, you must specify the delimiter to signify the missing (default) options. For example, (////XBUF0) tells XBUF to use defaults for the first four options and to assign the specific value XBUF0 to the last option.

Option 1: Multiple Entry Groups

The first positional option for a cache file DDname is the multiple entry group specification. It determines how XBUF maps disk file pages into a cache area and determines the precision with which XBUF manages the pages. The value is a number that is a power of two between 1 and 256, inclusive. The default is 1.

This value of this option is a constant during execution; it can be varied only by reassembling the CACHE macro. You cannot change it with console commands.

If the multiple entry option equals 1, XBUF operates at the individual record level. That is, exactly one cache file record is acquired for each disk file record that needs to be kept in cache. If XBUF needs to add a disk file record to a full cache, only one cache record (the oldest one) is dropped.

If it is greater than 1, XBUF allocates cache space in groups of records. For example, if it equals 8, XBUF manages cache space in groups of eight records. Every record belongs to a group of eight records, and cache space is acquired and dropped in groups of eight contiguous records. That is, eight contiguous disk file pages map into the eight contiguous cache records. Anticipatory fetching of pages from disk to cache does not occur. Disk file pages are written to and read from cache one at a time.

Suppose the multiple entry option equals 4. When SYSTEM 2000 software needs to add a record to a full cache, the following activity occurs:

1. XBUF checks to see whether a group of four records already has space available in cache. If so, space for the new record is already reserved, and it is written into its proper location among the four. If an old copy of the new page exists, it is overlaid (dropped).

2. If the check in Step 1 fails, the entire oldest group of records, which can be one to four records, is dropped from cache, and that space is used to receive the new record.

If the pattern of I/O against a cache shows good locality of reference, fairly high values for this option can be specified without much degradation. If the I/O is dispersed across many different groups of pages, the drops-per-minute rate for the cache usually begins to rise, which means records are not being retained in cache long enough to be found again.

The multiple entry option should be regarded primarily as a mechanism for conserving main memory. Space requirements are discussed more completely in Chapter 7, "XBUF Main Memory," but, generally speaking, memory requirements for XBUF tables can be cut roughly in half by doubling the value of this option.

This option provides a way to trade precision of cache space management for main memory space. The smaller the value, the more main memory XBUF needs in order to keep track of that cache file, and the more accurately XBUF handles paging.

Sample multiple entry specifications:

```
CACHE (FAST1,(128),LIBRARY)
CACHE (FAST1,(16),PERSONN5,PERSONN6)
```

Options 2 and 3: Block Size and File Size

The second and third positional options allow you to specify block size (bk) and file size (mp) in two situations: for in-memory cache areas (real or simulated) and for simulated disk cache files. Statistical modeling and in-memory cache areas do not require a real disk cache file. When the cache file does not exist, you must specify the block size and number of records in the options after the cache file DDname. For real disk cache files, block size and file size come from the JCL.

The second positional option is the block size (bk), which must be a valid SYSTEM 2000 page size. The default model block size is 0. (See the *SYSTEM 2000 Product Support Manual* for a list of supported page sizes.)

The third positional option is file size (mp), which specifies the number of records in the cache file. This number is not checked for any maximum when the CACHE macro is generated. Specifying a very large number (say, over a million) usually causes failure at run time. The default number of records is zero.

When you specify XBUF=YES in your SYSTEM 2000 execution parameters, XBUF looks at the cache file name in the CACHE macro. If the cache file name begins with MEM or MEMX, the block size and file size options are required. If the cache file name does not begin with MEM OR MEMX, the block size and file size options are ignored.

The sample CACHE macro statement below requests XBUF to model the behavior of a cache file with 10,000 2492-byte records (blocks).

```
CACHE (FAST1,(/2492/10000),LIBRARY),DEL=SLASH
```

If you specify XBUF=STAT in the SYSTEM 2000 execution parameters, XBUF builds tables for a 10,000-record cache of 2492-byte records, ensures that valid SYSTEM 2000 pools are available, and does everything except actually read and write on the nonexistent cache file.

XBUF console commands are available to display read, write, and drop rates. These rates are probably lower than an actual high-speed device could produce, but they are still useful.

For example, you might find that the LIBRARY data base is too large and that the I/O against it is too randomly dispersed. If so, the drop rate for cache file FAST1 would be high compared to the read and write rate. You could bring the system down, reassemble the CACHE macro with a file size of 20,000 records, and try again. Remember that you can model an XBUF cache system without purchasing an actual high-speed cache device.

The example below requests XBUF to acquire central memory for 1,000 2492-byte records, dedicated to the LIBRARY data base. Approximately 2.5 meg of 24-bit virtual memory is needed for the in-memory cache area.

```
CACHE (MEM01,(/2492/1000),LIBRARY),DEL=SLASH
```

The next example shows a request for XBUF to acquire central memory for 10,000 4060-byte records. Here you could dedicate 40 meg of virtual memory to buffers for the first 100 data base files that happen to be required in cache. Notice the use of a cache file DDname beginning with MEMX, which causes XBUF to acquire and access memory in the XA manner.

```
CACHE (MEMX001,(/4060/10000),SYS,(100)),DEL=SLASH
```

Option 4: Runtype (REAL, STAT, and NONE)

The fourth positional option is runtype. Specifying different runtypes for different cache files merely allows different kinds of modeling, monitoring, and real I/O to take place within the same system.

The runtype options are REAL, STAT, and NONE. The default is REAL. If you specify NONE for the cache file, the runtypes of all its associated disk files become NONE. If you specify STAT for the cache file, any of its disk files having runtype REAL become STAT. Cache file runtype REAL does not override any disk file runtype.

See **Priority of runtypes** on page 4-20 for a summary of the three methods of setting runtypes and the order of precedence among runtypes for cache files and their disk files.

Runtype NONE The meaning of runtype NONE as a cache file option is different than specifying XBUF=NO in the SYSTEM 2000 execution parameters. When XBUF=NO is specified as a SYSTEM 2000 execution parameter, XBUF executable code is never loaded. No XBUF activity occurs, and the entire CACHE macro is irrelevant.

However, if you specify YES or STAT in the XBUF execution parameter, NONE becomes meaningful as an option after a cache file DDname. The NONE option means that all disk files whose DDnames follow that cache file DDname should be monitored, but they do not actually participate in cache I/O or table management. The NONE option provides a convenient, inexpensive way to monitor many data base files.

Suppose you do not want actual caching to take place, but you want to see if you have a data base I/O bottleneck. The example below shows how you can set up I/O monitoring for 100 disk files with no CPU or memory overhead for real cache management. SYS disk file runtypes default to REAL in the CACHE macro, but the Cache Table shows that by execution time they are overridden with the NONE option, which is shown here for cache file FAST1.

```
CACHE (FAST1,(///NONE),SYS,(100)),DEL=SLASH
```

Cache Table	
Cache Files	Disk Files
FAST1 (1/0/0/NONE/XBUF1)	SYS1 (0/100/NONE) SYS2 (0/100/NONE) . . . SYS100 (0/100/NONE)

A cache file DDname having the NONE option behaves the same whether you specify XBUF=YES or XBUF=STAT in your execution parameters. If a cache file DDname has runtype NONE, the disk files in that cache file have runtype NONE also; runtypes specified for those disk files are ignored.

Runtype STAT If you specify STAT for a cache file, all of its disk files that have runtype REAL or STAT are treated as STAT; that is, the disk files are models only. Disk files with NONE runtype are treated as NONE.

Runtype REAL You can specify runtype REAL as the fourth positional option for any cache file. (REAL is also the default.) REAL does not override any runtypes specified for the disk files. However, a REAL cache file can be downgraded to STAT or NONE at run time through the XBUF execution parameter. If so, the disk files are downgraded accordingly.

Real caching implies opening and formatting the cache files, acquiring space for tables, and true caching of frequently read records in the designated cache area.

Priority of runtypes You can set runtype with options (or defaults) at three different levels:

- in the XBUF execution parameter when SYSTEM 2000 software is initialized (see **XBUF Execution Parameter** on page 3-2)
- as an option after a cache file DDname in the basic CACHE macro (see **Option 4: Runtype (REAL, STAT, and NONE)** on page 4-19)
- as an option after a disk file DDname in the basic CACHE macro (see **Runtype Option for Disk File DDnames** on page 4-28).

If not specified after a DDname, the default runtype for a cache file or disk file is REAL. Also, within certain restrictions, you can reset runtypes with console commands while the system is running.

The order of precedence for the different runtypes is as follows:

- The XBUF execution parameter downgrades the runtype specified for a cache file DDname to the level of the XBUF execution parameter.
- The runtype specified for a cache file DDname downgrades the caching level specified for its disk file DDnames.
- Disk file runtypes do not override anything. They can be overridden by the runtype for their cache file or by the runtype set by the XBUF execution parameter.

XBUF=NO in the execution parameter is the lowest level of caching. NO means that the XBUF executable code is not loaded, so any macros you have coded are irrelevant.

XBUF=STAT is a simulation of caching, causing all cache files to be modeled only. No I/O is done, and the cache files are not opened or formatted for BDAM I/O. Therefore, STAT in the execution parameter overrides any REAL runtype specified in the CACHE macro. Also, you cannot change the runtype of files to REAL with console commands. Both NONE and STAT runtypes are allowed (not overridden) in the CACHE macro, and files can be reset from NONE to STAT with a console command.

XBUF=YES, which means real caching, does not override anything. Unless the CACHE macro requests otherwise, cache files are opened and formatted, and caching occurs with real I/O. You can reset cache file runtypes to STAT or NONE with console commands at any time.

The following tables show the order of precedence and the resulting runtype (RT) for disk file DDnames, given any of the 27 possible combinations of

- the value of the XBUF execution parameter
- runtype (RT) given with cache file DDname
- runtype (RT) given with disk file DDname.

Table 1: XBUF=YES Execution Parameter			
DDname Runtype	Disk RT=REAL	Disk RT=STAT	Disk RT=NONE
Cache RT=REAL	REAL	STAT	NONE
Cache RT=NONE	NONE	NONE	NONE
Cache RT=STAT	STAT	STAT	NONE

Table 2: XBUF=STAT Execution Parameter			
DDname Runtype	Disk RT=REAL	Disk RT=STAT	Disk RT=NONE
Cache RT=REAL	STAT	STAT	NONE
Cache RT=NONE	NONE	NONE	NONE
Cache RT=STAT	STAT	STAT	NONE

Table 3: XBUF=NO Execution Parameter			
DDname Runtype	Disk RT=REAL	Disk RT=STAT	Disk RT=NONE
Cache RT=REAL	<na>	<na>	<na>
Cache RT=NONE	<na>	<na>	<na>
Cache RT=STAT	<na>	<na>	<na>

Suppose the XBUF execution parameter equals STAT, the cache file DDname has runtype REAL, and the disk file DDname has runtype REAL. Table 2 shows XBUF=STAT possibilities. The first row in Table 2 shows cache DDnames with runtype REAL; the first column shows what happens to disk file DDnames with runtype REAL. XBUF initialization routines reset these disk file runtypes to STAT.

Table 3 is the trivial case of not loading or using XBUF because the SYSTEM 2000 execution parameter is XBUF=NO.

Option 5 - Dual Logging: Maptype (XBUF1 and XBUF0)

The fifth positional option after a cache file DDname is the maptype. The value for maptype is either XBUF0 or XBUF1. A cache file with XBUF1 maptype can have one or more disk files sharing the same cache. Unless the cache file is for dual logging, let the cache file maptype default to XBUF1.

If you specify XBUF0, the cache file will be used for dual logging. That is, XBUF copies one entire disk file to a dual logged cache file when the data base is opened. Updates to this cache file are done in synch with the disk file updates. XBUF does one-for-one mapping for dual logged files and does not need a table to know where each page resides.

The following rules apply to dual logging:

- You must specify XBUF0 as the fifth option after the cache file DDname.
- You can have only one disk file in the cache file.
- Only REAL and NONE runtypes are allowed for the cache file and its single disk file.
- The cache file primary space must be equal to or larger than the disk file.

Consistency among the first three rules is verified when the CACHE macro is generated; errors suppress macro generation. The last rule is verified when the data base is opened. If the cache file is not large enough, runtype is set to NONE and a nonfatal WTO message appears.

Here are some sample maptype requests. The first example specifies dual logging for data base file LIBRARY6 in cache file FAST1. Cache file FAST2 includes Files 1 through 6 for the PUBLISH data base. The slash is the delimiter.

Sample maptype requests:

```
CACHE (FAST1,(////XBUF0),LIBRARY6),                              C
      (FAST2,(////XBUF1),PUBLISH),DEL=SLASH
```

The next example specifies runtype NONE for dual logging of LIBRARY6. The FAST2 cache file has real dual logging for PUBLISH1. The slash is the delimiter.

```
CACHE (FAST1,(///NONE/XBUF0),LIBRARY6),                          C
      (FAST2,(///REAL/XBUF0),PUBLISH1),DEL=SLASH
```

Advantages of dual logging Dual logging allows quick recovery because XBUF keeps the disk file and the cache file synchronized; therefore, two complete, identical copies of the file should always exist. If a catastrophic hardware error occurs, either the disk file or its dual log in cache usually survives.

Also, if you have enough space on the cache device for an entire disk file, dual logging saves the main memory and CPU time that XBUF would otherwise spend on the problem of caching the best possible disk file pages.

Candidate files for dual logging The Update Log (File 7) and the Rollback Log (File 8) used in SYSTEM 2000 Coordinated Recovery are good candidates, as well as data base File 1, which contains general data base information. Dual logging does not improve performance, but it does offer an additional safety measure. If data base File 1, 7, or 8 is damaged through a hardware error, rollback usually fails; the data base must be restored from the latest available backup tape. However, if XBUF keeps a duplicate copy of Files 1, 7, and 8, you can recover the data base simply by changing the JCL to allocate the cache file copy of the damaged file.

The Rollback Log block size equals the largest data base file block size, plus twelve. For example, if all files in a data base have a 2492 block size, the block size for the Rollback Log (File 8) would be 2504 bytes. This Rollback Log would fit in a 2492-byte pool because SYSTEM 2000 software reserves space for the 12-byte header.

You must code the DD statement correctly for the cache file that is to accommodate the Rollback Log. The example below assumes the largest block size for the six PUBLISH data base files is 2492.

```
CACHE (FAST1,(////XBUF0),PUBLISH8), . . .

//FAST1 DD UNIT=2305-2,DCB=BLKSIZE=2504, . . .
```

If your Rollback Logs occasionally go through file extension, see Chapter 5, "Data Base File Extension and XBUF Caching" for details about data base file extension and XBUF caching.

Note: the Rollback Log (File 8) is very heavily updated. Do not specify it as an XBUF1 file to improve performance, because the excessive writes cause the average I/O to occur at disk speed.

Also, you can use MUSTATS console commands to identify small, highly active disk files that might benefit from dual logging.

SPECIFYING DISK FILES FOR A CACHE FILE

Each cache file is associated with one or more disk data base files. The cache file DDname and its associated disk file DDnames are positional parameters, enclosed in parentheses. The first DDname in the parenthetical list is always the cache file DDname. Succeeding DDnames are disk file DDnames of data base files that share the cache file.

```
( cacheDDname,  [ ( me / bk / mp / |REAL  / |XBUF1 ) ] ,
                                   |STAT    |XBUF0
                                   |NONE

                diskDDname, [ ( lp / hp / |REAL ) ] , ...
                                          |STAT
                                          |NONE

                SYS, [ ( sysnum / lp / hp / |REAL ) ] )
                                            |STAT
                                            |NONE
```

In the example below, the slash is the delimiter. LIBRARY1 and LIBRARY2 are disk file DDnames belonging to the cache file FAST1. The parenthetical options (//STAT) are associated with LIBRARY1; the first two options use defaults, and the third option is set to STAT.

```
CACHE DEL=SLASH,                                                    C
      (FAST1,(////XBUF1),LIBRARY1,(//STAT),LIBRARY2),               C
      (FAST2,(////XBUF0),PERSONN1),                                 C
      (FAST3,(///NONE),SYS,(100///NONE))
```

Data Base Disk File DDnames

The CACHE macro accepts up to 256 cache files. One cache file can serve up to 256 disk files. Each disk file is a SYSTEM 2000 data base file. Each disk file DDname is eight characters, representing a specific data base file, for example, LIBRARY1, TEST4, PUBLISH5, or PUBLISH6.

You can also specify a 7-character DDname when you want an entire data base to share one cache file. The CACHE macro allows you to refer to the DDnames of the six data base files generically, by specifying only the first seven characters of the data base name. For example, PERSONN is treated by the CACHE macro as though you specified PERSONN1, PERSONN2, PERSONN3, PERSONN4, PERSONN5, PERSONN6. The Update Log (File 7) and the Rollback Log (File 8) are not included in the cache file because they tend to be written more than read.

Note: the block size of each disk file must be equal to the block size of the cache file it is sharing. If you have disk files with different block sizes and you want them to benefit from XBUF caching, you must group the files by block size and assign each group to a cache file with that block size. However, a disk file does not have to have the same number of physical blocks as a cache file.

SYS Disk File DDname

The special DDname SYS is the only valid disk file DDname shorter than seven characters. SYS is a broad, generic name that allows XBUF to add data base files to the specified cache file regardless of their real DDnames. SYS reserves space in the Cache Table to hold whatever data base files are opened at run time that have not been specifically designated for caching.

```
( cacheDDname,  [ ( me / bk / mp / |REAL  / |XBUF1 ) ] ,
                                   |STAT    |XBUF0
                                   |NONE

                diskDDname, [ ( lp / hp / |REAL ) ] , ...
                                          |STAT
                                          |NONE

                SYS, [ ( sysnum / lp / hp / |REAL ) ] )
                                            |STAT
                                            |NONE
```

The rules for using the SYS DDname are shown below.

- SYS can be included in a cache file that has other disk file DDnames.
- SYS can be specified in its own separate cache file.
- Up to 256 data base files can be added with SYS. The files are included in the designated cache file when each data base is opened.
- Several cache files can have the SYS DDname. XBUF determines which cache file is the least active and puts the newly opened data base files in the best cache file.

SYS should never be used to cache every file in the system. Usually, high-speed buffer space is too expensive for that. The SYS DDname allows modeling to take place on a demand basis with whatever data base files are opened. SYS allows run-time monitoring of files, without having to hard-code every DDname that interests you.

DISK FILE OPTIONS

One or more disk file DDnames that share a cache file follow the cache file DDname. You can give three options after each disk file DDname. Four options can follow the special DDname SYS. The disk file options are enclosed in parentheses. If you want to use all the defaults, simply omit the parenthetical options.

```
( cacheDDname,  [ ( me / bk / mp / |REAL  / |XBUF1 ) ] ,
                                   |STAT    |XBUF0
                                   |NONE

                diskDDname, [ ( lp / hp / |REAL ) ] , ...
                                          |STAT
                                          |NONE

                SYS, [ ( sysnum / lp / hp / |REAL ) ] )
                                            |STAT
                                            |NONE
```

The system recognizes an option according to its position within the parentheses. To use defaults for rightmost options in the list, simply omit them. To use defaults for the leftmost options in the parenthetical list, you must specify the delimiter to signify the missing (default) options. The delimiter here is the slash. For example, (//NONE) tells XBUF to use defaults for the first two options and assign the specific value NONE to the third option.

If you specify parenthetical options after a 7-character data base DDname, those options apply to all six data base files. For example, specifying LIBRARY,(//STAT) is the same as specifying LIBRARY1, LIBRARY2, and so forth, each followed by (//STAT).

Disk File Space Priorities

The first two positional options after a disk file DDname allow you to specify the low and high percentages of cache file space that you want assigned to a disk file. Since these options are percentages, they must be numbers from 0 to 100. Also, the high percentage (default of 100) must be greater than the low percentage (default of 0). You can reset these priority percentages with XBUF console commands at any time.

The high percentage keeps a disk file from attempting to force a drop of other files if it already has as much space in cache as it is allowed. This action is enforced regardless of whether the cache is full.

XBUF never allows a disk file to use more space than you have specified in the high percentage, regardless of its I/O activity. After a disk file exceeds the low percentage of space, XBUF tries to prevent it from going below that amount of space (regardless of I/O activity) until the disk file is closed. That is, XBUF tries to keep the space for the file until the data base is closed.

XBUF ignores the low percentage option until the cache file fills completely and the disk files begin competing for space. Only then can a disk file page be dropped from the cache file when a newer page needs to be written to cache.

If a disk file cannot tolerate having a page (or multiple entry group of pages) dropped without going below the low percentage of space, XBUF allows a drop only when the new page belongs to a file that is also below its low percentage of space. If the drop is not allowed, the file that was trying to force the drop is not updated in the cache file. If the drop is allowed, the older page is overlaid and the older file falls below your requested low percentage of the cache space.

Suppose you specify low percentage as 30 and high percentage as 70 for a group of six data base files. Obviously, all six files cannot have 30% of the cache space at any given time; the cache has only 100% of itself to give.

Instead, the files are constrained somewhat by the priority percentages, but they can still compete for cache space on a demand basis. In the current example, after a file reaches 30%, it cannot force drops except for other files that have also achieved 30%. As long as a file remains below the low percentage, it can force drops for any other files.

All low percentages for disk files in a cache file do not have to total a certain number, nor do all the high percentages. In fact, all low percentages can default to 0, and all high percentages can default to 100. The low and high percentages serve only as target minimums and maximums.

If you want to show favoritism to a specific disk file (or group of files), you can give it an advantage over others in competition for cache space by using the low and high percentage options.

Suppose you have a very important data base that is seldom used. If you want to give it more than its fair share of performance improvement from the cache, set its low and high percentages high. XBUF attempts to retain as much of the files on cache as it is told, even if usage goes to zero. As long as the files stay open, they are always available for fast retrieval.

As mentioned before, when the number of users accessing a data base goes to zero (unless a special zap is applied), SYSTEM 2000 software physically closes the data base files. XBUF frees all their cache space for use by other data bases. These actions take place regardless of low and high percentages assigned.

While priorities have their advantages, they also prevent XBUF from servicing I/O requests on a strict demand basis. System throughput is impacted by your priority specifications because you are preventing new pages from being added to cache when they need to be. However, the mechanism is lenient; no disk file can take over the system or lock up cache space as a result of your low or high percentage settings.

On the other hand, the priority mechanism does not give you too much authority. Even if you want to allow some file to lock up cache space, priority does not do it; that is not the intent of priorities. You need to assign each disk file to its own individual cache file if you want to obtain complete control over the cache space.

Sample priority specifications:

```
CACHE (FAST1,LIBRARY,(40/100))
CACHE (FAST1,LIBRARY2,(60/100),LIBRARY5,(/66))
CACHE (FAST1,LIBRARY,(75),PERSONN)
```

Runtype Option for Disk File DDnames

The third option after a disk file DDname is one of the familiar NONE, REAL, or STAT runtypes. The default is REAL.

As explained previously, any disk file runtype can be downgraded by the XBUF execution parameter or by the runtype of the cache file to which it belongs. The runtype of a disk file does not alter any other runtype. See **Priority of runtypes** on page 4-20 for a summary of the three methods of setting runtypes and the order of precedence among runtypes for cache files and their disk files.

Since STAT processing is meaningless with a dual log cache file, you cannot specify STAT for a disk file DDname that is participating in dual logging.

Within certain restrictions, you can reset runtype with console commands while the system is running. For example, if a particular file seems to have excessive writes, you could reset its runtype to NONE and free the cache space for other disk files that might be better able to use it.

You can never upgrade a disk file runtype to REAL unless its cache file runtype was REAL at initialization time. Only the XBUF initialization routines can format REAL cache files for I/O.

Dual logging has some restrictions when you reset the disk file runtype with console commands. A dual log cache file must maintain a complete copy of its disk file. Suppose you specify runtype REAL for a dual logged cache file. XBUF formats the cache file for real I/O, and the system begins running. Then suppose you reset runtype to NONE with a console command. Immediately, caching stops, dual logging stops, and updates affect only the disk file. Therefore, XBUF cannot guarantee that the contents of the cache file exactly match the contents of the disk file.

This situation is acceptable, but you cannot reset that runtype to REAL because XBUF cannot assume anything about cache contents. The only way you can reset runtype to REAL is to ensure that the data base file is physically closed. (Unless you apply a special zap, SYSTEM 2000 software closes data base files when the number of users of that data base goes to zero.)

After the file is closed, you can give a console command to reset runtype to REAL. When the data base is subsequently opened, XBUF will copy the entire disk file to the cache file, and the contents of both files will match.

SYS Disk File Options

Four options can follow the disk file DDname SYS. The options specified (or their defaults) apply to all data base files included in the SYS file.

```
( cacheDDname,  [ ( me / bk / mp / |REAL  / |XBUF1 ) ] ,
                                   |STAT    |XBUF0
                                   |NONE

                diskDDname, [ ( lp / hp / |REAL ) ] , ...
                                          |STAT
                                          |NONE

                SYS, [ ( sysnum / lp / hp / |REAL ) ] )
                                            |STAT
                                            |NONE
```

The first option for SYS files is sysnum. It pertains to SYS DDnames only. Options 2, 3, and 4 following sysnum are the same as disk file options lp, hp, and runtype.

Sysnum is the number of Cache Table slots you want to reserve for data base files that SYSTEM 2000 software opens and that are not hard-coded with specific DDnames in the CACHE macro.

Usually NONE runtype is the best runtype for SYS files. If you specify REAL, XBUF attempts to do real cache I/O for whatever data base files happen to be opened first, unless their DDnames are among those you hard-coded in the CACHE macro.

Specify runtype NONE for the SYS file if you want to monitor a large number of data base files without having to code all the necessary DDnames in the CACHE macro.

Because one cache file can handle only 256 disk files, sysnum cannot exceed 256. The default for sysnum is 1. However, you can specify SYS for more than one cache file, thereby spreading data base files over several cache files. XBUF chooses the least active cache file and adds the data base file accordingly.

Notice in the example on the next page that the CACHE macro invents DDnames for SYS files: SYS1, SYS2, and so on. These DDnames are only placeholders and do not correspond to DDnames in your JCL; they are overlaid with real DDnames when XBUF opens the files. Console commands do not display files that have never been opened, so you do not see SYS1 or SYS2 while displaying file activity. Instead, you see the actual DDnames that have overlaid SYS1, SYS2, and so on.

```
CACHE (FAST1,(///NONE),SYS,(100)),DEL=SLASH
```

Cache Table	
Cache Files	Disk Files
FAST1 (1/0/0/NONE/XBUF1)	SYS1 (0/100/NONE) SYS2 (0/100/NONE) . . . SYS100 (0/100/NONE)

If you specify XBUF=YES in your execution parameters, cache files named in the CACHE macro (for example, FAST1) usually require a DDname in the run-time JCL. However, the example below illustrates an exception. To minimize JCL, you can assign NONE to the cache file rather than to the individual SYS files. The initialization routines recognize that you are not using a real cache file, and a DDname is not required.

Sample SYS specifications:

```
CACHE (FAST1,SYS,(24///NONE))          (requires FAST1 DDname in
                                         run-time JCL if XBUF=YES
                                         parameter was specified)

CACHE (FAST1,(///NONE),SYS,(24) )      (does not require FAST1
                                         DDname in run-time JCL)
```

DEFAULTS AND LIMITS FOR CACHE MACRO PARAMETERS

Here is a summary of the defaults, minimums, and maximums for the CACHE macro parameters and file options.

Parameter/Option	Default if Omitted	Minimum	Maximum
DEL=	COMMA	-	-
DSECT=	NO	-	-
TIME=	YES	-	-
SECI= nn	10	1	60
SWAP=	YES	-	-
CNTIN=jjj	500	100	32767
LTDROPC=kkk	400	0	32767
HTDROPC=lll	600	0	32767
RRAT=mmm	75	0	100
me	1	1	512
bk	0	2016	14616
mp	0	1	2147483647
runtype	REAL	-	-
maptype	XBUF1	-	-
lp	0	0	100
hp	100	1	100
sysnum	1	1	256

SAMPLE CACHE MACRO STATEMENTS AND CACHE TABLES

Here are some sample CACHE macro statements.

```
CACHE (FAST1,LIBRARY1)

CACHE (FAST1,LIBRARY1,LIBRARY2)

CACHE (FAST1,LIBRARY),TIME=NO

CACHE (FAST1,LIBRARY),(FAST2,PUBLISH)

CACHE DEL=SLASH,(FAST1,(////XBUF0),PUBLISH8))

CACHE DEL=SLASH,                                                    C
      (FAST1,LIBRARY),(FAST2,(4/2492/5000/STAT),PUBLISH)

CACHE (MEM01,(1,6440,1200,REAL),EMPLOYE)

CACHE (FAST1,LIBRARY),SWAP=NO,                                      C
      (FAST2,PERSONN,PUBLISH,(50/)),                                C
      (FAST3,(////XBUF0),EMPLOYE8),DEL=SLASH

CACHE (FAST1,(///NONE),SYS,(50)),DEL=SLASH

CACHE (FAST1,(64),EMPLOYE),CNTIN=100,RRAT=50,LTDROPC=100

CACHE (FAST1,(64/6440/1000/STAT),EMPLOYE),                          C
      LTDROPC=50,HTDROPC=1200,SECI=60,                              C
      (FAST2,LIBRARY,(60/90),SYS,(24///NONE),PUBLISH),              C
       DEL=SLASH
```

General Format of the Cache Table

The form and semantics of the CACHE macro have already been discussed. Here is a template for a general Cache Table. Later examples show how different CACHE macro statements would fill in the template to produce Cache Tables that are tailor-made for different applications.

Cache Table										
System-Wide Parameters DEL= CNTIN= SWAP= TIME= LTDROPC= HTDROPC= RRAT= SECI=										
Cache Files	ME	BK	MP	RT	MT	Disk Files	#	LP	HP	RT

The Cache Table assigns tags to the positional options specified in the CACHE macro. The first option for a cache file is its multiple entry group number, abbreviated ME. The second and third cache file options are model block size (BK) and model number of records (MP). RT means runtype, and MT means maptype.

For disk files, the column heading # applies to SYS files only. The value displayed is the first option after the SYS DDname. It is the number of data base files that you want XBUF to automatically allocate to a particular cache file. LP is the low priority percentage, and HP is the high priority percentage. RT is the disk file runtype.

Example 1: Caching an Entire Data Base

Suppose you want to optimize the performance of the LIBRARY data base by allowing its six data base files to share one cache file. This example shows the minimal CACHE macro for this task and the resulting Cache Table. Defaults are used for the system-wide parameters. Because the disk file DDname is only seven characters (LIBRARY), XBUF includes data base Files 1 through 6 in the FAST1 cache file.

```
CACHE (FAST1,LIBRARY)
```

Cache Table

System-Wide Parameters
DEL=COMMA CNTIN=500 SWAP=YES TIME=YES
LTDROPC=400 HTDROPC=600 RRAT=75 SECI=10

Cache Files	ME	BK	MP	RT	MT	Disk Files	#	LP	HP	RT
FAST1	1	0	0	REAL	XBUF1	LIBRARY1	-	0	100	REAL
						LIBRARY2	-	0	100	REAL
						LIBRARY3	-	0	100	REAL
						LIBRARY4	-	0	100	REAL
						LIBRARY5	-	0	100	REAL
						LIBRARY6	-	0	100	REAL

Example 2: Caching Only Five Data Base Files

After using the cache arrangement shown in Example 1 for awhile, suppose you discover that the Hierarchical Table (LIBRARY5) seems to experience too much random activity and updating in your application environment. You decide to remove it from the cache. Here are the minimal CACHE macro parameters and the resulting Cache Table. You specify each individual LIBRARY disk file, omitting LIBRARY5.

```
CACHE (FAST1,LIBRARY1,LIBRARY2,LIBRARY3,                C
       LIBRARY4,LIBRARY6)
```

Cache Table

System-Wide Parameters
DEL=COMMA CNTIN=500 SWAP=YES TIME=YES
LTDROPC=400 HTDROPC=600 RRAT=75 SECI=10

Cache Files	ME	BK	MP	RT	MT	Disk Files	#	LP	HP	RT
FAST1	1	0	0	REAL	XBUF1	LIBRARY1	-	0	100	REAL
						LIBRARY2	-	0	100	REAL
						LIBRARY3	-	0	100	REAL
						LIBRARY4	-	0	100	REAL
						LIBRARY6	-	0	100	REAL

Example 3: Caching Two Data Bases

Now suppose you want to add the PUBLISH data base to the caching system. But before doing real caching, you want to try runtype STAT to look at the I/O activity patterns for the PUBLISH data base. All PUBLISH data base files are added to the FAST1 cache file. The CACHE macro parameters and resulting Cache Table are shown below.

```
CACHE DEL=SLASH,                                          C
      (FAST1,LIBRARY1,LIBRARY2,LIBRARY3,                  C
      LIBRARY4,LIBRARY6,                                  C
      PUBLISH,(//STAT))
```

Cache Table

System-Wide Parameters

DEL=SLASH CNTIN=500 SWAP=YES TIME=YES
LTDROPC=400 HTDROPC=600 RRAT=75 SECI=10

Cache Files	ME	BK	MP	RT	MT	Disk Files	#	LP	HP	RT
FAST1	1	0	0	REAL	XBUF1	LIBRARY1	-	0	100	REAL
						LIBRARY2	-	0	100	REAL
						LIBRARY3	-	0	100	REAL
						LIBRARY4	-	0	100	REAL
						LIBRARY6	-	0	100	REAL
						PUBLISH1	-	0	100	STAT
						PUBLISH2	-	0	100	STAT
						PUBLISH3	-	0	100	STAT
						PUBLISH4	-	0	100	STAT
						PUBLISH5	-	0	100	STAT
						PUBLISH6	-	0	100	STAT

Example 4: Giving Priorities to Cached Data Bases

Now suppose that the user community for the LIBRARY data base is much more important than the users for the PUBLISH data base. Still, you definitely want to try runtype REAL for PUBLISH. You decide to restrict PUBLISH so that none of its files can ever use more than twenty percent of the cache. Since statistics indicate that LIBRARY File 1 is read very frequently, you decide to give it maximum access to the cache file by assigning LIBRARY1 a low percentage of 100.

The CACHE macro parameters and Cache Table are shown below. The delimiter is changed to the slash.

```
CACHE DEL=SLASH,                                    C
      (FAST1,LIBRARY1,(100),                        C
      LIBRARY2,LIBRARY3,                            C
      LIBRARY4,LIBRARY6,                            C
      PUBLISH,(/20))
```

Cache Table

System-Wide Parameters

DEL=SLASH CNTIN=500 SWAP=YES TIME=YES
LTDROPC=400 HTDROPC=600 RRAT=75 SECI=10

Cache Files	ME	BK	MP	RT	MT	Disk Files	#	LP	HP	RT
FAST1	1	0	0	REAL	XBUF1	LIBRARY1	-	100	100	REAL
						LIBRARY2	-	0	100	REAL
						LIBRARY3	-	0	100	REAL
						LIBRARY4	-	0	100	REAL
						LIBRARY6	-	0	100	REAL
						PUBLISH1	-	0	20	REAL
						PUBLISH2	-	0	20	REAL
						PUBLISH3	-	0	20	REAL
						PUBLISH4	-	0	20	REAL
						PUBLISH5	-	0	20	REAL

Example 5: Specifying Dual Logging

By analyzing many displays from console commands, you notice that File 1 pages stay in the cache area during normal application work loads, and response time improves. You decide to save the resources required to manage the aging tables for LIBRARY1 by making it a dual logged file. So, you move it to a different cache file, FAST2, which is dedicated to LIBRARY1. The CACHE macro parameters and Cache Table are shown below.

```
CACHE DEL=SLASH,                                    C
      (FAST1,                                       C
      LIBRARY2,LIBRARY3,                            C
      LIBRARY4,LIBRARY6,                            C
      PUBLISH,(/20)),                               C
      (FAST2,(////XBUF0),LIBRARY1)
```

Cache Table

System-Wide Parameters
DEL=SLASH CNTIN=500 SWAP=YES TIME=YES
LTDROPC=400 HTDROPC=600 RRAT=75 SECI=10

Cache Files	ME	BK	MP	RT	MT	Disk Files	#	LP	HP	RT
FAST1	1	0	0	REAL	XBUF1	LIBRARY2	-	0	100	REAL
						LIBRARY3	-	0	100	REAL
						LIBRARY4	-	0	100	REAL
						LIBRARY6	-	0	100	REAL
						PUBLISH1	-	0	20	REAL
						PUBLISH2	-	0	20	REAL
						PUBLISH3	-	0	20	REAL
						PUBLISH4	-	0	20	REAL
						PUBLISH5	-	0	20	REAL
						LIBRARY6	-	0	20	REAL
FAST2	1	0	0	REAL	XBUF0	LIBRARY1	-	-	-	REAL

Chapter 5

Data Base File Extension and XBUF Caching

When SYSTEM 2000 data base files become full, their initial physical extent boundaries, as requested in the primary space allocation in the JCL, can no longer handle your data. SYSTEM 2000 software then closes the file as a BDAM file and opens it as a BSAM (sequential) file. It formats another extent if the space is physically available on the disk volume. Then the file is closed and reopened as a BDAM file.

Extension is associated with updates and with a data base that is growing beyond your predictions. File extension is not desirable for a retrieval-biased production environment because the data base is unavailable to your applications during extension. Still, extension does provide some emergency padding in case your data base suddenly grows unexpectedly.

If a disk data base file that is cached needs extension, the following actions occur:

1. SYSTEM 2000 software closes the file as a BDAM file.

2. XBUF intercepts the close operation and drops all cached disk pages from the cache.

3. SYSTEM 2000 software opens the file as BSAM, writes on it, and closes it again. XBUF ignores all sequential file activity.

4. SYSTEM 2000 software opens the file as BDAM.

5. XBUF intercepts the open operation.

If the file has not grown by more than 256 records, no adjustment to the tables is necessary. If it has, a control block called the Disk Page Table (DPT), explained in **XBUF Control Block Tables** on page 7-2, must be discarded and a new DPT must be created. Normally, this action is quick and transparent to you. However, if XBUF cannot acquire memory for a new DPT, a WTO message is issued and runtype is set to NONE.

Usually XBUF copies the entire contents of a dual logged file from disk to cache during open processing. Copying is suppressed for extension because the file is still intact, and a complete copy exists on both disk and cache.

For a dual logged file, the primary space for the cache file must be large enough to handle any extensions that occur on its disk file. The exact amount of space does not have to be equal, but the cache primary space must always have as many (or more) pages as the disk file.

Chapter 6

XBUF Console Commands

With XBUF console commands you can display information about cache I/O and you can display and reset certain parameters in the CACHE macro. Also, when XBUF is activated, the MUSTATS DBN and POOLS console commands display data base and pool statistics. For details about MUSTATS commands, see *Technical Report: S2-105 Changes and Enhancements to SYSTEM 2000 Software, Release 11.6 under IBM OS and CMS.*

After XBUF displays information or performs a reset, a trailer line of output appears. It shows you how many cache files and disk files were successfully displayed or reset according to your DDname specification.

When you are working with the Cache Tables, isolate the DDnames that interest you. You do not need to distinguish between cache file and disk file DDnames syntactically; XBUF recognizes the difference.

You can also use the common notation of an asterisk as a "wild card" in some parts of the syntax. If you enter only an asterisk where XBUF expects to find a DDname, XBUF behaves as though you repeated the command once for every DDname in the XBUF caching system. If you enter a series of characters followed by an asterisk, XBUF assumes you want the command carried out for all DDnames beginning with those characters.

XBUF CONSOLE COMMAND SYNTAX

Here is the general format for XBUF console commands. The asterisk (instead of SEND XBUF) is a fast, abbreviated format.

```
|SEND XBUF  command
|*  command
```

If you submit XBUF commands from the master console under MVS, you must precede each command with the following:

```
|MODIFY  jobname,
|F  jobname,
```

where jobname is the job name of the Multi-User system. Alternate consoles do not need these identifiers. For example,

```
F $MU, *D TIME                    master console under MVS
MODIFY $MU, SEND XBUF D TIME      master console under MVS
*D TIME                           alternate console
SEND XBUF D TIME                  alternate console
```

If you are using S2OP at an alternate console, you can use the plus sign (+) to repeat the previous XBUF command. In fact, the plus sign repeats any Multi-User console command, for example, the D A S command. The + must be the only character on the line. Also, it cannot be used from the master console.

DISPLAYING CACHE I/O INFORMATION

You can display two kinds of information for cache files and cached disk files:

- the rates of reads, writes, and drops (RATES command)
- the pages being cached at the moment (PAGES command).

The syntax for XBUF display commands in a Multi-User environment is shown below.

```
|SEND XBUF |DISPLAY  |*          |*
|*         |D        |DDname     |RATES
                                  |PAGES
```

A DDname with an asterisk as a suffix means all files beginning with the given characters. An asterisk instead of a DDname means all files. An asterisk instead of RATES or PAGES produces both reports. XBUF knows (from its internal tables) whether you have specified cache files or disk files.

I/O rates vary during normal work loads, and XBUF intentionally keeps only approximations. In effect, each rate is rounded to a precision of one event per minute. Rounding minimizes the CPU time and main memory overhead for monitoring. However, rounding errors occasionally become apparent when a cache rate, such as reads-per-minute for the cache file, is compared to individual disk file rates for the cache. For example, the disk file reads-per-minute might be 10, 20, and 30 while the cache reads-per-minute displays as 61 or 59 instead of the expected 60.

Similarly, when swapping occurs, it might take a moment for the rate counters to reflect the new situation. A disk file that is swapped out by XBUF acquires a runtype of NS (NONE system-swapped), and its read and write rates do not appear in the console displays until the internal counters for the cache file have caught up. The SECI parameter determines how frequently the rates are recomputed.

examples

```
F $MU, *DISPLAY LIB* *          master console under MVS
F $MU, *D PERSONN8 *            master console under MVS
*D * *                          alternate console
SEND XBUF DISPLAY FAST* RATES   alternate console
```

RATES Output

The examples below display rate information for all DDnames beginning with FAST. (By convention, DDnames beginning with FAST are cache files.)

```
F $MU, * D FAST* RATES          master console under MVS
* D FAST* RATES                 alternate console
```

The following information appears in the RATES output:

DBRPM is the rate of reads-per-minute for disk data base files.

DBWPM is the rate of writes-per-minute for disk data base files.

XBRPM is the rate of cache file reads-per-minute.

XBWPM is the rate of cache file writes-per-minute.

XBDPM is the rate of cache file drops-per-minute.

For cache files, only XBRPM, XBWPM, and XBDPM rates appear. For disk data base files, DBRPM and DBWPM appear. If no XBUF I/O occurred in the last interval (SECI), the data is considered too old to be meaningful and asterisks are displayed. For details on calculating rates, see **SECI Parameter** on page 4-10.

If the computer environment is CPU-bound with MVS running under VM, the I/O rates are rather low in comparison to those you can expect with native MVS. Even so, the numbers are genuine and approximately consistent with what you will see.

Dual logged files show zero DPTs (Disk Page Tables), because they require no mapping and have no DPTs. Other cache areas might have zero DPTs if they are inactive, that is, if all their disk files were closed. For details about DPTs, see **XBUF Control Block Tables** on page 7-2.

The number of writes to cache (XBWPM) for a disk file can be a positive number, even though the number of writes to disk (DBWPM) is zero. This situation does not contradict the statement that all writes are dual writes. All writes to disk cause writes to cache, but the opposite is not true. Writes to cache might occur in response to reads from disk if they result in SYSTEM 2000 software dropping its oldest buffer. XBUF intercepts the buffer drop process and writes the buffer contents to cache storage. Also, some retrievals cause XBUF to drop buffers and, therefore, prompt XBUF to write them to cache.

PAGES Output

The sample commands below display page information for the LIBRARY data base files.

```
F $MU, SEND XBUF D LIBRARY* PAGES    master console under MVS
SEND XBUF D LIBRARY* PAGES           alternate console
```

PAGES output for cache files Two lines of output appear for each cache file DDname that qualifies, according to the DDname specification in your console command.

ME is the multiple entry number. The default is 1.

RT is the runtype.

R means REAL.
S means STAT.
N means NONE.
RS means XBUF swapped the disk file to REAL.
NS means XBUF swapped the disk file to NONE.

BK is the block size.

MEMFAIL is the number of times XBUF needed more memory if attempts to acquire memory for tables failed.

TOTAL is the number of pages in the cache file.

IN USE is the number of pages in use.

DPTS is the number of Disk Page Tables (DPTs) in use. DPTS is a measure of how widely scattered the I/O requests have been. For more details about DPTS, see **XBUF Control Block Tables** on page 7-2.

PAGES output for disk files Two lines of output appear for each disk file DDname that qualifies according to the DDname specification in your console command.

The first line begins with the letter C or the letter O.

C means the disk file is currently closed.
O means the disk file is currently open.

Other fields displayed are described below.

RT is the runtype.

R means REAL.
S means STAT.
N means NONE.
RS means XBUF swapped the disk file to REAL.
NS means XBUF swapped the disk file to NONE.

DISPLAYING AND RESETTING THE TIME PARAMETER

The XBUF console commands allow you to display and reset the TIME parameter. XBUF timing is a different function than SYSTEM 2000 timing. The CPU and elapsed time that SYSTEM 2000 software records for job accounting, QAEXIT/QASTAT, and the SCF TIMING ON command are completely separate from XBUF timing.

To display the current setting of the TIME parameter, use this syntax:

```
|SEND XBUF    |DISPLAY  TIME
|*            |D
```

To reset the TIME parameter, use the following syntax:

```
|SEND XBUF    |RESET    |TIME=      |YES
|*            |E        |TIMING=    |NO
```

TIME=YES enables XBUF timing. TIME=NO disables timing. The default is YES. Note: you cannot display or reset the SECI parameter with console commands.

Suppose you want to avoid the overhead of continuous timing except when you are particularly interested in performance. In that case, you could generate the CACHE macro with TIME=NO, and whenever you wanted to display rate information, you could enable timing with * E TIME=YES. When you finish displaying rates, you can turn off the XBUF timing with * E TIME=NO.

XBUF calculates and reports timing information in events-per-minute. However, you specify the interval at which the internal counters and rate values are recomputed with the CACHE macro SECI parameter. If you disable timing during execution, it will take up to SECI seconds for XBUF to reset the counters to blanks.

examples

```
F $MU, * D TIME                    master console under MVS
F $MU, SEND XBUF E TIME=YES        master console under MVS
* D TIME                           alternate console
SEND XBUF DISPLAY TIME             alternate console
* E TIMING=NO                      alternate console
```

CONTROLLING THE SWAPPING PARAMETERS

Displaying and Resetting the SWAP Parameter

To display the current setting of the SWAP parameter, use the following syntax:

```
|SEND XBUF |DISPLAY |SWAP
|*        |D
```

You can also enable or disable file swapping with the following console command syntax:

```
|SEND XBUF |RESET   |SWAP= |YES
|*        |E               |NO
```

SWAP=YES enables swapping. SWAP=NO disables swapping. The default is YES.

examples

```
F $MU, * D SWAP                    master console under MVS
F $MU, SEND XBUF E SWAP=YES        master console under MVS
* D SWAP                           alternate console
SEND XBUF DISPLAY SWAP             alternate console
* RESET SWAP=NO                    alternate console
```

Displaying and Resetting Target Drop Criteria

You can display the current settings for low (LTDROPC) and high (HTDROPC) target drop criteria. LTDROPC defaults to 400. HTDROPC defaults to 600.

```
|SEND XBUF |DISPLAY  |LTDROPC     displays low target drop criteria
|*        |D        |HTDROPC     displays high target drop criteria
                    |TDROPC      displays both LTDROPC and HTDROPC
```

You can also reset LTDROPC and HTDROPC as long as LTDROPC is less than HTDROPC. You must issue two commands to reset both LTDROPC and HTDROPC.

```
|SEND XBUF  |RESET    |LTDROPC=nnn
|*         |E        |HTDROPC=mmm
```

examples

```
F $MU, SEND XBUF E LTDROPC=125     master console under MVS
MODIFY $MU, *E HTDROPC=300         master console under MVS
SEND XBUF RESET LTDROPC=100        alternate console
* E HTDROPC=150                    alternate console
```

XBUF CONSOLE COMMAND SYNTAX SUMMARY

Here is the general format for XBUF console commands.

```
|SEND XBUF  |DISPLAY  |operand  [operand]
|*          |D        |keyword

            |RESET     keyword    value
            |E
```

For the master console under MVS, you must precede each command with

```
|MODIFY  MUjobname,
|F
```

For example, to display the SWAP parameter value at the master console, enter

```
F $MU, * D SWAP
```

Summary of XBUF Display and Reset Commands

```
|SEND XBUF |DISPLAY  |LTDROPC
|*         |D        |HTDROPC
                     |TDROPC

                      SWAP

                      TIME

                     |*           |*
                     |DDname      |RATES
                                  |PAGES


|SEND XBUF  |RESET   |TIME=    |YES
|*          |E       |TIMING=  |NO

                      SWAP=     |YES
                                |NO

                     |HTDROPC=nnn
                     |LTDROPC=mmm

                     |*          |PRTY=kk/hhh
                     |DDname     |RT=  |N
                                       |S
                                       |R
```

If the DDname has less than eight characters and terminates with an asterisk, all DDnames beginning with the specified characters qualify for the console command. An asterisk by itself means "all" appropriate files in the XBUF caching system.

Chapter 7

XBUF Main Memory

S2KXBUF AND XBUFTBL LOAD MODULES

XBUF executable code is contained in one load module named S2KXBUF, which is loaded by SYSTEM 2000 at initialization time if the XBUF execution parameter is REAL or STAT. S2KXBUF occupies approximately 32K bytes of main memory.

The link-edited output from assembling your XBUF macros is a load module named XBUFTBL, which has an optional suffix if you create several different XBUFTBL configurations (see **XBUFSUF Execution Parameter** on page 3-3). XBUFTBL is loaded immediately after S2KXBUF. The XBUFTBL size in bytes can be determined by analyzing the assembly and the link-edit listings from its generation or by using the following formula:

```
SIZE = 832 + F*280 + D*224
```

where

F is the number of cache files specified.

D is the number of disk files specified.

For example, the following CACHE macro statement

```
CACHE (FAST1,PERSONN),(FAST2,LIBRARY,PUBLISH)
```

generates an 832-byte prefix, two 280-byte tables for cache files FAST1 and FAST2, and eighteen 224-byte tables for the three sets of six data base files. The total required space is 5360 bytes.

If all cache files and disk files have runtypes of NONE, S2KXBUF and XBUFTBL are sufficient. Dual logged files also require no space outside of the Cache Table. Additional control block tables are necessary to manage REAL or STAT cache files that are not dual logged. These tables are described next.

XBUF CONTROL BLOCK TABLES

XBUF creates three types of control block tables. Space for these tables is acquired dynamically and, once acquired, is never freed.

XBUFTBL Table

Each REAL or STAT cache file that is not used for dual logging, has an entry in the XBUFTBL table, which is acquired immediately after opening the file at system initialization time. If a cache file is dual logged or has runtype NONE, XBUFTBL table space is never acquired.

The XBUFTBL table is the map that shows where a particular disk file page resides in cache storage. Each multiple entry group requires eight bytes of XBUFTBL space. The position of each entry in XBUFTBL signifies the cache file page location, and the rightmost three bytes contain the corresponding disk file page number. So, by inspecting the entry, XBUF can identify the cache file page containing the disk file page. The other bytes contain the relative age of the cache file page and a unique identifier for the disk file DDname to which the entry is currently dedicated.

If the default arrangement of one cache file page per multiple entry (ME) group is used, one page of cache space requires one 8-byte entry.

The following formulas use integer arithmetic to compute the space required in bytes:

```
SIZE = ((PAGES + ME - 1) / ME) * 8
```

For example, given a cache file with 25,000 pages and the default ME of 1,

```
SIZE = ((25000 + 1 - 1) / 1) * 8 = 25000 * 8 = 200000 bytes
```

If 200,000 bytes is too large, you could specify ME = 16 to bring the space requirement down, as shown below.

```
SIZE = ((25000 + 16 - 1) / 16) * 8 = (25015/16) * 8 = 12504 bytes
```

Because the size calculation uses integer arithmetic, the remainder is discarded after division. Also, notice the order of precedence indicated by the parentheses when carrying out the operation.

Disk Page Table (DPT)

A Disk Page Table (DPT) is acquired and formatted for each REAL or STAT disk file that is not being dual logged. Each DPT entry contains information about 256 disk file pages. Each DPT entry is four bytes long. The formula below calculates the DPT size for one disk file:

```
SIZE = ((PAGES + 255) / 256) * 4
```

Suppose a cache file is shared by three disk files of the same data base that contain 200, 6,000, and 100,000 pages, respectively. When the data base is opened, XBUF calculates the space as follows:

```
SIZ1 = ((200 + 255) / 256) * 4 = ((455 / 256) * 4 = 4
SIZ2 = ((6000 + 255) / 256) * 4 = (6255 / 256) * 4 = 24 * 4 = 96
SIZ3 = ((100000 + 255) / 256) * 4 = (100255 / 256) * 4 = 1564
SIZE = SIZ1 + SIZ2 + SIZ3 = 1664 bytes
```

DPTs are not very large; therefore, the system does not always try to conserve them. When a data base is closed, the DPTs are not freed, and they are not available for reuse by other disk files. If a data base is reopened, the DPTs are reused if they are still large enough. If a data base was closed in order to restore a larger copy of it from tape or to perform file extension, some DPT might not be adequate. In that unusual situation, the old DPT is wasted, and a new one is acquired.

Each DPT entry is a 1-byte counter followed by a 3-byte address pointer into the next control block, the DSCT. During processing, when XBUF intercepts a SYSTEM 2000 request for a given disk file page, it inspects the DPT entry for the 256-page group of pages, which includes the page currently requested. If the DPT is all zeros, none of the 256 disk file pages (including the page currently requested) are in the cache area. If the DPT is not zero, XBUF must follow the pointer into the DSCT.

Disk Segment Control Table (DSCT)

The Disk Segment Control Table (DSCT) is the most complicated XBUF table, and it is very important. Each XBUFTBL has a chain of DSCTs. One is allocated and formatted at initialization time when XBUFTBL is set up, and others are acquired and chained to the first one later if necessary.

The following discussion uses the labels generated if you specify DSECT=YES in the CACHE macro to assist in dump analysis.

A DSCT is a set of 256-entry bitmaps that direct XBUF to the general vicinity in the XBUFTBL where the location of a given disk file page is recorded. Without the DSCT, XBUF would have to search the entire XBUFTBL for each disk file page, but the DSCT limits the search to a segment of no more than 128 entries. The number of entries in the XBUFTBL (DBENTS) is

```
DBENTS = (PAGES + ME - 1) / ME
```

The number of 128-entry segments (DBNUMSEG) in the XBUFTBL is

```
DBNUMSEG = (DBENTS + 127) / 128
```

DBIBITS is the number of bits required to count up to DBNUMSEG plus 1, relative to 0. DBIBITS is the number of bits needed for each disk file page to point to the proper part of the XBUFTBL, with one special bit configuration left over to indicate that the disk file page is not in the cache area at all. Two bits are required for DBNUMSEG=3, four bits are required for DBNUMSEG=4, and so on. Here is the formula for DBIBITS.

```
DBIBITS = 1 + log-base-2 DBNUMSEG          (discard mantissa)
```

For example, suppose a cache file has 25,000 pages and ME equals 1. This cache file has 25,000 entries that are divided into 196 segments (195 128-entry segments and a short 40-entry segment on the end). Eight bits are needed to count to 196 (because 2 to the 7th power is 128 and 2 to the 8th power is 256). Therefore, each disk file page that is to reside in this cache area requires an eight-bit pointer to the correct part of XBUFTBL.

The table below shows DBIBITS and the amount of storage required for XBUFTBL, for DBENTS from one to 32,460. For example, if DBENTS equals 25000, DBIBITS equals 8. The middle column shows the approximate upper and lower bounds for XBUFTBL size. For example, DBENTS=16257 needs 127K for XBUFTBL, and DBENTS=32640 needs 255K.

DBENTS	XBUFTBL Size (K-bytes)	DBIBITS
1-128	1	1
129-384	1-3	2
385-896	3-7	3
897-1920	7-15	4
1921-3968	15-31	5
3969-8064	31-63	6
8065-16256	63-127	7
16257-32640	127-255	8

DBIENTS is the number of bytes required for a bitmap of 256 entries each DBIBITS long. In the current example, DBIENTS is 256 bytes. Note: the bit entries are contiguous, not byte-aligned. Here is the formula for DBIENTS:

```
DBIENTS = DBIBITS * 256 / 8 = DBIBITS * 32
```

The DSCT consists of a 36-byte prefix followed by N bitmaps, each DBIENTS long. XBUFIMX is the maximum number of disk files that can be open.

```
n = ((DBNUMSEG * XBUFIMX + 7) / 8) * 8

if n is less than 8, N = 8
if n is greater than 248, N = 248
if n is between 8 and 248, inclusively, N = n
```

This somewhat arbitrary calculation tries to arrive at a reasonable number of 256-entry bitmaps for each DSCT. At initialization time, the calculation is performed once for each XBUFTBL; subsequently, it is a fixed number for that cache file.

Each DSCT requires the following space in bytes:

```
SIZE = 36 + (N * DBIENTS)
```

Suppose you request that the 25,000 page cache file discussed here should be shared by 24 disk files.

```
n = ((196 * 24 + 7) / 8) * 8 = (4711/8) * 8 = 4704
```

Since n is greater than 248, N equals 248.

```
SIZE = 36 + (248 * 256) = 63524
```

All disk files sharing a cache file share the DSCT. In the current example, the first DSCT has 248 available bitmaps, acquired along with the XBUFTBL at initialization. The first 248 DPTs that experience I/O activity point to bitmaps on this DSCT.

The next DPT that becomes active forces the acquisition of another 63,524-byte DSCT via a conditional GETMAIN. If the GETMAIN fails, XBUF issues a WTO and sets a flag so that it does not try any more GETMAINs for this XBUFTBL. XBUF then makes the best use of the existing DSCTs.

XBUF checks the counter in the first byte of the DPT to see whether all pages for a given disk file were dropped from a DSCT bitmap. If all pages were dropped, the bitmap is freed for use by the next DPT that needs it.

Chapter 8

XBUF Messages

ERROR MESSAGES FROM FRONT-END MACROS

These messages appear because of errors in specifying the front-end XBUF macros. Some messages are issued by macros that are generated at layers below the simple macros. If you receive an error message that you do not understand, for example, an error in setting a swap parameter, contact SAS Institute Inc.

The messages are listed alphabetically by the message code prefix.

XB-01	FIRST POSITIONAL PARAMETER MUST BE EITHER 'START' OR 'END'
XB-02	XB START MUST BE FIRST MACRO IN XBUF TABLE
XB-03	EITHER USER-SPECIFIED OR SYSTEM-ASSIGNED FILES MUST BE SPECIFIED
XB-04	XB END MACRO OUT OF ORDER. MUST FOLLOW XBDUAL, XBSTAT, XBCACHE END, OR XBMEMORY END MACRO
XB-05	SWAP MUST BE 'YES' OR 'NO'. FOUND value
XB-06	TIME MUST BE 'YES' OR 'NO'
XB-07	LTDROPC MUST BE LESS THAN OR EQUAL TO HTDROPC
XB-08	CNTIN MUST BE NUMERIC
XB-09	LTDROPC MUST BE NUMERIC
XB-10	HTDROPC MUST BE NUMERIC
XB-11	RRAT MUST BE NUMERIC
XB-12	SECI MUST BE NUMERIC
XBCACHE-01	FIRST POSITIONAL PARAMETER MUST BE EITHER 'START' OR 'END'

XBCACHE-02	DDNAME MUST START WITH ALPHA CHARACTER
XBCACHE-03	DDNAME MUST HAVE NUMERIC 8TH CHARACTER
XBCACHE-04	XBCACHE END MACRO OUT OF ORDER. MUST FOLLOW XBDD MACRO
XBCACHE-05	XBCACHE MACRO OUT OF ORDER. MUST FOLLOW XBDUAL, XBSTAT, XBCACHE END, XBMEMORY END, OR XB START MACRO
XBDD-01	DDNAME MUST HAVE 8-CHARACTER VALUE
XBDD-02	DDNAME MUST START WITH ALPHA CHARACTER
XBDD-03	DDNAME MUST HAVE NUMERIC 8TH CHARACTER
XBDD-04	XBDD MACRO CANNOT FOLLOW XB END MACRO
XBDDVER-00	DUPLICATE DDNAME
XBDDVER-01	DDNAME CANNOT DUPLICATE AN XBNAME
XBDUAL-01	DDNAME MUST HAVE 8-CHARACTER VALUE
XBDUAL-02	DDNAME MUST START WITH ALPHA CHARACTER
XBDUAL-03	DDNAME MUST HAVE NUMERIC 8TH CHARACTER
XBDUAL-04	DUPLICATE XBNAME
XBDUAL-05	XBDUAL MACRO OUT OF ORDER. MUST FOLLOW XBDUAL, XBSTAT, XBCACHE END, XBMEMORY END, OR XB START MACRO
XBLKSIZE-01	BLKSIZE MAY BE INCORRECT. WILL BE CHECKED WHEN SYSTEM 2000 SOFTWARE IS INITIALIZED
XBMEMORY-01	FIRST POSITIONAL PARAMETER MUST BE EITHER 'START' OR 'END'
XBMEMORY-02	DDNAME MUST START WITH ALPHA CHARACTER
XBMEMORY-03	DDNAME MUST HAVE NUMERIC 8TH CHARACTER
XBMEMORY-04	AT LEAST ONE XBDD MUST BE ASSOCIATED WITH THIS XBMEMORY MACRO
XBMEMORY-05	XBMEMORY END MACRO OUT OF ORDER. MUST FOLLOW XBDD MACRO
XBMEMORY-06	XBMEMORY START MACRO OUT OF ORDER. MUST FOLLOW XBDUAL, XBSTAT, XBCACHE END, XBMEMORY END, OR XB START MACRO
XBSTAT-01	BLKSIZE MUST BE PRESENT
XBSTAT-02	COUNT OF BUFFERS MUST BE PRESENT

XBSTAT-03	COUNT OF SYSTEM-GENERATED DDNAMES MUST BE PRESENT
XBSTAT-04	XBSTAT MACRO OUT OF ORDER. MUST FOLLOW XBDUAL, XBSTAT, XBCACHE END, XBMEMORY END, OR XB START MACRO
XBXBVER-01	DUPLICATE XBNAME
XBXBVER-02	XBNAME CANNOT DUPLICATE A DDNAME

FATAL INITIALIZATION MESSAGES

S2K2001/<sid>- XBUF COULD NOT LOAD MEMBERS: XBUFTBL, S2KXBUF -
is issued if XBUF software cannot be loaded. If you specify XBUF=YES or XBUF=STAT in your execution parameters, SYSTEM 2000 software checks to see whether members named S2KXBUF and XBUFTBL are on the load library (JOBLIB or STEPLIB). This WTO message appears if either or both members cannot be loaded, followed by user abend 601.

Also, if you specify the XBUF execution parameter with a value other than NO, YES, or STAT, WTO message S2K0102 advises you of the invalid parameter value, and the SYSTEM 2000 initialization terminates.

INFORMATIVE INITIALIZATION MESSAGES

S2K0131/<sid>- nnnnnnnn NOW BEING FORMATTED -
is issued when the cache files are being formatted. XBUF formats cache files with the same SYSTEM 2000 routine that formats scratch pads. nnnnnnnn is the cache file DDname. Cache files are formatted after scratch pads.

Also, when a dual logged cache file is opened, an informative message shows how many disk file records were copied to the cache file.

RUN-TIME ERROR MESSAGES FROM XBUF

If initialization is successful, XBUF rejects invalid console commands with explanatory messages. The general form of the message is

```
S2Knnnn/<sid>-XBUF    text
```

where nnnn is the message number and text is the message explaining the response to your command.

XBUF does not interleave messages into SCF output or send return codes to PLEX programs. Instead, it sends write-to-operator (WTO) messages to the console operator, and these are usually routed by your system programmers to job control listings for hard copy. However, if you are using an alternate console, WTO messages are sent to your terminal and are not routed to hard copy.

2000 XBUF NOT ACTIVE -

is issued if you enter an XBUF console command and the XBUF execution parameter equals NO. The XBUF software is not available unless XBUF=YES or XBUF=STAT.

2020 CANNOT LOG I/O RATES. TOD CLOCK IS yyyyy

indicates an initialization error. The initial STCK instruction, which started the clock for subsequent rate computations, received a bad return code. Either a hardware failure occurred, or the operator changed the master system clock. The exact reason is given in yyyyy, that is, IN ERROR STATE, STOPPED, or NOT OPERATIONAL. XBUF runs, but rates contain zeros in the display.

2021 CANNOT SET DISK xxxxxxxx RT=z WHEN XBUF yyyyyyyy RT=N

indicates an error in a console command. You are trying to upgrade a disk file to a level of caching greater than that of the cache to which it belongs. xxxxxxxx is the DDname of the disk file; z is the requested (invalid) runtype, and yyyyyyyy is the DDname of the cache file. z must always be R or S. If any disk file runtypes really need to be R or S, make a new Cache Table with runtype R or S for the cache file.

2022 yyyyyyyy CANNOT BE SET TO RT=S WHEN OLD RT=x

indicates an error in a console command. Runtype STAT is invalid for dual logged cache files. Therefore, you cannot reset a dual logged cache file to RT=S, regardless of the old runtype. x is the current runtype; R or N are possible values. yyyyyyyy is the cache file DDname.

2023 GETMAIN FAILED FOR xxxxxxxx ... LEAVING RT='yy'

indicates an error in a console command. You tried to upgrade caching from NONE to REAL or STAT for disk file xxxxxxxx, but the system needed to do a GETMAIN for DPTs. Either no DPT was acquired at system initialization time, or the file has grown to more than 256 pages, so more space is now required. Unfortunately, the GETMAIN failed and runtype must stay as N (NONE) or NS (NONE-system-swapped).

2024 OPEN yyyyyyyy CANNOT BE SET TO RT=R WHEN OLD RT=N

indicates an error in a console command. Dual logged files are special; a complete copy of the disk file is in cache. XBUF does a page-for-page copy at the time the data base disk file is opened. Therefore, you can reset a dual logged file to REAL only when the disk file is closed. Resetting NONE to REAL if the file is open would not maintain the required exact copies of pages in cache. yyyyyyyy is the cache file DDname.

2025 XBUF IGNORES SECONDARY SPACE FOR CACHE FILE xxxxxxxx

is an informative WTO message. You allocated some secondary space that will never be used for DDname xxxxxxxx. Simplify your JCL by removing the secondary space value.

2026 XBUF FOUND BAD BLKSIZE nnnnn FOR DDname xxxxxxxx

indicates an initialization error. The cache file with DDname xxxxxxxx was found to have BLKSIZE=nnnnn at open time, and nnnnn was not a valid SYSTEM 2000 block size. A user abend follows this message, because the system cannot proceed.

2027 XBUF FOUND BLANK FIELD

indicates an error in a console command. You probably forgot to specify a required field.

2028 XBUF FOUND BLKSIZE nnnnn FOR xxxxxxxx, BUT yyyyyyyy BLKSIZE IS mmmmm XBUF SETTING xxxxxxxx RUNTYPE = 'NONE'

These messages indicate an initialization error. Except for runtype NONE, disk file block sizes must exactly match that of their cache file. These messages appear when the sizes do not match. nnnnn is the block size of disk file xxxxxxxx, and mmmmm is the block size of cache file yyyyyyyy.

2029 XBUF FOUND {DBMB / DBMP} =0 AND XBUF=STAT FOR DDNAME xxxxxxxx

indicates an initialization error. This message is followed by user abend 2. DBMB and DBMP are modeling parameters for cache files. You cannot let them default to zero if you specify XBUF=STAT in your run-time execution parameters. No real file exists from which the system can derive the model block size (DBMB) or the model number of pages in cache (DBMP). You must specify them in the CACHE macro, even if all disk files for a cache file have runtype NONE and the cache file has runtype NONE.

2030 XBUF FOUND ERROR IN PRTY FIELD ... MUST BE nn/nnn

indicates an error in a console command. You have specified too many or too few characters in the percentages, forgotten the slash, and so on.

2031 XBUF FOUND INVALID CHARACTER x IN DDNAME FIELD -

indicates an error in a console command. The character displayed in the message is not allowed in IBM DDnames.

2032 XBUF FOUND INVALID CHARACTER x IN PRIORITY FIELD -

indicates an error in a console command. Only numbers and the slash are allowed in the priority field.

2033 XBUF FOUND INVALID CHARACTER x IN RUNTYPE FIELD -

indicates an error in a console command. Only N, R, and S are meaningful for runtype.

2034 XBUF FOUND LOW PRTY GREATER THAN HIGH PRTY ... NOT ALLOWED -

indicates an error in a console command. You cannot set the low percentage higher than the high percentage. Raise the high percentage and try again.

2035 XBUF FOUND NEW HTDROPC LESS THAN OLD LTDROPC ... NOT ALLOWED -

indicates an error in a console command. You cannot decrease the high target drop criterion below the low target drop criterion. Decrease LTDROPC and then try to reset HTDROPC.

2036 XBUF FOUND NEW LTDROPC GREATER THAN OLD HTDROPC ... NOT ALLOWED -

indicates an error in a console command. You cannot increase the low target drop criterion above the current high target drop criterion. Increase HTDROPC and then try to reset LTDROPC.

2037 XBUF FOUND NONNUMERIC DATA IN TDROPC FIELD -

indicates an error in a console command. You tried to reset LTDROPC or HTDROPC, but you specified a nonnumeric new value.

2038 XBUF FOUND NO POOL FOR BLKSIZE nnnnn, DDNAME xxxxxxxx

indicates an initialization error. SYSTEM 2000 pools, as specified in the execution parameters, cannot handle the block size nnnnn for the cache file with DDname xxxxxxxx. Since the system cannot proceed, this message is followed by an abend.

2039 XBUF FOUND NUMBER GREATER THAN 32767 IN TDROPC FIELD -
indicates an error in a console command. You tried to reset LTDROPC or HTDROPC, but you specified a value that was too large.

2040 XBUF FOUND PRTY GREATER THAN 100 ... NOT ALLOWED -
indicates an error in console command. The high priority percentage for a disk file cannot be greater than 100.

2041 XBUF FOUND TDROPC. YOU CAN RESET ONLY LTDROPC OR HTDROPC -
indicates an error in a console command. You tried to reset TDROPC, which is not specific enough. You can display TDROPC and get both LTDROPC and HTDROPC values, but you must reset LTDROPC and HTDROPC separately.

2042 XBUF FOUND {TIME / SWAP} ALREADY 'yyy'
is an informative message. yyy is YES or NO. Your reset command was ignored because the TIME or SWAP parameter was already set the way you wanted it.

2043 XBUF FOUND TIME=NO. UNABLE TO REPLY -
indicates you requested some rate information, but the clock is disabled. You cannot display rate information unless the XBUF timer is enabled. To display the rates, reset TIME=YES, wait SECI seconds, and re-enter your display command.

2044 XBUF FOUND TOO MANY CHARACTERS IN DDNAME FIELD -
indicates an error in a console command. IBM allows only eight characters in a DDname, but you have requested some action for a longer DDname.

2045 XBUF FOUND UNRECOGNIZABLE COMMAND -
indicates an error in a console command. The only valid commands are D, DISPLAY, E, R, and RESET.

2046 XBUF FOUND UNRECOGNIZABLE THIRD FIELD IN COMMAND -
indicates an error in a console command. The first two fields in the command are valid, but XBUF does not recognize the third field. If you have specified DDnames or * in the second command field, you cannot display anything but RATES or PAGES in the third field. If you specified RT or PRTY in the second field, XBUF expects numbers in the third field.

2047 XBUF FOUND VERSION #nnnn IN XBUFTBL. THIS IS XBUF Rmmmm
indicates an initialization error. The Cache Table that was loaded had been assembled with an incompatible CACHE macro. The version number in the Cache Table (and the macro) was nnnn, but the version number in the executing system, load module S2KXBUF, was mmmm. This message is followed by user abend 2. You must reassemble your Cache Table with the correct version of the CACHE macro.

2048 XBUF FOUND ZERO PRIMARY SPACE FOR xxxxxxxx
indicates an initialization error. You must allocate primary space for a cache file. The cache file with DDname xxxxxxxx has no primary space allocated to it. User abend 104 follows the WTO.

2049 XBUF FOUND nnnnn BLKS ON xxxxxxxx
2050 XBUF FOUND mmmmm BLKS ON yyyyyyyy
2051 XBUF SETTING RUNTYPE TO N.

When a REAL dual logged cache file is opened, these messages appear if the disk file will not fit completely in the primary space allocated to the cache file. Runtype is set to NONE for the duration. nnnnn is the number of records in the disk file with DDname xxxxxxxx; mmmmm is the number of records in the cache area with DDname yyyyyyyy.

2052 XBUF GETMAIN FAILED FOR xxxxxxxx DSCT

XBUF failed to acquire a DSCT for the cache file with the DDname displayed in xxxxxxxx. When irregular work loads are being processed, XBUF sometimes needs more memory for DSCTs. If XBUF cannot acquire the memory, that particular cache file reaches its limit of DSCTs. XBUF does not attempt any more GETMAINs that would fail. Processing might be cramped, but no runtypes are reset. The MEMFAIL value displayed by the PAGES console command shows how often this situation has occurred.

2053 XBUF GETMAIN FAILED FOR xxxxxxxx ... SETTING RUNTYPE TO 'NONE'

During open processing for the disk file with DDname xxxxxxxx, XBUF needed to acquire memory for DPTs. Either it was the first open or the disk file was closed and reopened, having grown more than 256 pages in the interim. At any rate, the GETMAIN failed. Therefore, the disk file will be processed from now on as though it has runtype NONE.

2054 XBUF I/O ERROR WRITING ON xxxxxxxx
2055 XBUF SETTING RUNTYPE TO 'NONE' FOR xxxxxxxx

XBUF detected a hardware problem after attempting to write on a cache file. Since an exact copy of every page that was written to cache exists on disk, the disk file is good and the data base is not flagged as damaged. Instead, the runtypes of the failing cache file and its disk files are set to NONE. Processing continues.

2057 XBUF=STAT WAS SPECIFIED IN PARMS. YOU CANNOT RESET TO REAL -

indicates an error in a console command. You are trying to upgrade the level of caching above the level requested in the SYSTEM 2000 execution parameters. You must bring down the system and change your XBUF execution parameter to XBUF=YES if you want to do real caching at run time. XBUF cannot format cache files for real I/O or even check for the presence of their DDnames unless XBUF=YES.

2058 XBUF FOUND ILLEGAL {TIME / SWAP} **REQUEST nnn. TIME= YES/NO REQUIRED**

indicates an error in a request to reset the TIME or SWAP value. The value must be YES or NO.

2059 XBUF SET {TIME / SWAP} **TO** {YES / NO}

is an informative message.

2060 XBUF {TIME / SWAP} = {YES / NO}

is an informative message.

2061 XBUF {LTDROPC / HTDROPC} **= nnn (, HTDROPC = NNN)**
is an informative message.

2062 XBUF STAGED nnnn RECORDS FROM xxxxxxxx TO yyyyyyyy
is an informative message, where xxxxxxxx is a data base file name and yyyyyyyy is a cache file name.

2063 XBUF type RT = runtype xxxxxxxx BLKSIZE = nnnnn COUNT =00000
is an informative message issued as files are being formatted. Type is one of the following: XA MEMORY, MEMORY, DISK, or DUAL LOG. Runtype is either REAL, STAT, or NONE. xxxxxxxx is a cache file name. nnnnnnn is the block size.

2064 XBUF WILL COPY xxxxxxxx ONTO yyyyyyyy WHEN OPENED
is an informative message issued as files are being formatted. xxxxxxxx is the data base file name, and yyyyyyyy is the dual log file name.

2065 XBUF RESET {LTDROPC / HTDROPC} **TO nnn**
is an informative message.

2066 XBUF RESET xxxxxxxx RT FROM n TO n
is an informative message, issued when XBUF resets the runtype for a file (named xxxxxxxx).

2067 XBUF FOUND NO PADS
is an informative message.

2068 XBUF RESET nnn OF nnnn DISKS
is an informative message.

Index

X

3