English

FUJITSU

FUJITSU Software

BS2000 OSD/BC V11.0
DMS Introduction

User Guide

Edition June 2020

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to: bs2000services@ts.fujitsu.com.

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Table of Contents

DMS Introduction

1 Preface

This manual describes the functions of the Data Management System (DMS) of BS2000. DMS is an autonomous subsystem within BS2000. It forms the connecting link between the user-controlled accesses to data objects and the central device drivers of the base system. DMS in turn makes use of certain services offered by the base system.

DMS provides support primarily for data processing on public volume sets (often abbreviated to “pubsets”).

Individual disk volumes are grouped together in a pubset. In addition to the home pubset, which accommodates all the files needed for the session, further pubsets can be added (imported) to the system. The system administrator can assign user IDs and thus the data of the users to the various pubsets in such a manner that a favorable distribution is achieved for system operation. Performance and availability are decisive criteria in this context.

For the “normal” user of BS2000, this means: unless he explicitly requests private volumes, all of his files are created on the pubset which systems support has defined as his default pubset. He does not need to request volumes or devices. DMS also manages his memory requirements.

An entry in the SYSSRPM user catalog is kept in each pubset for each user who is authorized to access this pubset. The files for the users are maintained in the TSOSCAT file catalog of the pubset. All user files are uniquely identified by the user ID and the catalog ID (the ID of the pubset on which the catalog is kept). At the same time, this ensures that file access is permitted only to the owner of the files unless he explicitly permits other users to access his files.

In addition to storing files on pubsets, DMS supports the user in managing data on private volumes (e.g. tape media) and on Net-Storage. Net-Storage is storage space which is connected to BS2000 via NFS. In every case the files to be processed must be cataloged in a file pubset's file catalog.

Functions of the Data Management System

The Data Management System (DMS) enables users to process their data by maintaining files and by using the various functions necessary for file processing. The DMS functions can be roughly divided into the following groups:

- creating and managing files, including memory space management
- managing catalogs
- making files available and processing files using the access methods
- assigning files to programs.

In addition to this, DMS permits the user to define data and file protection features at the file level. Data security is also supported by DMS, for example by setting locks during file access.

The functions of DMS are implemented via the Assembler interfaces described in the “DMS Macros” manual [1 (Related publications)] and the command interfaces described in the “Commands” manuals [3 (Related publications)].

Versions of related software products

References in the manual to the following software products always refer to the specified versions:

- DAB V9.5
- HSMS V12.0
- HIPLEX-MSCF V11.0

- SECOS V5.5

1.1 Objectives and target groups of this manual

This manual is intended for users who wish to manage and process their files with the aid of the Assembler interface or DMS commands, or who wish to control data and file protection, data security, etc. by means of macros or commands and who require a general introduction to the functions of DMS.

This manual contains a general description of the DMS functions and serves as an introduction to DMS. The DMS interfaces are described in detail in the manuals “DMS Macros” [1 (Related publications)] and “Commands” [3 (Related publications)]. These also contain information on the commands and macros mentioned in the present manual which are not accompanied by a reference to another manual. Abbreviated titles are as a rule used for references to other manuals in the text. The Related publications section contains the full titles.

1.2 Summary of contents

The main topics discussed in this manual are as follows:

Overview of the interfaces of the DMS functions

This section contains tables explaining the basic functions of DMS and the macros and commands available for each function.

Volumes and files in BS2000, file management and file protection

These sections give an overview of the volumes available in BS2000. The various file types and structures, file and catalog management, file access and file and data protection are defined and explained.

Files on disk and tape

These sections describe the different volumes, volume and device management, tape file labels and label processing, memory management, and MF/MV sets.

Opening and closing files

These sections deal with OPEN and CLOSE processing within DMS.

Access methods

These sections provide a general description of record, block and disk formats, and deal with the access methods UPAM, FASTPAM, DIV, SAM, BTAM, EAM and ISAM.

Appendix

The appendix contains a number of tables and lists referred to at various places in this manual.
The device and volume type tables are contained in the "System Installation" manual [15 (Related publications)].
Information on the volume types for tape processing and the meaning of the output values of SHOW commands is provided in the "Commands" manual [3 (Related publications)].

Readme file

The functional changes to the current product version and revisions to this manual are described in the product-specific Readme file.

Readme files are available to you online in addition to the product manuals under the various products at *http://bs2manuals.ts.fujitsu.com*. You will also find the Readme files on the Softbook DVD.

Information under BS2000

When a Readme file exists for a product version, you will find the following file on the BS2000 system:

```
SYSRME.<product>.<version>.<lang>
```

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `/SHOW-FILE` command or an editor.
The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product’s files are stored.

Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available online at *http://bs2manuals.ts.fujitsu.com*.

1.3 Changes since the last edition of the manual

The “Introductory Guide to DMS” was last published for BS2000 OSD/BC V10.0. The following major changes have been made since the last edition of the manual:

- The “Net-Storage” section in the “Volumes” chapter now includes the processing of SAM node files, which are supported from BS2000 OSD/BC V11.0 onwards.
- The NET-CODED-CHARACTER-SET (NETCCS) file attribute describes how the actual coding is to be converted on the Net-Storage, when node files are processed or displayed in BS2000. The file attribute is supported for node files, but the entry is only evaluated for SAM node files. For PAM node files created in BS2000 OSD/BC V10.0, the value is implicitly assumed to be *NO-CONV.
  The NET-CODED-CHARACTER-SET can be explicitly specified with the CREATE-FILE or MODIFY-FILE-ATTRIBUTES commands. By default, the NETCCSN of a file to be created is determined from the user specific setting. This setting, in turn, is based on the NETCODE system parameter when the user is being created.
- Node files can now also be imported as SAM files, using IMPORT-NODE-FILE.
- The SHOW-FILE-ATTRIBUTES command outputs the value of the last-byte pointer as LAST-BYTE in the information section *Allocation* or as S variables if the respective valid bit has been set in the catalog entry of the file. In addition, files can be selected based on the properties of the last-byte pointer.
- When creating ISAM files, the new system parameter ISBLKCTL controls the BLK-CONTR file attribute, if the user has made no other specification. The default setting, ISBLKCTL=c'NONKEY', causes NK-ISAM files (BLK-CONTR=DATA file attribute) to be created by default on K disks as well This default setting is a consequence of the fact that K-ISAM is becoming less and less relevant in practice.
- From BS2000 OSD/BC V11.0B onwards:
  To support version backup (from HSMS V12.0 onwards), the file catalog has been expanded with the file attribute NUM-OF-BACKUP-VERS. The value defined for this file attribute (0 to 32) determines whether the file is part of the version backup and, if yes, also the maximum number of file versions to be saved in the version backup archive. If the value is 0, the file is not part of the version backup. Temporary files, files on private disk, files on tape and file generations are not supported by the version backup, i.e. for these files the value can only be 0. The setting is specified in the CATAL macro (NUM_OF_BACKUP_VERS operand) or in the CREATE-FILE and MODIFY-FILE-ATTRIBUTES (NUM-OF-BACKUP-VERS operand) commands. If no value is specified, the value from the NUMBACK system parameter is taken over for all files that support version backup.

Overview of the changes to the DMS macro interface:

Macro / Operand or RC	Nature of the change
CATAL	New operand: NETCCS determines the NET-CODED-CHAR-SET.
	New operand from V11.0B onwards: NUM_OF_BACKUP_VERS specifies whether the file is part of the version backup and, if yes, also the maximum number of file versions to be saved in the version backup archive.
FSTAT	New operand: LBPOINT for selecting files based on the value of the last-byte pointer. The output area also contains the NETCCS file attribute.

	New operand from V11.0B onwards: VERSION-BACKUP for selecting files based on whether they can be part of an version backup. A file can only be part of an version backup if the NUM-OF-BACKUP-VERS file attribute has a value > 0. The #BACK-VERS output field includes the value for the NUM-OF-BACKUP-VERS file attribute.
IMPNFIL	New operand value: FILESTR=*SAM imports node files as SAM files.

A full description of the macros is provided in the “DMS Macros” manual [1 (Related publications)].

Overview of the changes to the DMS command interface

Command / Operand	Nature of the change
CREATE-FILE	New operand: NET-CODED-CHAR-SET determines the NET-CODED-CHAR-SET for node files.
	New operand from V11.0B onwards: NUM-OF-BACKUP-VERS specifies whether the file is part of the version backup and, if yes, also the maximum number of file versions to be saved in the version backup archive.
IMPORT-NODE-FILE	New operand value: FILE-STRUCTURE=*SAM imports node files as SAM files.
MODIFY-FILE-ATTRIBUTES	New operand: NET-CODED-CHAR-SET determines the NET-CODED-CHAR-SET for node files.
	New operand from V11.0B onwards: NUM-OF-BACKUP-VERS specifies whether the file is part of the version backup and, if yes, also the maximum number of file versions to be saved in the version backup archive.
SHOW-FILE-ATTRIBUTES	New operand: LAST-BYTE-POINTER selects files based on the value of the last-byte pointer. Output to SYSOUT and in S variables expanded for the lastbytepointer and NETCCS.
	New operand from V11.0B onwards: VERSION-BACKUP selects files based on whether they can be part of an version backup. A file can only be part of an version backup if the file attribute NUM-OF-BACKUP-VERS has a value > 0. When output to SYSOUT, the NUM-OF-BACKUP-VERS file attribute is displayed in the `#BACK-VERS` output field, when output to S variables, it is displayed in `var(*list).NUM-OF-BACKUP-VERS`.

A full description of the commands is provided in the “Commands“ manual [3 (Related publications)].

General change

GS (global storage) as a non-volatile cache medium is no longer supported.

From BS2000 OSD/BC V11.0 onwards, the connection of peripheral devices is only supported via type FC channel. Together with the type S channel, the function “shareable private disk” (SPD) has been omitted.

From BS2000 OSD/BC V11.0 onwards, encrypting and decrypting of files is done via the CRYPT subsystem, see the “CRYPT” manual [24 (Related publications)].

1.4 Notational conventions

The metasyntax used in this manual to describe statements is explained in the "Commands" manual [3 (Related publications)]. Operands reserved for privileged users are shown in the formats with a gray background.

The following typographical elements are used in this manual:

i	For notes on particularly important information
!	This symbol designates special information that points out the possibility that data can be lost or that other serious damage may occur.
`[ ]`	Related publications are referred to in short form throughout the manual. The full title of each publication referenced in the manual appears in the list of related publications on "Related publications".
`Input`	Inputs and system outputs in examples are shown in `typewriter font`.

2 Overview of the interfaces of the DMS functions

The following tables provide an overview of the interfaces of the DMS functions. Frequent references to these interfaces are made in the various chapters of this manual. If both interfaces are available, separate tables are provided for the command interfaces and program interfaces (macros). A description of the interfaces and all the operands can be found in the manuals “Commands” [3 (Related publications)], “DMS Macros” [3 (Related publications)] and “Executive Macros” [3 (Related publications)].

In addition to the DMS commands, a number of NDM (Nucleus Device Management) commands have been included; these support the user in device and volume management, provide information on user entries, etc.

Table of DMS macros

Macro	Brief description
ADDPLNK	*ISAM:* Defines a pool link name.
BTAM	Controls all BTAM actions.
CATAL	Creates or updates a catalog entry.
CHKFAR	Checks file access rights.
CHNGE	Changes a TFT entry.
CLOSE	Close file
COMPFIL	Compares two disk files.
COPFILE	Copies a file.
CREAIX	*ISAM*: Creates a secondary key (alternate index) for an ISAM file.
CREPOOL	*ISAM*: Creates an ISAM pool.
DECFILE	Converts an encrypted file into an unencrypted file.
DELAIX	*ISAM*: Deletes secondary keys of an ISAM file.
DELPOOL	*ISAM*: Deletes/releases an ISAM pool.
DIV	Permits file access via the virtual address space.
DROPTFT	Unlocks locked TFT entries.
EAM	Macro (type R)
ELIM	*ISAM:* Deletes a record from the file.
ENCFILE	Converts an unencrypted file into an encrypted file.
ERASE	Deletes one or more files.
EXLST	Compiles a list of exit addresses (type O).

EXRTN	Implements a return from error routines (type R).
FCB	Defines a file control block (type O).
FCBAD	Creates FCB addresses (type O).
FEOV	*BTAM/SAM*: Initiates a tape swap.
FILE	Defines file attributes/controls file processing.
FILELST	Creates variable operand areas for the FILE macro.
FPAMACC	*FASTPAM:* Defines file accesses.
FPAMSRV	*FASTPAM:* Defines management calls.
FSTAT	Requests catalog information.
GET	*ISAM/SAM:* Reads the next record.
GETFL	*ISAM:* Reads the next record following the flag.
GETKY	*ISAM:* Reads the record with the specified key.
GETR	*ISAM:* Sequential reverse read.
IDBPL	*BTAM:* BTAM operand list (type O)
IDFCB	Supplies FCB with symbolic names (type O).
IDFCBE	Supplies FCBE with symbolic names (type O).
IDMCB	Supplies MFCB (EAM control block with symbolic names).
IDPPL	*UPAM:* PAM operand list
IMPNFIL	Creates (imports) catalog entries for node files
IMPORT	Creates (imports) catalog entries for files
INSRT	*ISAM:* Inserts a record (type R)
ISREQ	*ISAM:* Cancels a lock (type O)
LBRET	Implements a return from a user label handling routine (type R).
LFFSNAP	Lists files from a Snapset
LJFSNAP	Lists job variables from a Snapset
MAILFIL	Sends a file or library member to a user ID by email.
NDWERINF	*BTAM:* Interrogates the status bytes.
OPEN	Opens a file (type R)

OSTAT	*ISAM:* Requests information about currently open files (type R).
PAM	*UPAM:* Performs UPAM actions.
PUT	*ISAM/SAM:* Writes a record.
PUTX	*ISAM/SAM:* Replaces a record.
RDTFT	Displays TFT and TST information.
RELTFT	Deletes a TFT entry.
REMPLNK	*ISAM:* Deletes a pool link name.
RETRY	*ISAM:* Repeats the last macro.
RELSE	Closes block
RFFSNAP	Restores files from a Snapset
RJFSNAP	Restores job variables from a Snapset
SETL	*ISAM/SAM:* Positions the pointer in the file.
SHOPLNK	Displays information about ISAM pool link names.
SHOPOOL	Displays information about ISAM pools.
SHOWAIX	Displays information about secondary keys.
STORE	*ISAM:* Stores a record.
VERIF	Restores a file.

Table 1: DMS macros

DMS macros and DMS commands with identical functionality

Macro	Command	Function
ADDPLNK	ADD-ISAM-POOL-LINK	Defines pool link names for an ISAM pool.
CATAL	CREATE-FILE	Creates a catalog entry.
	CREATE-FILE-GROUP	Defines file generation groups.
	MODIFY-FILE-ATTRIBUTES	Defines protection mechanisms.
	MODIFY-FILE-GROUP-ATTRIBUTES	Defines protection mechanisms.
CHNGE	CHANGE-FILE-LINK	Updates a file link name in the TFT.
COMPFIL	COMPARE-DISK-FILES	Compares two disk files.

COPFILE	COPY-FILE	Copies a file.
CREAIX	CREATE-ALTERNATE-INDEX	Creates a secondary index for an ISAM file.
CREPOOL	CREATE-ISAM-POOL	Creates an ISAM pool.
DECFILE	DECRYPT-FILE	Converts an encrypted file into an unencrypted file.
DELAIX	DELETE-ALTERNATE-INDEX	Deletes secondary keys of an ISAM file.
DELPOOL	DELETE-ISAM-POOL	Deletes/releases an ISAM pool.
DROPTFT	UNLOCK-FILE-LINK	Unlocks locked TFT entries.
ENCFILE	ENCRYPT-FILE	Converts an unencrypted file into an encrypted file.
ERASE	DELETE-FILE	Deletes one or more files.
	DELETE-FILE-GENERATION	Deletes file generations.
	DELETE-FILE-GROUP	Deletes one or more file generation groups.
	DELETE-SYSTEM-FILE	Deletes one or more system files.
	EXPORT-FILE	Exports one or more files.
	EXPORT-NODE-FILE	Exports node file(s).
FILE	CREATE-FILE	Creates a file.
	CREATE-FILE-GENERATION	Creates a file generation.
	CREATE-TAPE-SET	Creates a TST entry.
	EXTEND-TAPE-SET	Extends a TST entry.
	IMPORT-FILE	Imports a file.
	MODIFY-FILE-ATTRIBUTES	Modifies attributes of a file.
	MODIFY-FILE-GENERATION-SUPPORT	Modifies attributes of a file generation.
	ADD-FILE-LINK	Creates a TFT entry.
FSTAT	IMPORT-FILE	Supplies information from the file catalog.
	SHOW-FILE-ATTRIBUTES	Displays file attributes.

IMPNFIL	IMPORT-NODE-FILE	Creates (imports) catalog entries for node files.
IMPORT	CHECK-IMPORT-DISK-FILE	Checks file import. up front.
	IMPORT-FILE	Creates (imports) catalog entries for files.
LFFSNAP	LIST-FILE-FROM-SNAPSET	Lists files from a Snapset.
LJFSNAP	LIST-JV-FROM-SNAPSET	Lists job variables from a Snapset.
MAILFIL	MAIL-FILE	Sends a file or library member to a user ID by email.
RDTFT	SHOW-FILE-LINK	Supplies information from the TFT.
RELTFT	REMOVE-FILE-LINK	Removes a TFT entry.
	DELETE-TAPE-SET	Deletes a TST entry.
REMPLNK	REMOVE-ISAM-POOL-LINK	Deletes pool link names.
RFFSNAP	RESTORE-FILE-FROM-SNAP-SET	Restores files from a Snapset.
RJFSNAP	RESTORE-JV-FROM-SNAP-SET	Restores job variables from a Snapset.
SHOWAIX	SHOW-INDEX-ATTRIBUTES	Displays information about secondary keys of an ISAM file.
SHOPLNK	SHOW-ISAM-POOL-LINK	Displays assignments of pool link names to ISAM pools.
SHOPOOL	SHOW-ISAM-POOL-ATTRIBUTES	Displays information about an ISAM pool.
VERIF	CHECK-FILE-CONSISTENCY	Restores file consistency.
	REMOVE-FILE-ALLOCATION-LOCKS	Removes locks from files, file generations and FFGs set as a result of a system crash or an aborted job.
	REPAIR-DISK-FILES	Restores files, file generations and FFGs locked as a result of a system crash or an aborted job.

Table 2: Comparison of DMS macros / DMS commands

DMS commands without equivalent macro

Command	Function
ADD-CRYPTO-PASSWORD	Stores the crypto password for decrypting encrypted file contents in the task’s password table.

ADD-PASSWORD	Enters a password in the password table of a job.
CONCATENATE-DISK-FILES	Concatenates SAM files.
EDIT-FILE-ATTRIBUTES EDIT-FILE-GROUP-ATTRIBUTES EDIT-FILE-GENERATION-SUPPORT	Activates the guided dialog of the corresponding MODIFY command and enables an existing catalog entry to be "edited".
EDIT-FILE-LINK	Activates the guided dialog of the ADD-FILE-LINK command and enables an existing TFT entry to be "edited".
LIST-NODE-FILES	Lists node files of a Net-Storage volume.
LOCK-FILE-LINK	Locks a TFT entry; the lock can subsequently be cancelled by an UNLOCK-FILE-LINK command.
REMOVE-CRYPTO-PASSWORD	Removes the crypto password from the password table of the ongoing task.
REMOVE-PASSWORD	Removes a password from the password table of a job.
REPAIR-FILE-LOCKS	Eliminates unauthorized file locks
RESTART-PROGRAM	Resumes a program at a point previously saved by means of a WRCPT (or CHKPT) macro.
SHOW-BLOCK-TO-FILE-ASSIGNMENT	Privileged command: Displays files in which the requested blocks are located.
SHOW-FILE	Outputs a file to SYSOUT
SHOW-FILE-LOCKS	Displays locks set for a file.
START-FILE-CACHING	Starts caching of open files
STOP-FILE-CACHING	Terminates caching of open files

Table 3: DMS commands without equivalent macro

2.1 File maintenance

File maintenance includes not only creating, copying and deleting files, but also maintenance of the file catalog by the user. Information on the topic of “restoration” is provided in section “Data security”.

Macros

Macro	Operands	Brief description
CATAL		Creates or updates catalog entries.
COMPFIL		Compares two disk files.
COPFILE		Copies files.
ERASE		Erases/exports files.
FILE		Creates a catalog entry and reserves storage space for noncataloged files.
	SPACE	Reserves or releases storage space.
	NFTYPE	Creates a catalog entry for the file types BS2000 file or node file on Net-Storage.
	STATE	Creates catalog entries for files on private disk.
FSTAT		Displays information from the file catalog.
IMPNFIL		Creates (imports) catalog entries for node files
IMPORT		Creates (imports) catalog entries for files
MAILFIL		Sends a file or library member to a user ID by email.
VERIF		Restores corrupted files.

Commands

Command	Brief description

CREATE-FILE
CREATE-FILE-GROUP
CREATE-FILE-GENERATION
	Creates a catalog entry / reserves memory space.

MODIFY-FILE-ATTRIBUTES MODIFY-FILE-GROUP-ATTRIBUTES MODIFY-FILE-GENERATION-SUPPORT	Modifies a catalog entry / reserves/releases memory space.

EDIT-FILE-ATTRIBUTES EDIT-FILE-GROUP-ATTRIBUTES EDIT-FILE-GENERATION-SUPPORT	Activates the guided dialog of the corresponding MODIFY command and enables an existing catalog entry to be "edited".
IMPORT-FILE	Creates (imports) catalog entries for files
IMPORT-NODE-FILES	Creates (imports) catalog entries for node files
COMPARE-DISK-FILES	Compares two disk files.
COPY-FILE	Copies files.
CONCATENATE-DISK-FILES	Concatenates SAM files
DELETE-FILE DELETE-FILE-GROUP DELETE-FILE-GENERATION DELETE-SYSTEM-FILE	Deletes file Deletes FGG Deletes file generation Deletes system file
EXPORT-FILE	Exports catalog entries of files on private volumes and Net-Storage volumes.
EXPORT-NODE-FILES	Exports catalog entries for node files (on Net-Storage volumes).
LIST-NODE-FILES	Provides information on node files on Net-Storage volumes.
MAIL-FILE	Sends a file or library member to a user ID by email.
SHOW-FILE-ATTRIBUTES	Displays information from the file catalog.
CHECK-FILE-CONSISTENCY REPAIR-DISK-FILE	Restores corrupted files.
REMOVE-FILE-ALLOCATION-LOCKS	Removes a file lock.
REPAIR-FILE-LOCKS	Eliminates unauthorized file locks.

2.2 Controlling ACS functions

Users can optionally address files they require by using alternative file names or aliases. Internal assignments between aliases and actual file names are maintained in an alias catalog. Each user may also define a specific file name prefix for his or her file names. The following commands are available for controlling the ACS functions (no program interface is provided):

Command	Brief description
ADD-ACS-SYSTEM-FILE	Defines an ACS system file.
ADD-ALIAS-CATALOG-ENTRY	Adds an entry to the alias catalog of the current task and defines where the substitution applies.
HOLD-ALIAS-SUBSTITUTION	Interrupts the ACS function for the current task. No alias substitution occurs following this command.
LOAD-ALIAS-CATALOG	Transfers entries saved in an AC file to the alias catalog.
MODIFY-ACS-OPTIONS	Modifies task-local options.
MODIFY-ALIAS-CATALOG-ENTRY	Modifies an existing entry in the alias catalog.
PURGE-ALIAS-CATALOG	Deletes the alias catalog of the current task, but retains the options that where set for it.
REMOVE-ALIAS-CATALOG-ENTRY	Deletes an existing entry from the alias catalog.
RESUME-ALIAS-SUBSTITUTION	Resumes the alias substitution function
SET-FILE-NAME-PREFIX	Defines a prefix for file/JV names and specifies where the prefix applies.
SHOW-ACS-OPTIONS	Displays options (for local task).
SHOW-ACS-SYSTEM-FILES	Shows the predefined alias catalog system files.
SHOW-ALIAS-CATALOG-ENTRY	Displays selected alias catalog entries (on SYSOUT).
SHOW-FILE-NAME-PREFIX	Shows the current prefix and its scope.
STORE-ALIAS-CATALOG	Stores the alias catalog in a file.

2.3 Controlling file processing

To enable a file to be processed by a program, it must be possible to set up a connection between the two. This connection can either be defined in the FCB or, if the program uses an internal file name (the file link name), be established using the FILE macro. The connection is stored together with other information in the task file table (TFT). File processing is thus controlled via the TFT.

For NK-ISAM files, file processing control also incorporates the management of user ISAM pools, in which these files are processed. The user can also use standard ISAM pools of the system, but then has no influence on pool size or reservation.

Macros

Macro	Operands	Brief description
ADDPLNK		Assigns a pool link name to a user ISAM pool.
CHNGE		Assigns a new file link name to a file.
CREAIX		Creates a secondary key (alternate index) for an ISAM file.
CREPOOL		Creates a user ISAM pool.
DELAIX		Deletes secondary indices of an ISAM file.
DELPOOL		Deletes a user ISAM pool.
DROPTFT		Unlocks a locked TFT entry.
FCB	FILE	Defines a fixed link between file and program.
	LINK	Defines a file link name in the program.
	POOLLNK	Sets up a connection to a user ISAM pool.
FILE	LINK	Creates a TFT entry; further operands describe file and processing attributes.
	BLKCTRL	Defines the file format.
	NFTYPE	Defines the file type of a file on Net-Storage (BS2000 file or node file)
	POOLLNK	Sets up a connection to a user ISAM pool.
RDTFT		Displays TFT information.
RELTFT		Deletes a TFT entry.
REMPLNK		Deletes the pool link name.
SHOPLNK		Shows assignments of pool link names to ISAM pools.
SHOPOOL		Returns information on the attributes and occupancy of ISAM pools.

Commands

Command	Operands	Brief description
ADD-FILE-LINK		Creates a TFT entry; further operands describe file and processing attributes.
	BLOCK-CONTROL-INFO	Defines the file format.
	POOL-LINK	Sets up the connection to the ISAM pool.
EDIT-FILE-LINK		Activates the guided dialog of the ADD-FILE-LINK command and enables an existing TFT entry to be "edited".
SHOW-FILE-LINK		Displays TFT information.
CHANGE-FILE-LINK		Assigns a new file link name to a file.
LOCK-FILE-LINK		Locks a TFT entry, thus inhibiting any REMOVE-FILE-LINK command.
UNLOCK-FILE-LINK		Cancels the lock (LOCK-FILE-LINK) for the TFT entry.
REMOVE-FILE-LINK		Deletes a TFT entry.
CREATE-ISAM-POOL		Creates a user ISAM pool.
SHOW-ISAM-POOL-ATTRIBUTES		Displays information on the attributes and usage of ISAM pools.
DELETE-ISAM-POOL		Deletes the user ISAM pool.
ADD-ISAM-POOL-LINK		Assigns a pool link name to a user ISAM pool.
REMOVE-ISAM-POOL-LINK		Deletes the pool link name.
SHOW-ISAM-POOL-LINK		Shows assignments of pool link names to ISAM pools.
START-CACHING-FILES		Starts caching of open files
STOP-CACHING-FILES		Terminates caching of open files

2.4 Data protection and security support

The mechanisms for file and data protection automatically supported by DMS (access authorization checks, etc.) can be extended by the user, e.g. with the aid of passwords. Data security is assured by various mechanisms for recovering files or programs, etc.

2.4.1 File protection

Macros

Macro	Operands	Brief description
CATAL	SHARE	Controls shareability.
	ACCESS	Controls the type of access.
	OWNERAR GROUPAR OTHERAR	Defines access rights in the BASIC-ACL
	GUARDS	When SECOS is used: provides enhanced access protection for files.
	EXPASS RDPASS WRPASS	Define passwords for the various access levels.
	RETPD	Specifies a retention period.
	PROTECT	Transfers the protection attributes
CHKFAR		Checks the caller' s file access rights.
COPFILE	PROTECT	Transfers the protection attributes when a file is copied.
DECFILE		Converts an encrypted file into an unencrypted file.
ENCFILE		Converts an unencrypted file into an encrypted file.
FILE	RETPD	Defines a retention period (valid only if specified when the file is opened).
FCB	PASS	Permits access to password-protected files.
	RETPD	Specifies a retention period.

Commands

Command	Operands	Brief description

CREATE-FILE
CREATE-FILE-GROUP
MODIFY-FILE-ATTRIBUTES
MODIFY-FILE-GROUP-ATTRIBUTES
	USER-ACCESS	Controls shareability.

	ACCESS	Controls the type of access.
	EXEC-PASSWORD READ-PASSWORD WRITE-PASSWORD	Defines passwords for the various access levels.
MODIFY-FILE-ATTRIBUTES MODIFY-FILE-GROUP-ATTRIBUTES	RETENTION-PERIOD	Defines a retention period.

CREATE-FILE MODIFY-FILE-ATTRIBUTES	BASIC-ACL/GUARDS	Sets the access rights for the file.
COPY-FILE	PROTECTION=SAME	Transfers protection attributes when copying a file.
COPY-FILE	REPLACE-OLD-FILES	Determines whether a file can be overwritten using the COPY-FILE command.
ADD-FILE-LINK	RETENTION-PERIOD	Defines a retention period (valid only if specified when the file is opened).
ADD-PASSWORD		Specifies a password governing access to password-protected files.
REMOVE-PASSWORD		Deletes a password from the password list.

2.4.2 Data protection

Macros

Makro	Operanden	Kurzbeschreibung
CATAL	DESTROY	Specifies in the catalog entry that disk files or superfluous foreign data on tape files are to be overwritten during deletion (cf. FILE: DESTOC).
DECFILE		Converts an encrypted file into an unencrypted file.
ENCFILE		Converts an unencrypted file into an encrypted file.
ERASE	DESTROY	Specifies data destruction in conjunction with deletion.
FILE	DESTOC	Specifies that any data remaining on the tape in the case of a tape swap or after closing a tape file is overwritten.

Commands

Command	Operands	Brief description
ADD-CRYPTO-PASSWORD		Stores the crypto password for decrypting encrypted file contents in the task's password table.
ADD-FILE-LINK	DESTROY-OLD-CONTENTS	Specifies that any data remaining on a tape when it is swapped, or after a tape file is closed, is overwritten.

CREATE-FILE
CREATE-FILE-GROUP
MODIFY-FILE-ATTRIBUTES
MODIFY-FILE-GROUP-ATTRIBUTES
	DESTROY-BY-DELETE	Specifies in the catalog entry that disk files or superfluous foreign data on tape files are to be overwritten during deletion (see ADD-FILE-LINK: DESTROY-OLD-CONTENTS).

DECRYPT-FILE		Converts an encrypted file into an unencrypted file.
DELETE-FILE	DESTROY-ALL	Specifies data destruction in conjunction with deletion.
DELETE-FILE-GROUP DELETE-FILE-GEN	DESTROY-ALL	Specifies data destruction in conjunction with deletion.
ENCRYPT-FILE		Converts an unencrypted file into an encrypted file and defines the associated crypto password.

REMOVE-CRYPTO-PASSWORD		Removes the crypto password from the password table of the ongoing task.

2.4.3 Data security

Macros

Macro	Operands	Brief description
CATAL	BACKUP	Specifies the frequency of automatic saving.
	NUM_OF_BACKUP_VERS	Specifies whether the file is part of the version backup and, if yes, also the maximum number of file versions to be saved in the version backup archive.
COPFILE	REPLACE	Specifies whether an existing file is to be overwritten during copying.
CREPOOL	WROUT	Writes updated blocks in ISAM files back to disk immediately.
FILE	WRCHK	Performs a read-after-write check as a safeguard against recording errors.
	WROUT	Writes updated blocks in ISAM files back to disk immediately.
VERIF		Restores file structures, unlocks files.
WRCPT		Writes a checkpoint / creates a checkpoint file for restart with the RESTART-PROGRAM command.
FCB	EXIT	The address of an exit routine or of an EXLST macro.
	WRCHK	Performs a read-after-write check as a safeguard against recording errors.
EXLST		Defines exit routines for errors and other events.
LFFSNAP		Lists files from a Snapset
LJFSNAP		Lists job variables from a Snapset
RFFSNAP		Restores files from a Snapset
RJFSNAP		Restores job variables from a Snapset

Commands

Command	Operands	Brief description

CREATE-FILE
CREATE-FILE-GROUP
MODIFY-FILE-ATTRIBUTES
MODIFY-FILE-GROUP-ATTRIBUTES
	BACKUP-CLASS	Specifies the automatic backup frequency.

CREATE-FILE MODIFY-FILE-ATTRIBUTES	NUM-OF-BACKUP-VERS	Specifies whether the file is part of the version backup and, if yes, also the maximum number of file versions to be saved in the version backup archive.
COPY-FILE	REPLACE-OLD-FILES	Specifies whether an existing file is to be overwritten during copying.
CREATE-ISAM-POOL	WRITE-IMMEDIATE	Activates the WROUT function for SCOPE=TASK (updated blocks are written to disk immediately).
	SCOPE=HOST-SYSTEM	Activates the WROUT function implicitly.
ADD-FILE-LINK	SUPPORT=DISK (WRITE-CHECK)	Specifies a read-after-write check to prevent recording errors.
	SUPPORT=DISK(ISAM-ATTR=*PAR(WRITE-IMMEDIATE))	Writes updated records in ISAM files back to disk immediately (for SHARED-UPDATE=*NO)
RESTART-PROGRAM		Restarts an aborted program for which a checkpoint file has been created.
SECURE-RESOURCE-ALLOCATION		Reserves files or data volumes for a user job.
CHECK-FILE-CONSISTENCY REPAIR-DISK-FILE		Recovers file structures.
REMOVE-FILE-ALLOCATION-LOCKS		Unlocks files.
LIST-FILE-FROM-SNAPSET		Lists files from a Snapset
LIST-JV-FROM-SNAPSET		Lists job variables from a Snapset

RESTORE-FILE-FROM-SNAPSET		Restores files from a Snapset
RESTORE-JV-FROM-SNAPSET		Restores job variables from a Snapset

2.5 Device and volume management

DMS supports users in the processing of files on private volumes by making it possible for them to reserve volumes and devices for their jobs.

Macros

Macro	Operands	Brief description
FILE	DEVICE VOLUME	Defines the device type and volumes for a file on private disk.
	MOUNT	Issues a mount request at the console for private disks.
IMPORT		Creates (imports) catalog entries for files
RELTFT		Deletes TFT entries and implicitly releases devices

Commands

Makro	Operanden	Brief description
UNLOCK-FILE-LINK		Cancels a TFT lock: any preceding REMOVE-FILE-LINK request can now take effect and release private devices.
CREATE-FILE CREATE-FILE-GROUP CRE-FILE-GENERATION MODIFY-FILE-ATTRIBUTES MODIFY-FILE-GENERATION-SUPPORT MODIFY-FILE-GROUP-ATTRIBUTES	DEVICE-TYPE/ VOLUME	Defines devices and volumes for a file.
ADD-FILE-LINK	ADD-CATALOG-VOLUME	
CREATE-FILE CRE-FILE-GEN IMPORT-FILE MODIFY-FILE-GENERATION-SUPPORT MODIFY-FILE-ATTRIBUTES	SUPPORT=TAPE (PREMOUNT-LIST)	Issues a request to provide private volumes or Net-Storage volumes on the console

MODIFY-FILE-ATTRIBUTES MODIFY-FILE-GENERATION-SUPPORT	SUPPORT=DISK (VOLUME-ALLOCATION)	
ADD-FILE-LINK	NUMBER-OF-PREMOUNTS	
LOCK-FILE-LINK		Locks a TFT entry; prevents use of REMOVE-FILE-LINK and device being released
IMPORT-FILE		Creates (imports) catalog entries for files
REMOVE-FILE-LINK		Deletes TFT entries and implicitly releases devices
SECURE-RESOURCE-ALLOCATION		Reserves devices/volumes for a user job

2.6 Managing and using Net-Storage

DMS supports systems support in managing Net-Storage and the user in editing files on a Net-Storage volume.

Macros

Macro	Operands	Brief description
CATAL		Creates or updates catalog entries.
ERASE		Erases/exports files.
FILE	DEVICE=NETSTOR, VOLUME=	Defines device type and volume for a file on a Net-Storage volume.
	NFTYPE=	Defines the file type of a file on a Net-Storage volume: BS2000 file or node file.
	STATE=FOREIGN	Imports a file from a Net-Storage volume.
FSTAT	FROM=(vsn, NETSTOR)	Information from the catalog of the Net-Storage volume identified with "vsn".
	STOTYPE=*NETSTOR	Only files which reside on Net-Storage volumes are selected.
	VTOC=*YES	Displays the catalog entries from the catalog of a Net-Storage volume.
IMPORT	DEVICE=NETSTOR, VOLUME=	Imports catalog entries of files on a Net-Storage volume.
RDTFT		Requests information about TFT entries

Commands

Command	Operands	Brief description
MOUNT-NET-STORAGE		Connects Net-Storage.
UMOUNT-NET-STORAGE		Disconnects Net-Storage.
LIST-NET-DIRECTORIES		Lists Net-Storage directories.
SHOW-NET-STORAGE		Displays Net-Storage mounted in BS2000.
ADD-NET-STORAGE-VOLUME		Creates a Net-Storage volume and assigns it to a local pubset.
REMOVE-NET-STORAGE-VOLUME		Removes a Net-Storage volume from a local pubset.
SHOW-PUBSET-NET-STORAGE		Displays Net-Storage mounted in BS2000.

CREATE-FILE, MODIFY-FILE-ATTRIBUTES	STORAGE-TYPE= *NET-STORAGE, DEVICE-TYPE= NETSTOR, VOLUME=	The file is created on a Net-Storage volume.
	FILE-TYPE=	Creates the file as a BS2000 file or node file.
SHOW-FILE-ATTRIBUTES	STORAGE-TYPE=*NET-STORAGE	Selects files from Net-Storage.
	FROM-CATALOG=*NET	Displays catalog entries from the Net-Storage volume.
	FILE-TYPE=	Selects according to file type: BS2000 file or node file.
LIST-NODE-FILES		Lists node files on Net-Storage volume.
DELETE-FILE	STORAGE-TYPE=*NET-STORAGE	Selects files from Net-Storage.
	FILE-TYPE=	Selects according to file type: BS2000 file or node file.
IMPORT-FILE, CHECK-IMPORT-DISK-FILE	DEVICE-TYPE=NETSTOR, VOLUME=	Imports/checks catalog entries of files on Net-Storage volumes.
IMPORT-NODE-FILE		imports node files from Net-Storage.
EXPORT-FILE	STORAGE-TYPE=*NET-STORAGE, VOLUME=	Exports files from Net-Storage.
EXPORT-NODE-FILE		Exports node files.
SHOW-FILE-LINK		Displays information about TFT entries.
CHECK-FILE-CONSISTENCY, REMOVE-FILE-ALLOCATION-LOCKS, REPAIR-DISK-FILES	SELECT=*NET-STORAGE	Selects files from Net-Storage.

2.7 Access to files

Access to files is executed by calling action macros for the various access methods. DMS also handles the opening and closing of files (OPEN/CLOSE processing) as a function of the access method involved.

2.7.1 File processing

The DMS macros for file processing (i.e. the “service macros”) are macro calls which are valid for all access methods.

Macro	Brief description
FCB	Creates a file control block (FCB).
FCBAD	Creates the FCB in the literal pool of a program.
OPEN	Opens a file.
CLOSE	Closes one or more files.
EXLST	Defines error exits.
EXRTN	Implements a return from EXLST routines.
LBRET	Implements a return from user label handling routines (tape processing).

2.7.2 Macros specific to the access methods

The following access methods exist:

- BTAM
- DIV
- EAM
- FASTPAM
- SAM
- ISAM
- UPAM

In the tables below the macros for file access are assigned to the various access methods.

UPAM = User Primary Access Method

The basis for file processing is the standard block (= PAM page). UPAM supports blockoriented file access. One or more logical blocks or parts of logical blocks are transferred. UPAM can also be used to process files that were not created with UPAM.

Macro	Brief description
PAM	Controls all UPAM accesses
For event-driven processing, the following macros are also significant (for a detailed description, see the “Executive Macros” manual [2 (Related publications)]):	
CHKEI	Checks the queue status for an event item.
CONTXT	Accesses the register set of the interrupted task/process.
DISCO	Closes the routine for the contingency process.
DISEI	Disconnects the user program from the event item.
ENACO	Opens a routine as a contingency process and assigns it a name and a priority.
ENAEI	Creates an event item and/or establishes the link between the calling process and the event item.
FECB	Creates a file event control block.
LEVCO	Changes the priority of the called process.
POSSIG	Signals an event.
RETCO	Terminates the calling contingency process.
SOLSIG	Requests a signal from the event item.
SUSPEND	Places the calling process in an interruptible wait state.

FASTPAM = Fast Primary Access Method

The basis for file processing is the standard block (= PAM page). The basis for file processing is the standard block (= PAM page). FASTPAM supports block-oriented file access. One or more logical blocks or parts of logical blocks are transferred. FASTPAM can also be used to process files that were not created with FASTPAM.

Macro	Brief description
FPAMSRV	Management functions • enable the system environment (FASTPAM environment) • enable I/O areas (FASTPAM I/O area pool) • open a file for processing • close a file opened with FPAMSRV • disable the system environment (FASTPAM environment) • disable I/O areas (FASTPAM I/O area pool)
FPAMACC	File access functions • synchronous reading and writing of logical blocks • asynchronous reading and writing of logical blocks • wait for the end of asynchronous I/O jobs • report the end of asynchronous I/O jobs

DIV = Data In Virtual

DIV allows files to be mapped directly to a virtual address space. Transfer is performed “on demand”, initiated by a page fault when accessing a memory page in a window. DIV can also be used to process files that were not created with DIV.

Macro	Brief description
DIV	Process files with the DIV access method • open a file • define a window (i.e. a work area in virtual address space) • write modified pages from the window back to the file on disk • undo changes in the window • release windows in virtual address space • close a file, releasing any existing windows with default values, if applicable

SAM = Sequential Access Method

A SAM file is a sequence of records (contained in logical blocks). DMS allows the user to process records sequentially in either direction (beginning-of-file to end-of-file or vice versa). For tape processing, SAM complies with all requirements of DIN 66029 up to exchange level 3. Files with either standard or nonstandard blocks can be processed.

Macro	Brief description
FEOV	Initiates a tape swap.
GET	Retrieves the next record.
PUT	Writes the next record to the current end of the file.
PUTX	(Locate mode only) replaces a record in the buffer to which processing has been positioned by means of GET.
RELSE	Terminates a data block.
SETL	Positions to beginning-of-file, to end-of-file, or to a record which is to be retrieved by means of GET.

BTAM = Basic Tape Access Method

BTAM is an access method for block-oriented tape processing; it can also be used to process tape files which were not created with BTAM. During processing of a tape file, the direction in which the file is processed can be changed as desired, and tapes can be positioned to any desired block or section. BTAM processes files with or without standard blocks.

Macro	Brief description
BTAM	Controls all BTAM actions.
FEOV	Initiates a tape swap.
NDWERINF	Interrogates the status bytes.

EAM = Evanescent Access Method

EAM is used to process temporary files in the SYSEAM area. It is a block-oriented access method and is particularly suitable for rapid processing of job-specific work files.

Macro	Brief description
EAM	Controls all EAM accesses

ISAM = Indexed-Sequential Access Method

An ISAM file consists of a set of records. Each record contains a key and the keys are the criterion for sorting the records into blocks. The highest key in each index or data block is placed, together with the block number, in an index block at the next higher level of the index.

Macro	Brief description

ELIM	Deletes a record from the file.
GET	Reads the next record from the file (sequential read).
GETFL	If flagged ISAM keys are used: reads the next record within the flag range (sequentially).
GETKY	Reads the first record with the specified key.
GETR	Reads the previous record (sequential reverse read).
INSRT	Inserts a record into the file with a new ISAM key.
ISREQ	Clears an ISAM lock.
OSTAT	Informs the caller about the number and type of concurrent file accesses.
PUT	Sequentially writes records to the end of the file (and also checks that the keys are in the right order).
PUTX	Replaces a record read by means of GET or a similar macro.
RETRY	After execution of the EXLST PGLOCK exit, resets the ISAM pointer and repeats the last macro.
SETL	Positions the ISAM pointer to the beginning of the file, to the end of the file or to a specific record for subsequent sequential processing.
STORE	Inserts a record into the file with a new ISAM key. Overwrites a record with an existing ISAM key if duplicate ISAM keys are not permitted, or Inserts into the file a record with an existing ISAM key as the last record with this key.

The ISAM pool macros are listed on "Controlling file processing".

2.8 Generation of operand lists for control blocks, DMS tables, etc.

The user can generate program areas or DSECTs (Dummy SECTions) which permit him/her to access the contents of DMS tables, file control blocks, etc. or the operand lists of DMS macros with the aid of symbolic addresses.
For most macros this is possible with the aid of the MF operand, alternatively there are special DSECT macros. In the case of "older" macros (e.g. CATAL) which were only converted in a later version, the VERSION operand decides whether the special DSECT macro must still be used or whether specification using the MF operand is possible.

Control blocks and macros specific to access methods

Macro	Brief description
IDBPL	Operand list for the BTAM macro
IDECB	For the UPAM file event control block (FECB).
IDFCB	For the file control block (FCB) of the user program at the TU level.
IDFCBE	Extension of the 24-bit TU FCB.
IDMCB	EAM control block for EAM macro.
IDPPL	Operand list for the PAM macro.
IDOST	Operand list for the OSTAT macro.

DMS tables, file catalog, etc.

Macro	Brief description
IDCE	Catalog entry.
IDCEG	Catalog entry (extension for file generation groups).
IDCEX	Catalog entry (extension).
IDEE	Catalog entry (extent list).
IDEMS	DMS error messages.
IDTFT	TFT entry.
IDVT	Volume label entry (in the volume table).

2.9 Output of information on files, volumes, devices, etc.

Various DMS macros are provided enabling information on catalog entries, file status, the task file table, device and volume allocation, etc. to be requested in the program at any time and utilized for further processing.

Macros

Macro	Brief description
FSTAT	Requests information from the file catalog or about the catalog entries for files
RDTFT	Requests information about TFT entries
OSTAT	Requests information about the number and types of accesses to an ISAM file by various jobs
SHOPOOL	Requests information on ISAM pools
SHOPLNK	Requests information on ISAM pool link names
SHOWAIX	Requests information on the secondary keys (alternate indices) of an ISAM file

Commands

Command	Brief description
SHOW-FILE	Outputs a file to SYSOUT
SHOW-FILE-ATTRIBUTES	Displays information from the file catalog or on catalog entries
SHOW-FILE-LINK	Requests information about TFT entries
SHOW-FILE-LOCKS	Displays information about file locks
SHOW-DEVICE-STATUS	Displays information on the devices reserved online by the user job
SHOW-DISK-DEFAULTS	Displays information on the default values defined for private disks by the operator
SHOW-INDEX-ATTRIBUTES	Displays information on existing secondary indices of an NK-ISAM file
SHOW-ISAM-POOL-ATTRIBUTES	Displays information on the attributes and usage of NK-ISAM pools
SHOW-ISAM-POOL-LINKS	Displays information on the assignment of pool link names to NK-ISAM pools
SHOW-MOUNT-PARAMETER	Displays information on the values set by the operator for the mounting/dismounting of volumes

SHOW-PUBSET-FILE-SERVICES	Displays information on the services provided by an SM pubset
SHOW-RESOURCE-ALLOCATION	Displays information on device and volume reservations as well as on outstanding operator actions for the requesting job
SHOW-NET-STORAGE	Displays Net-Storage mounted in BS2000.
SHOW-PUBSET-NET-STORAGE	Displays a pubset's Net-Storage.

3 Volumes

In BS2000 there are public and private volumes and also Net-Storage. Only disks can be public volumes.
Net-Storage does not belong to the public volumes, but is used to extend them. With respect to the device type, Net-Storage in BS2000 is regarded as belonging to the group of disks.
Private volumes are all tapes and those disks which are not marked as public disks. Both types of volumes can be used together.

In the definition of disks, DMS distinguishes between public volumes, which are grouped into pubsets, and private disks. Net-Storage volumes are used to extend pubsets.

All volumes in BS2000 are identified by a name of up to six characters. This name is referred to as VSN (volume serial number) or archive number. Any of the alphanumeric characters A..Z, 0..9 and the special characters period (.), @, # and $ may be used in the name. Pubset volumes can be distinguished from other volumes by their VSN where the VSNs of public volumes are subject to a convention: they must begin with the string “PUB” or contain a period in the third, fourth or fifth position. This VSN syntax may not be used for private disks and Net-Storage volumes.

DMS manages the storage space on disks. Some control functions (access authorization, saturation control, etc.) can only be used on a pubset, not on private disks or Net-Storage.

3.1 Public volumes (pubsets)

Public disks are grouped into volume sets or single-feature (SF) pubsets. One or more volume sets form a system-managed (SM) pubset. A multiple public volume set (MPVS) enables several pubsets to be simultaneously installed in one and the same session. Net-Storage volumes can be mounted on a pubset. However, they are not part of the pubset (IMPORT-PUBSET is also possible without mounted Net-Storage volumes).

One special pubset called the “home pubset” must always be available whenever the system is running, since it contains the data needed for loading, operating and terminating the system. The other pubsets can be imported and exported by systems support. Users' default pubsets should always be imported.

All volumes of a pubset are handled and managed by the system as a single unit. Every user who has been authorized by systems support with the ADD-USER command or the system parameter FSHARING may use any DMS function to create, process and delete files on imported pubsets.

There are two different types of pubset: SF pubset (Single Feature pubset) and SM pubset (System Managed pubset). They have different features, operating parameters and service attributes.

SF pubsets

All volumes of an SF pubset have the same physical features. An SF pubset offers a limited and homogeneous set of services.

SM pubsets

An SM pubset consists of one or more volume sets. Each volume set must be
homogeneous with respect to the features of its volumes. No file can be distributed over more than one volume set. Each volume set in an SM pubset represents a specific service type which is taken into account when the storage location is selected for a file. The selection is based on the user's specification (via the logical file attributes) of the services requested.

Pubset operating modes

Shared pubset

Where the product MSCF is used with an appropriate hardware configuration, it is possible to access a common pubset simultaneously from BS2000 systems. Up to 16 BS2000 systems can be linked in a common MSCF network, and can access this shareable pubset as „sharer“ via a direct hardware path. One of these networked processors is named as the temporary owner of the pubset, and it then manages the files, the users and the accesses

to the pubset for all the other systems. All management requests from a subordinate processor, known as a “slave sharer”, must be directed via MSCF to the owner system (the “master”). If the owner system crashes, a pubset-specific job variable is set in all the dependent systems. If a backup owner was defined by systems support with the SET-PUBSET-ATTRIBUTES command, this slave sharer can assume the role of the crashed master processor (master change). If no such backup owner is defined, systems support can then export the pubset to the slave sharer and specify a new owner in a subsequent IMPORT-PUBSET command.

The overall concept of shared pubsets (hardware configuration, pubset management and data access) is described in detail in the “HIPLEX MSCF” manual [11 (Related publications)].

XCS pubset

HIPLEX MSCF offers an enhanced network functionality: the XCS (cross-coupled system) XCS results in a closer coordination of the networked systems. Each system has a consistent and complete view of the entire network. The XCS network thus offers mechanisms for implementing distributed applications. It is primarily designed for availability and load sharing in BS2000. The XCS network thus offers mechanisms for implementing distributed applications.

- Distributed lock manager (DLM):
  This function implements cross-system lock management, thus supporting crosssystem synchronization and serialization. SFS (see below) is based on this function.
- Shared file system (SFS):
  SFS permits cross-system updating of files on shared pubsets within the XCS network; the pubsets do not necessarily have to be XCS pubsets. HIPLEX MSCF supports this global shared update for both block-oriented and byte stream-oriented access methods i.e. UPAM, FASTPAM and DIV.

The following conditions apply to XCS networks:

- Any one system can be part of no more than one XCS network.
- The network must be fully intermeshed, i.e. all systems participating in the network must be interconnected via MSCF connections.
- The XCS network must include at least one XCS pubset, and access paths to this pubset must exist from all systems in the network.

An XCS pubset is the central storage location for all data that must be accessible on a network-wide basis. XCS pubsets are automatically imported by the system.

The overall concept of the XCS network (functionality, generation and operation) is described in detail in the “HIPLEX MSCF” manual [11 (Related publications)].

Pubset copies created using SHC-OSD

Special pubset copies (mirror disks) can be created locally on the disk storage system concerned using SHC-OSD in order to back up pubsets in disk storage systems. Two variants are available:

- Pubset copy created using Clone Units (e.g. for backing up a consistent status of the pubset with HSMS or FDDRL)
- Pubset copy created using Snap Units, also called a Snapset (see the section “Pubsetbackup with Snapsets”)

Pubset copies are permanently assigned to the pubset. The VSNs of their volumes comply with a special syntax (see “Special VSN name for pubset copies” in the next section). Only read access is permitted to these copies. For details, see the manuals “HSMS” [10 (Related publications)] and “Introduction to System Administration [7 (Related publications)].

Naming conventions for pubsets

The syntax of the names of disks assigned to a volume set/SF pubset must match that of the name of the volume set/SF pubset. A distinction must be made between single- and multiple-character volume set and pubset names:

- the PUB notation must be used for single-character names
- the period notation must be used for names with two to four characters.

Default names of Net-Storage volumes which are assigned to a pubset have a special format depending on the pubset notation, see "Notation of Net-Storage volumes" in the "Introduction to System Administration" [7 (Related publications)].

VSN in PUB notation

The character string “PUB” is a mandatory and invariable component of any VSN formed using this type of pubset addressing. The Term “PUB” stands for public and thus identifies the disk as a shared disk.

A VSN in PUB notation always consists of 6 characters and has the following format:

`PUBpxx`

PUB	invariable component to distinguish public from private disks (3 characters “PUB”) = type identifier
p	catalog ID (catid), (1 character; A..Z, 0..9)
`xx`	sequence number within the pubset/volume set, (2 characters; 00..31)

Up to 36 pubsets or volume sets, comprising up to 32 disks each, can be addressed with a single-character catalog ID.

Examples: PUBA00, PUBA25, PUB502

VSN in period notation

A period is used to separate the sequence number within the pubset/volume set from the catalog ID in any VSN formed using this type of identification. A VSN in period notation always identifies a public disk.

A VSN in period notation always consists of 6 characters and has the following format:

`pp[pp].[xy]z`

`pp[pp]`	catalog ID (catid), (2-4 characters from the range A..Z, 0..9), the prefix “PUB” is not permissible
.	period (1 character) = type identifier
`[xy]z`	sequence number within the pubset/volume set, (1-3 characters from the range 0..9)

A pubset or volume set in period notation may include up to 255 disks.

Examples: AA.001, AB.309, XYZ.23, OTTO.0, J19P.8

The catalog ID of an SF pubset is identical with the “catid” part of the VSN. The catalog ID of an SM pubset is always different from all volume set names of this SM pubset and thus also different from the “catid” part of the VSNs of all disks of the pubset.

Special VSN name for pubset copies

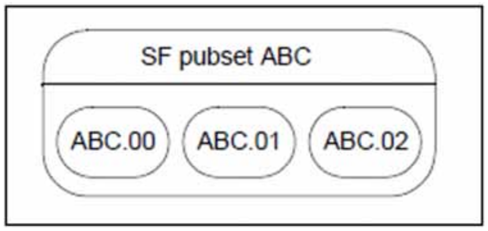
The VSNs of a pubset copy created using Clone Units are formed from the original VSNs. In the case of PUB notation the “U” in the “PUB” string is replaced by a colon, and in the case of dot notation the dot is replaced by a colon.

The VSNs of a pubset copy created with SHC-OSD using Snap Units, i.e. of a Snapset, are formed from the original VSNs in accordance with the following rule:

- Up to the 26th Snapset (i.e. for Snap IDs a through z):
  - In the case of PUB notation the “U” and “B” in the “PUB” string are each replaced by the Snapset ID in lowercase letters.
  - In the case of dot notation the dot is replaced by the Snapset ID in lowercase letters.
- From the 27th Snapset (i.e. for Snap IDs A through Z):
  - In the case of PUB notation the “P” and “U” in the “PUB” string are each replaced by the Snapset ID in lowercase letters.
  - In the case of dot notation the dot is replaced by the Snapset ID in lowercase letters and at the same time its character position is changed:
    - In the case of a 4-character catid the Snapset ID is inserted after the first character of the catid.
    - In the case of a 3-character catid the Snapset ID is inserted before the first character of the catid.
    - In the case of a 2-character catid the Snapset ID is inserted as the last character of the VSN.

Examples

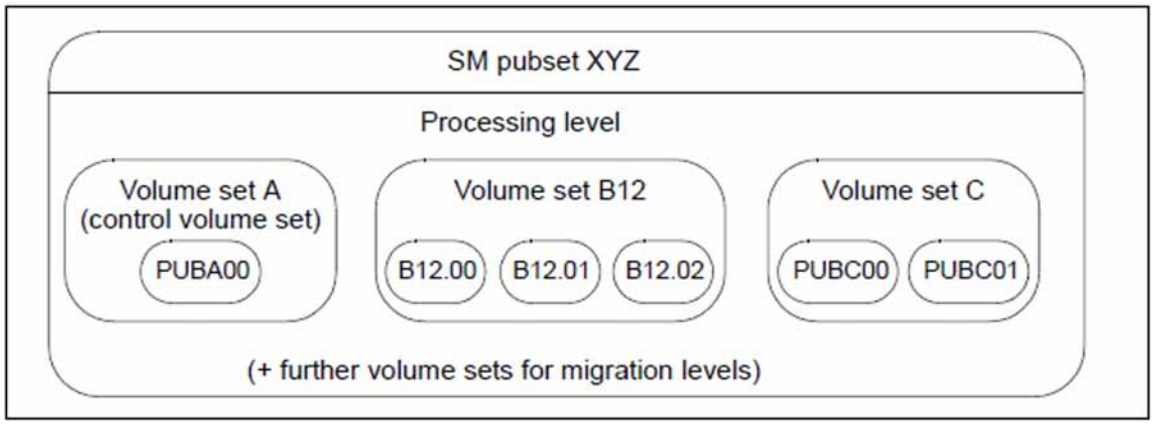
Volume configuration of an SF pubset:



The SF pubset with the catalog ID `ABC` consists of three disks.
Assuming a file whose file name is “MY.LIST” exists under user ID “MISCELLA” on any of the disks of SF pubset `ABC`, the path name to that file would be:
“:ABC:$MISCELLA.MY.LIST”.

An associated Snapset with the Snapset ID `x` would then consist of the volumes with the VSNs `ABCx00`, `ABCx01` and `ABCx02`.

Volume configuration of an SM pubset:



The SM pubset with catalog ID XYZ consists of three volume sets.
Assuming a file with the file name “PHONELIST” exists under user ID “GENERAL” on any of the disks belonging to any of the volume sets of the SM pubset XYZ, the path name to that file would be: “:XYZ:$GENERAL.PHONELIST”; this means that the internal structure of the SM pubset and the exact location of the file are irrelevant for the way it is addressed.

An associated Snapset with the Snapset ID `m` would then consist of the volumes with the VSNs `PmBA00`, `B12m00`, `B12m01`, `B12m02`, `PmBC00` and `PmBC01`.

User catalog

Each pubset contains a user catalog and its own file catalog.

Each entry in the JOIN file determines who, or which user ID, may access a pubset. Users who have no JOIN entry for a pubset cannot access any file in that pubset, even if such files are shareable, unless this is globally permitted by the system parameter FSHARING.

The JOIN file also defines the pubset-specific rights of each user listed in it. If the JOIN entry contains the specification PUBLIC-SPACE-LIMIT=0, for example, the user cannot create any permanent files on that pubset but may access all the shareable files cataloged in it. The creation of temporary files on the pubset may be suppressed with TEMP-SPACE-LIMIT=0 (see "Requesting storage space" for details on storage space management).
Specifying NET-STORAGE-USAGE=*ALLOWED specifies that the user may occupy storage space on Net-Storage.

Any user can obtain information about his/her JOIN file by means of the SHOW-USER-ATTRIBUTES command.

3.2 Private disks

Private disks, in contrast to public volumes, must be requested separately for file processing. They can be used exclusively by one job or be shared by several jobs which run simultaneously.
Normally the operating mode is “shareable”.
Exclusive use for one user job can be selected by means of the SECURE-RESOURCE-ALLOCATION command.
With the aid of special commands, the operator can disable the individual allocation modes.

File processing on private disks is exactly the same as for the files in an MPVS environment.

If a user without remote catalog access (RCA) wishes to access a file created by another system, he/she must first report this file to his/her own system (TSOSCAT) by means of a FILE macro or IMPORT-FILE command.

By means of the operand VTOC=YES/NO in the FSTAT macro or INFORMATION=... in the IMPORT-FILE command, the user can have the VTOC catalog entries of the private disks displayed instead of the TSOSCAT entries. The VTOC entry of the private volume replaces the corresponding TSOSCAT entry.

3.3 Net-Storage

BS2000 permits UNIX file systems to be accessed via NFS.
Files from BS2000 and from open systems can consequently be stored and edited in the server network in the storage space released by file servers.

The following terms are used in BS2000 when working with Net-Storage:

Net server

A file server in the worldwide computer network which provides storage space (Network Attached Storage, NAS) for use by other servers and offers corresponding file server services.

Net-Storage

Storage space provided by a net server in the computer network and storage space released for use by foreign servers. Net-Storage can be a file system or also just a node in the net server’s file system.

Net client

Implements access to Net-Storage for the operating system using it.
In BS2000 the net client, together with the BS2000 subsystem ONETSTOR, transforms the BS2000 file accesses to corresponding UNIX file accesses and executes them on the net server using NFS.

The net client for SUs /390 and S servers is the HNC, and for SUs x86 the X2000 carrier system.

Net-Storage volume

Net-Storage volumes represent Net-Storage in BS2000 which provides systems support as an enhancement of data pubsets.
Net-Storage volumes are addressed by means of their Volume Serial Number (VSN) and the volume type NETSTOR. In the released file system of the net server the VSN of the Net-Storage volume corresponds to the directory containing the user files and metadata.

i It is the task of systems support to provide and manage Net-Storage, see the "Introduction to System Administration" [7 (Related publications)].

Net-Storage file

Is a file which is created on a Net-Storage volume. On Net-Storage a distinction is made between the two file types BS2000 file and node file.

BS2000 file

Is a file which is only created and processed by BS2000. BS2000 files on Net-Storage (FILE-TYPE=BS2000) have been supported since BS2000/OSD-BC V9.0. They are located directly on a Net-Storage volume. Open systems may only access them in read mode.

Node file

Is a Net-Storage file (FILE-TYPE=NODE-FILE) which can be created and processed by both BS2000 and open systems. Node files have been supported since BS2000 OSD/BC V10.0. They are located on a Net-Storage volume in a user-specific directory (name of the user ID), and the file names comply with the BS2000 naming conventions.

Interoperability on Net-Storage

Both BS2000 and open systems can be connected to the same Net-Storage, create node files there, and can each process these:

- BS2000 can create and process node files on Net-Storage - open systems can also process these files
- Open systems can create and process node files on Net-Storage - BS2000 can import and also process these node files.

3.3.1 Access from BS2000 to Net-Storage

The BS2000 user accesses a file on Net-Storage via DMS interfaces and the net client on the net server in the following steps:

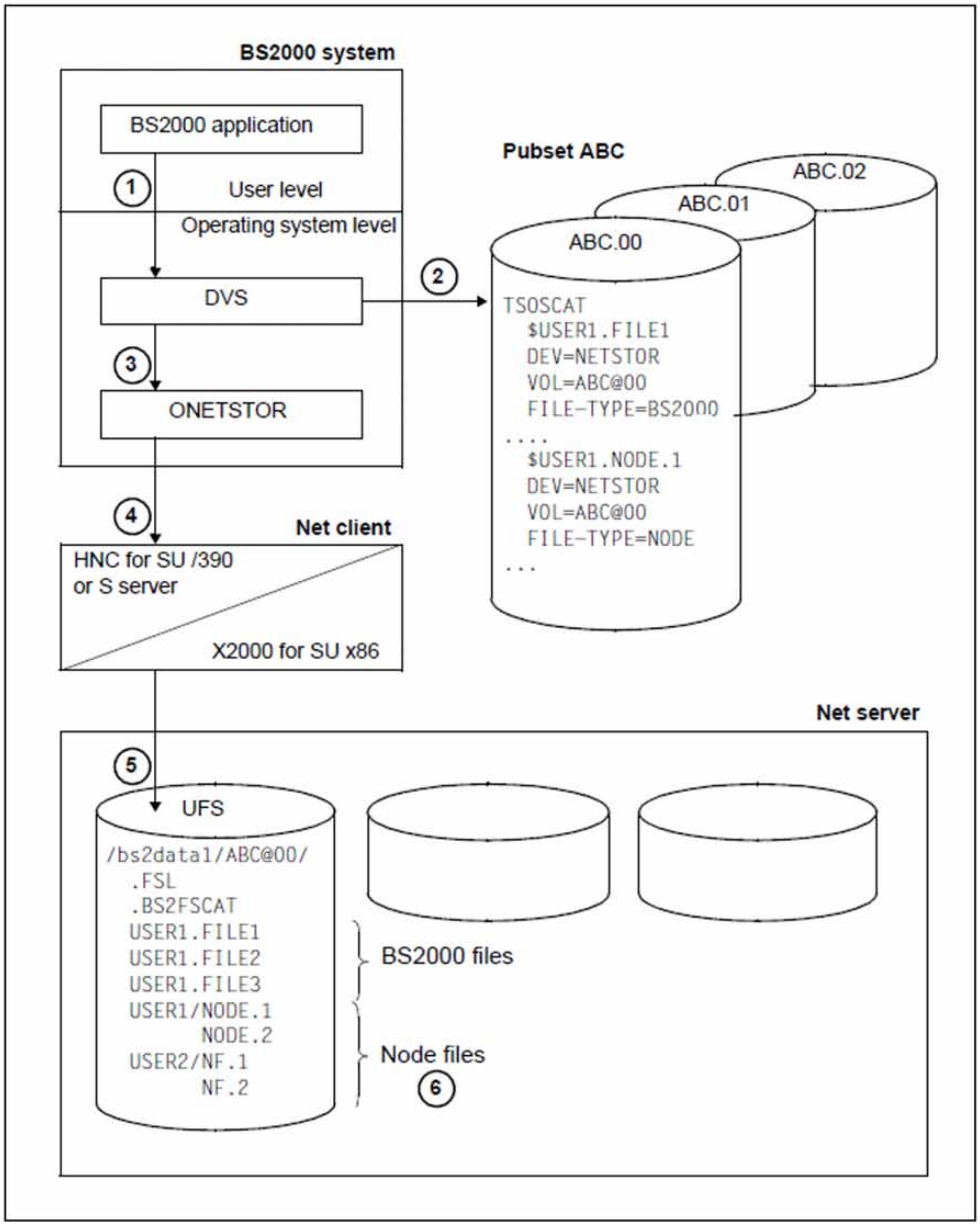
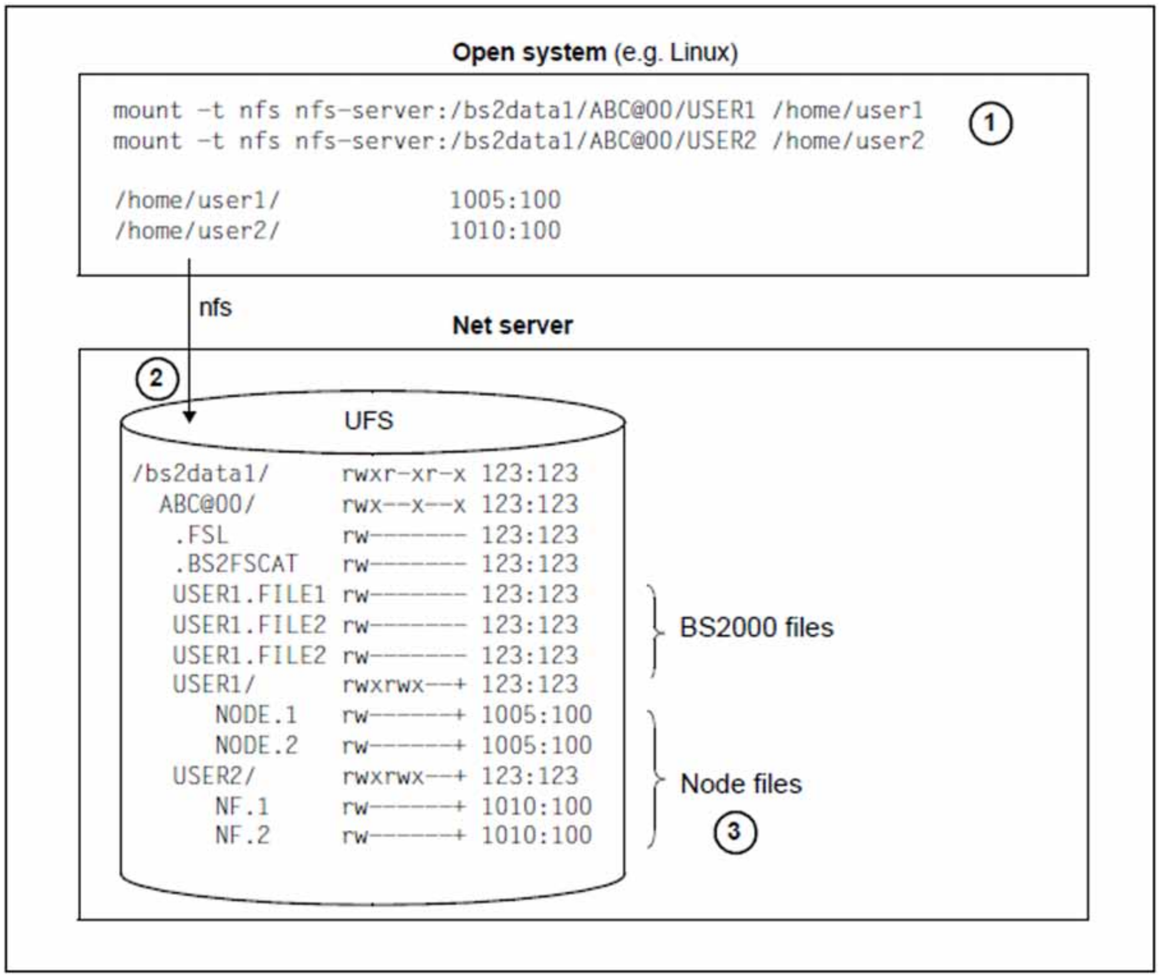


Figure 1: BS2000 access to Net-Storage

1. A BS2000 application under the user ID USER1 wishes to access the FILE1 file which resides on Net-Storage and is cataloged in the ABC pubset. This is done via the normal user interfaces of DMS, see section “Working with Net-Storage”.

2. DMS checks whether the file exists in the user and file catalogs of the `ABC` pubset. On the basis of the file attributes DEVICE=NETSTOR and VOLUME=ABC@00, DMS recognizes that the file concerned is contained in the `ABC@00` directory on the Net-Storage released by the net server.
3. The `FILE1` file is actually accessed via the BS2000 subsystem ONETSTOR and the net client.
4. The BS2000 subsystem ONETSTOR transforms the BS2000 file access to the corresponding file access in the UNIX file system (UFS) and forwards it to the net client.
5. On the net client the `/bs2data1` directory (with the `ABC@00` subdirectory) released by the net server is mounted. File access takes place via NFS in the net server's UFS.
6. When accessing a node file, e.g. the file `NODE.1`, DMS recognizes that a node file (FILE-TYPE=NODE) in the `ABC@00` directory on the Net-Storage released by the net server is concerned. Node files reside in a user-specific directory which has the name of the user ID, in this case the directory `USER1`. This directory is created automatically when BS2000 creates the user's first node file.
   To enable a user to work with node files, the POSIX user attributes user number and group number must be entered in the pubset's user entry. For the user `USER1`, for instance, 1005 and 100 are entered (see figure 2 (Access of open systems to Net-Storage)).

3.3.2 Access of open systems to Net-Storage



```
mount -t nfs nfs-server:/bs2data1/ABC@00/USER1 /home/user1
mount -t nfs nfs-server:/bs2data1/ABC@00/USER2 /home/user2

/home/user1/             1005:100
/home/user2/             1010:100
```

```
/bs2data1/       rwxr-xr-x 123:123
  ABC@00/        rwx--x--x 123:123
   .FSL          rw------- 123:123
   .BS2FSCAT     rw------- 123:123
   USER1.FILE1   rw------- 123:123
   USER1.FILE2   rw------- 123:123
   USER1.FILE2   rw------- 123:123
   USER1/        rwxrwx--+ 123:123
      NODE.1     rw------+ 1005:100
      NODE.2     rw------+ 1005:100
   USER2/        rwxrwx--+ 123:123
      NF.1       rw------+ 1010:100
      NF.2       rw------+ 1010:100
```

Figure 2: Access of open systems to Net-Storage

1. Open systems obtain access to BS2000 files and node files on the net server by means of mount commands.
2. When the user-specific directories are created, the POSIX ACLs are also supplied with the required information by BS2000. Under Linux the users `user1` and `user2` in the example above therefore obtain access to the directories `USER1` and `USER2`.
3. To grant a user read and write access from both BS2000 and from the open system, the same UserId and GroupId must be entered for him/her on both systems. The same applies for creating and deleting files.

i When NFSv4 is used, this must be guaranteed via a directory service, e.g. openLDAP or Active Directory, to which both the net client, the NFS server and the open system are connected.

3.3.3 Working with Net-Storage

After Net-Storage volumes have been created, BS2000 users can utilize them via the interfaces of DMS if this is permitted in their user entry. The available storage space on Net-Storage is then limited only by the size of the released storage space on the net server.

- You can use the DMS commands with the STORAGE-TYPE operand specified to create and edit files on Net-Storage volumes, e.g. with

  `/CREATE-FILE` or `/MODIFY-FILE-ATTRIBUTES`

  ```
                FILENAME=...,
                SUPPORT=*PUBLIC-DISK(STORAGE-TYPE=*NET-STORAGE,..)
  ```

  Catalog entries are generated in both the local pubset and in the Net-Storage volume's catalog. The files are also created in the UNIX file system with the file size 0. See also "General conditions".

  The system determines the Net-Storage volume on which the file is created as follows:

  - If a standard Net-Storage volume exists for the pubset specified under FILE-NAME, this is selected automatically.
  - If no standard Net-Storage volume exists for the specified pubset, a non-standard Net-Storage volume is selected. If more than one non-standard Net-Storage volume exist, the system determines a volume itself.

  The user must therefore specify a volume when he/she wishes to choose a particular non-standard Net-Storage volume from multiple existing non-standard Net-Storage volumes:

  ```
  /CREATE-FILE FILE-NAME=...,SUPPORT=*PUBLIC-DISK(STORAGE-CLASS=*NONE(
                VOLUME=NETS00, DEVICE=NETSTOR))
  ```

  The user can employ the SHOW-PUBSET-NET-STORAGE command to obtain information on the Net-Storage volumes assigned to a pubset.

  If the file type is not specified, a BS2000 file is created by default. To create a node file, you must specify the file type explicitly, e.g. with

  ```
  /CREATE-FILE FILE-NAME=...,
               SUPPORT=*PUBLIC-DISK(
                  STORAGE-TYPE=*NET-STORAGE(FILE-TYPE=*NODE-FILE),
                      STORAGE-CLASS=*NONE(VOLUME=NET001,DEVICE=NETSTOR))
  ```

  In this case, if it does not already exist a user-specific directory is generated in which the specified file is created. This directory is also assigned the POSIX ACLs required to guarantee the user access from open systems, too.

- The SHOW-FILE-ATTRIBUTES command displays the file attributes and the storage location (volume and device type NETSTOR) of the file. By default the information from the associated pubset's TSOSCAT is displayed. You can request information from the catalog of a Net-Storage volume as follows:

  ```
  /SHOW-FILE-ATTRIBUTES ...,
          SELECT=*BY-ATTRIBUTES(FROM-CATALOG=*NET(VOLUME=...))
  ```

- You can use the DMS commands with the STORAGE-TYPE selection operand specified to select files on Net-Storage volumes, e.g. with

  `/SHOW-FILE-ATTRIBUTES` oder `/DELETE-FILE`

  ```
          SELECT=*BY-ATTRIBUTES(STORAGE-TYPE=*NET-STORAGE,..,FILE-TYPE=...)
  ```

  With the FILE-TYPE operand you can restrict the selection to BS2000 files or node files.

- You can import BS2000 files which are not yet known in BS2000 from Net-Storage, i.e. create a catalog entry in the local pubset, e.g. with:

  ```
  /IMPORT-FILE SUPPORT=*DISK(VOLUME=P@BA00,DEVICE-TYPE=NETSTOR,...,PUBSET=A)
  ```

  In this case the Net-Storage volume must be assigned to the pubset into whose catalog the entries are to be imported. Systems support can assign a Net-Storage volume which has not yet been assigned using ADD-NET-STORAGE-VOLUME.

- You can also simulate the import in advance, e.g. with:

  ```
  /CHECK-IMPORT-DISK-FILE VOLUME=P@BA00,DEVICE-TYPE=NETSTOR,PUBSET=A,...
  ```

- You can import node files which are not yet known in BS2000 from Net-Storage, i.e. create a catalog entry in the local pubset, e.g. with:

  ```
  /IMPORT-NODE-FILE VOLUME=P@BA00, FILE-NAME=...[,FILE-STRUCTURE=...]
  ```

  The catalog entry is generated from the Inode attributes of the node file. The default setting for the FILE-STRUCTURE operand is *STD. This is equivalent to specifying *PAM, meaning the file is imported as a PAM file (FILE-STRUC=PAM file attribute). Node files containing binary data (e.g. types data, tar, executable etc.) should be imported as PAM files. Node files containing text based data (type text) should be imported as SAM files by explicitly specifying FILE-STRUCTURE=*SAM (FILE-STRUC=SAM and REC-FORM=V file attribute). For details on processing SAM node files, see section “Working with SAM node files”.

  "Normal" files can be imported only if their names comply with the BS2000 conventions.

- You can query the information on the node files on a Net-Storage volume irrespective of whether they are already known in BS2000. However, only node files which comply with the BS2000 naming conventions are displayed, i.e. files which can also be imported. You can obtain information on your own node files with, for instance,

  ```
  /LIST-NODE-FILES VOLUME=P@BA00, USER-DIRECTORY=*OWN, NODE-FILE-NAME=...
  ```

- You can export files on Net-Storage volumes using the STORAGE-TYPE operand, e.g. with

  ```
  /EXPORT-FILE ...,SELECT=*BY-ATTRIBUTES(STORAGE-TYPE=*NET-STORAGE,...)
  ```

  You can export node files with /`EXPORT-NODE-FILE`.

  > i It makes sense to export individual files when inconsistencies are corrected manually (e.g. after the failure of the Net-Storage).

- The CHECK-FILE-CONSISTENCY, REMOVE-FILE-ALLOCATION-LOCKS and REPAIR-DISK-FILES commands support operation of Net-Storage by means of the selection operand `SELECT=*NET-STORAGE`.

- The DMS program interfaces also support Net-Storage, e.g. the CATAL, ERASE, FILE, FSTAT, IMPORT and RDTFT macros.
  A file on a Net-Storage volume is accessed in the conventional manner by specifying the catalog ID, user ID and file name.

- The BS2000 software products HSMS and ARCHIVE enable you to save and restore files on Net-Storage like local files, see the "HSMS" manual [10 (Related publications)].

- Files on Net-Storage can be used directly in the UNIX or Windows system of the net server, e.g. BS2000 files as control or print files:
  - The CONVERT-FILE-TO-PDF command generates PDF files in a binarycompatible format.
  - The software product BS2ZIP generates zip archives in a WINZIP-compatible format.

> i Linux or Windows applications may only access BS2000 files in read-only mode. When Linux or Windows applications are also to access the files in write mode, node files must be used.

3.3.4 Working with SAM node files

From the BS2000 viewpoint, PAM files are non-structured and can be used for storing any binary content, SAM files, on the other hand, consist of sequences of records and can be used for storing text based content. To enable an exchange of text based data between BS2000 and systems of the open world, SAM files can be stored on Net-Storage as SAM node files.

3.3.4.1 SAM block structure in BS2000

In BS2000, SAM files are processed sequentially by using the record-oriented SAM access method. Regarding Net-Storage and node files, only SAM processing on disk is relevant, meaning that special aspects of tape processing do not play a role in this context. Only NK-SAM files in the format BLKCTRL=DATA and with variable-length records (RECFORM=V) are supported as SAM node files on Net-Storage.

For NK-SAM files with these file attributes, the SAM access method creates data blocks with the following structure:

- The logical block, which consists of n PAM pages (or logical halfpages LHP, (STD,n) with n = 1..16), starts with a 12-byte BLKCTRL field and a 4-byte data length field. The data length field specifies the length of the user data in the logical block.
- These are followed by the SAM records, each consisting of a record length field and data. The data between the user data and the end of the logical block are undefined. During the sequential writing of the file, the logical blocks are filled record by record (PUT macro), until the next record in the sequence cannot be fitted into the current block anymore. This record is then written into the next logical block.
- However, an application may also “close” the data block before it is completely full and start a new data block (RELSE macro).

The SAM access method, when reading these NK-SAM files, expects the data blocks to follow this structure. For reading the data blocks the following applies:

- Sequential reading is done via GET. Closing the current block and reading the next block is done via RELSE followed by GET.
- Within the file, SETL can be used to position to a specific block and record.

For details on the access method, see chapter “SAM – Sequential Access Method” (SAM - Sequential Access Method).

3.3.4.2 SAM converter

In comparison to the data structure expected by SAM, text based data of node files are, with the exception of line feeds (LF) used to separate records, non-structured. Therefore, the SAM access method needs an “interpreter”, the SAM converter, that can map the SAM specific processing of a BS2000 disk file onto the node file. The SAM converter is part of the net client.

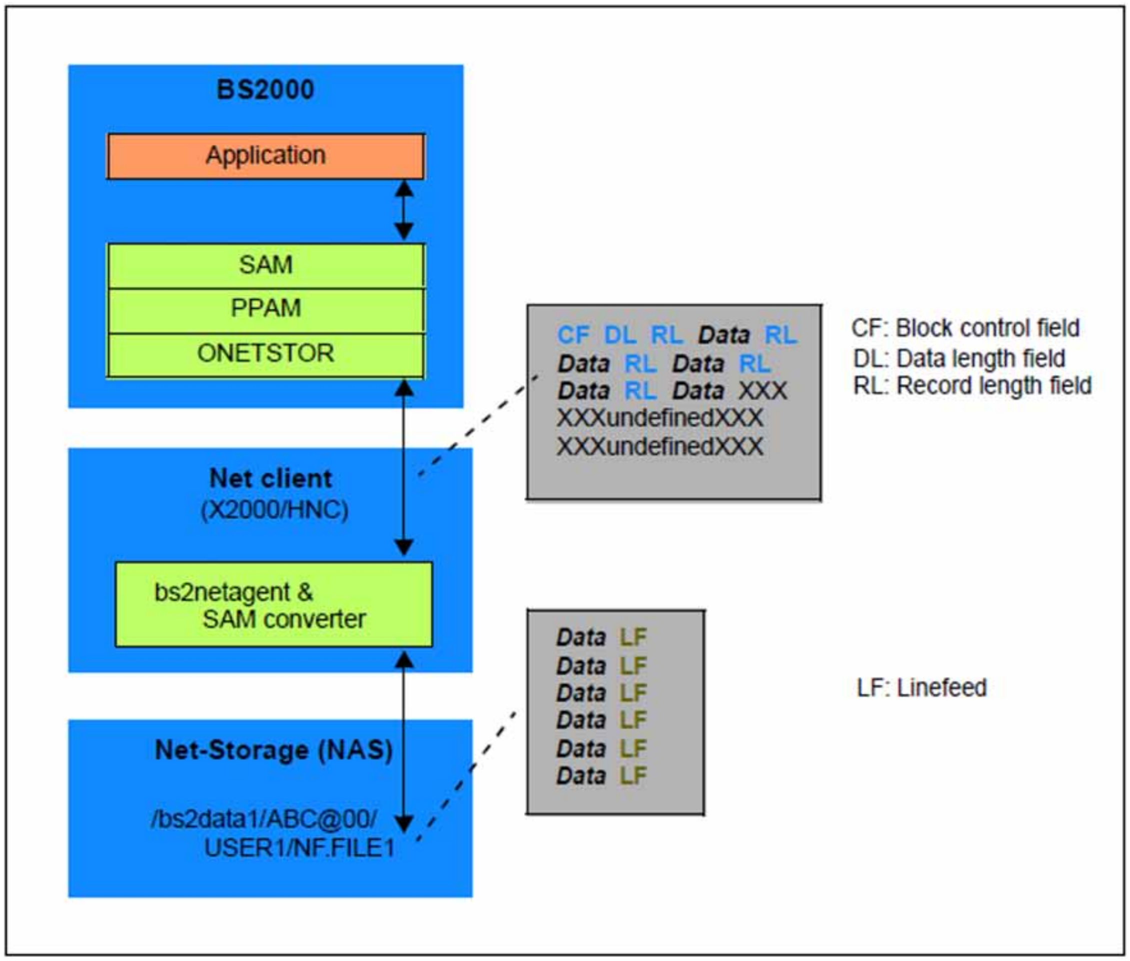


Figure 3: Converting a SAM node file

The SAM converter modifies logical SAM blocks during the writing or transferring from BS2000 to Net-Storage as follows:

- It removes the block control field (CF), the data length field (DL) and the record length fields (RL).
- Depending on the specification for the NET-CODED-CHAR-SET file attribute, a code conversion is performed for the data (see section “Character set for node files”).
- After each record, a line feed (LF) matching the coded character set (CCS) is inserted.

In the other direction, that is when reading the data, the SAM converter builds logical SAM data blocks as if the SAM access method would read the data from a BS2000 disk:

- At the beginning of each logical block, it inserts the block control field and the data length field.
- It adds record length fields to the records from the node file.

At every OPEN, the SAM converter analyzes the SAM node file and creates a node file positioning list, which states which virtual logical BS2000 block numbers correspond to which real byte positions within the file on the Net-Storage. While doing so, it determines the file size in PAM pages (LPP) and reports it back to BS2000. By referencing this list, the SAM converter can read any logical blocks in the file, prepare them and send them to BS2000 (e.g. display of a specific record for SHOW-FILE). This list is also a prerequisite for an OPEN EXTEND or REVERSE.

The positioning list exists temporarily while the file is opened, until the CLOSE. For very large files, it may take some time to build the list and require larger main memory resources of the net client. If extremely large files and many files at once are processed, it may come to resource shortages on the net client, which are answered with the corresponding return codes.

Information on using RELSE

Calling RELSE closes the current logical block and starts a new logical block. The block number of the retrieval address is increased by 1 and the record number reset to 0. At the same time, another entry is added to the positioning list of the net client, so that, as in SAM processing on public space, this position can later be found again and recognized as the start of a logical block (SETL).

However, in the case of SAM node files, the data are not written as blocks but as one continuous sequence without gaps. As soon as at least one RELSE is called during the processing of a SAM node file, the saved retrieval addresses become useless after the file is closed and opened again, because they may no longer match the node file positioning list.

3.3.4.3 Character set for node files

For BS2000 files, the CODED-CHARACTER-SET (CCS) represents the character set used in BS2000 (see section “File attributes for outputting files”). Normally, different character sets are used for the coding in SAM node files. To enable a code conversion for processing and displaying node files in BS2000, the character set used on Net-Storage is entered in the file catalog for the node files from BS2000 OSD/BC V11.0 onwards.

The NET-CODED-CHAR-SET (NETCCS) file attribute represents the actual character set that was used to store the data of a node file on the Net-Storage. When the file is displayed or processed in BS2000, EBCDIC character sets are converted according to this setting. For PAM node files, no conversion takes place.

The NETCCS of a node file is specified when the catalog entry is created with CREATE-FILE or IMPORT-NODE-FILE and can be changed with MODIFY-FILE-ATTRBUTES. As a default for CREATE-FILE and IMPORT-NODE-FILE, the CCS and NETCCS character sets are determined from the settings in the user entry. In case of CREATE-FILE, the user may also make a different specification for CCS and NETCCS.
If NET-CODED-CHAR-SET=*NO-CONV or *ISO, a target character set is automatically determined and entered in the file attributes. However, the NET-CODED-CHAR-SET may also be explicitly stated, just like the CODED-CHARACTER-SET. Reference the following table for an overview.

In the first two columns, you can see the settings in the user entry or the specification made during CREATE-FILE, from which the resulting NET-CODED-CHAR-SET in the catalog entry of the node file is determined (see column on the right):

User entry CCS 1)	**User entry NETCSS 1)**	**Resulting NETCSS in the catalog entry of the node files**
EDF03IRV/*NONE	*ISO	ISO88591; during code conversion, EDF041 is assumed for CCS
EDF03DRV	*ISO	ISO88591; during code conversion, EDF04DRV is assumed for CCS
EDF04DRV	*ISO	ISO88591
EDF04x	*ISO	ISO8859x with x=1,2,..F
ISO8859x	*ISO or *NO-CONV	ISOx
UTFx	*ISO or *NO-CONV	UTFx
<name_a 1..8>	<name_b 1..8>	<name_b 1..8>
<name_a 1..8>	*NO-CONV	<name_a 1..8>
1) User entry (SYSSRPM) or specification in CREATE-FILE / MODIFY-FILE-ATTRIBUTES		

Notes:

- The user’s specifications are not validated. Only at OPEN is XHCS called to request the required code table. If it is not available there, the OPEN is rejected.
- In the first place, changing the coded character sets (CCS and/or NETCCS) only changes the corresponding file attributes. Only when the file is processed (read and write), the code tables are used to transfer to/from the Net-Storage and simultaneously convert the data for the application.

- The required character sets have to be created before the file is being processed. The settings for CCS and NETCCS should not be changed after the data have been written on Net-Storage, as it is very likely that further converting the characters and processing them with the now changed code tables will lead to inconsistencies.
- All SAM node files and PAM node files created with BS2000 OSD/BC V11.0 or later have a defined NETCCS. PAM node files created with an older version than V11.0 are treated as if they were created with *NO-CONVERSION.
- The NETCCS is only relevant for processing SAM node files.
- Irrespective of the setting for the file attribute, a line feed control character compatible with the character set is inserted between every converted sentence while the node file is being written. This is for example x'0A' (for ISO88591, ISO646, UTF-8), x'000A' (for UTF-16) or x'15' (for EBCDIC).
- Even if the user has specified a 7-bit character set EDF03IRV or EDF03DRV as CCS, an eight-bit character set is used for the coded character set when the characters are being converted: EDF041 is assumed for EDF03IRV and EDF04DRV for EDF03DRV.
- If an ISO or UTF character set is specified for the CCS, no conversion will take place.
- A conversion from a 7- or 8-bit character code to a multibyte character set (e.g. UTF) is not possible.

3.3.5 General conditions

The following general conditions must be observed when working with Net-Storage:

- A file on Net-Storage consists of just one extent. Specification of a volume list is accepted, but only the first volume is used.
- Storage space assignment can be specified in the SPACE operand of the CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands, but no storage space of this size is occupied on Net-Storage. The (maximum) storage space assignment takes place only when the file is stored in the corresponding size.
- With respect to allocation, a Net-Storage volume behaves like an NK2 disk with the minimum allocation unit of 8 KB.
- Absolute allocation is not possible for files on Net-Storage.
- Files with the following attributes **cannot** be created on a Net-Storage volume:
  - Files with a PAM key
  - Work files
  - Temporary files
  - File generation groups

- The following applies to node files:
  - PAM node files are supported in BS2000 OSD/BC V10.0 and higher, SAM node files in BS2000 OSD/BC V11.0 and higher. Further requirements such as hardware dependencies are described in the current Release Notices.
  - For BS2000, only the following files can become node files:
    - PAM files without a PAM key (BLK-CONTROL=NO)
    - SAM files without a PAM key (BLK-CONTROL=DATA) and with variable record length (REC-FORM=V)
  - Node files are structured and end on a byte boundary.
  - Node files must be imported to be processed in BS2000.
  - Node files have the following access rights in UFS for the user concerned:

    ```
    rw- --- ---
    ```

    These are the minimum access rights required for BS2000 for the user concerned. A user of the open system can change these access rights. A BS2000 user cannot do this.

    If the ownership or access rights (chown, chmod) are changed from the open systems side, BS2000 may possibly no longer be able to access the files.
  - From the BS2000 side node files can only be created and managed as normal files. For example, links cannot be imported into BS2000 as special aspects of processing UNIX links (hardware and software links) cannot be mapped in BS2000.
  - If node files are imported as SAM file, it is important to note the following:
    - At first, the file size is adopted from the file system into the catalog entry. From the BS2000 viewpoint, this value is a bit too small, because so far it ignores the SAM structure (block control fields and length fields plus unused bytes at the end of a block). The correct value is only determined when the file is opened and then updated in the catalog.
    - For the file, the maximum block size of (STD,16) is entered in the catalog. The maximum possible record length is calculated by subtracting the length of the block control field and the record length field from this value. This way, BS2000 can read records of up to 32768-20=32748 bytes from the node file. If the blocking factor is lower, the maximum record length that can be processed is appropriately smaller. Because of this, it is recommended to always use the maximum block size for SAM node files.
  - When copying (COPY-FILE) from node files, it is important to note the following:
    - When copying into a node file, the node file must have a corresponding catalog entry. If necessary, a catalog entry must be created using the CREATE-FILE command:

      ```
      /CREATE-FILE <node-file>,SUPPORT=*PUBLIC-DISK(
                   STORAGE-TYPE=*NET-STORAGE(FILE-TYPE=*NODE))
      ```
    - When a SAM is being copied into a node file, the user can specify the desired NETCSS in the CREATE-FILE command when the node file is created. The CSS is adopted from the source file (*NONE is equivalent to EDF03IRV).
    - If a SAM node file is copied into a SAM file, the NETCSS is not transferred.
    - If a SAM node file is copied into a SAM node file, the NETCCS is adopted from the source file.

- The administrator of the file server can grant access to public files on Net-Storage for applications which run on different operating systems. The following must be borne in mind here:
  - Applications of foreign (non-BS2000) operating systems may only access BS2000 files on Net-Storage in read mode. They may access node files in write mode, too.
  - Users from an open system obtain access to node files by means of their user and group numbers (uid, gid). To enable joint access to node files, the user and group numbers of the BS2000 user and of the user in the open system must be coordinated. Under NFSv4 it is necessary to use a directory service (openLDAP or Active Directory) to do this.

  i Data access control in BS2000 (e.g. access only by the file owner, password protection) is effective only if access takes place from BS2000. This has no effect for accesses from open systems.

3.4 Magnetic tapes and magnetic tape cartridges

Magnetic tapes and magnetic tape cartridges are private volumes and must be requested by means of a PREMOUNT or MOUNT message. In contrast to disks (both public and private), a tape or a cartridge cannot be processed by more than one job at a time.

3.4.1 Magnetic tapes

Various types of magnetic tapes differ in the recording density with which the data is written (measured in bits per inch, “bpi” for short). The tape type on which a file is recorded is defined via the DEVICE operand of the FILE macro or by means of the DEVICE-TYPE operand in the CREATE-FILE and IMPORT-FILE commands.

3.4.2 Magnetic tape cartridges

Data is recorded on magnetic tape cartridges (MTCs) in accordance with the DIN standard 66229.A; the preceding standard is DIN 66029, which applies to magnetic tape devices.

The data transmission rate from and to magnetic tape cartridge is increased greatly by buffering the data in the controller of the MTC device. However, the user can deactivate this buffering for writing.

For the most part, the processing of files on MTC is the same as the processing of files on magnetic tapes. Program incompatibilities will result only when processing in buffered mode if an unrecoverable error occurs during writing and data is to be restored from the buffer by means of a program routine, or when writing checkpoints. If unbuffered mode is used, these incompatibilities will not occur, but processing will be considerably slower.

Further information about using MTC can be found in section “Use of MTC systems”.

3.5 Importing volumes from other computer centers

Private disks and tapes can be exchanged between BS2000 computer centers. The user can transfer a file to another system only if the user ID of the owner is recorded in this other system. The file is then cataloged under the owner's user ID. For files on private disks, DMS takes the file attributes from the F1 label of the disk, not from a FILE macro or IMPORT-FILE command or from an existing catalog entry.

Transfer to another system can be performed as follows:

- If several files on private disks are to be transferred, the IMPORT macro or IMPORT-FILE command, which can process partially qualified file names, can be used.
- If only a single file is to be transferred, a FILE macro with the following operands can be used:

  ```
  pfadname
  STATE=FOREIGN
  VOLUME=vsn
  DEVICE=gerät
  ```

  Single files can also be transferred with the IMPORT-FILE command. Disk files can be processed by means a FILE or an IMPORT macro, while tapes files can be processed only by means of a FILE macro. Both disk files and tape files can be processed with the IMPORT-FILE command.

3.6 Summary

The preceding sections dealt with the concept of volumes in BS2000. The following table provides an overview of the characteristics of the various types of volumes..

Volume characteristics	**Public volumes (pubsets)**		**Private volumes**	
	Disk	**Net-Storage** $^{1)}$	**Disk**	**Tape**
Number of jobs which may use the volume concurrently	Any number	Any number	Any number (exception: exclusive reservation)	No more than 1
File protection	Full file protection by DMS	Full file protection by DMS in BS2000$^{2)}$	Full file protection by DMS	Full file protection by DMS
Permissible processing method	DMS access methods; I/O macros for system files (RDATA etc.)	DMS access methods; I/O macros for system files (RDATA etc.)	DMS access methods; I/O macros for system files (RDATA etc.)	tape access methods of DMS: SAM, UPAM and BTAM
Labels			Standard labels	Standard and nonstandard labels
Volume serial number (VSN)	6 alphanumeric characters; see "Public volumes (pubsets)"	Standard name derived from the pubset (6 characters, alphanumeric); userdefined name like private disk	6 characters, alphanumeric; does not begin with "PUB" and does not contain a period	6 characters, alphanumeric; does not begin with "PUB" and does not contain a period
Volume request	Pubsets do not need to be requested by the user	Net-Storage volume does not need to be requested	Disks must be requested by the user	Tapes must be requested by the user
Dynamic storage space management	Yes	Yes	Yes	No
Volume reservation	Not possible	Not possible	Mandatory, otherwise the job may be aborted if the volume is not available	

Table 4: Volume characteristics

1) Net-Storage expands the storage space on public volumes, but is itself not counted as a public volume

2) The BS2000 protection attributes cannot be implemented on open systems which have access to Net-Storage files. By default, node files are assigned to the relevant user by means of user and group numbers (UID, GID), and read and write authorization is only entered for this user. However, the UNIX rights can be modified from the UNIX side (for details, see section “General conditions”)

4 Files in BS2000

The data to be processed by programs is generally collected in files. In BS2000, however, the programs themselves, both the source programs and the load modules, are regarded as files. In order to process their data, users must be able to access their files at any time. The Data Management System (DMS) provides them with support for all functions relating to file access, file management and file maintenance.

The information (data) to be processed by a system is collected into various units. The smallest addressable unit is the byte and the smallest unit which can be transferred is the physical block which, on disks, has a fixed length and is called a PAM page (section “Blockformats for disk files”).

Logical units are: character, field, record, file and logical block (= data block). The character is the smallest logical unit and corresponds to the byte. A field may contain one or more characters. Fields form the record, which is the basic element of a file. The file, finally, is the sum of the records which it contains.

The system (input/output devices, etc.) knows nothing about the record, which is a logical processing unit. The connecting link between the data storage devices and the transfer/processing devices is the data block. This is the transfer unit and may, for example, consist of one or more standard blocks. Standard blocks (PAM pages) have a specific format and length and can be used for disk and tape files. Nonstandard blocks may be used only for tape files (see section “Block formats for disk files” and section “Blockformats for tape files”).

The data block may contain one or more records which may extend over one or more PAM pages.

4.1 File types

A basic distinction must be made between three file types: permanent files, work files and temporary files. Permanent files and work files are not specific to the job which creates them. Temporary files are specific to the job in which they are created. They are automatically deleted during LOGOFF handling.

Spoolout files are also specific to the job in which they are created. They are not, however, managed by DMS and are described in detail in the "SPOOL" manual [4 (Related publications)].

4.1.1 Permanent files

Permanent files are created with the aid of the access methods SAM, ISAM, UPAM, FASTPAM, DIV and BTAM. They reside on external volumes Work files are retained after termination of the job in which they are created and cataloged until the user explicitly deletes them by means of the DELETE-FILE command. Their file names and attributes are defined by the user. By specifying certain file attributes in the catalog entry, the user may permit other users to access these files.

Remote file access (RFA) is also possible (see the “RFA” manual [6 (Related publications)]). Permanent files can be stored on public volumes, private volumes (private disks or tapes) or on Net-Storage. If they are on public volumes (pubsets), they are subject to public-space control by the system.

Permanent files are maintained in the catalog under a “path name” which consists of the catalog ID (“catid”), the user ID (“userid”) and the fully qualified file name assigned by the user (see section “Path name”).

4.1.2 Work files

Work files reside in a specific volume set of an SM pubset reserved for this type of file and identified as a “work volume set” by the work attribute. Work files are generated either by specifying the file attribute WORK-FILE=*YES (e.g. in the CREATE-FILE command) or by means of physical allocation (see "Physical allocation"), e.g. using the command

```
CREATE-FILE ...,VOLUME=<vsn of a disk from work volume set>
               ,DEVICE-TYP=<associated device type>).
```

No work file can be created unless a work volume set exists in the SM pubset. Work files can be processed with the access method SAM, ISAM, UPAM, FASTPAM or DIV.

Work files are retained after termination of the job in which they are created and cataloged until the user explicitly deletes them by means of the DELETE-FILE command. Their file names and attributes are defined by the user. By specifying certain file attributes in the catalog entry, the user may permit other users to access these files.

Remote file access (RFA) is also possible (see the “RFA” manual [6 (Related publications)]). Work files are not subject to public-space control by the system.

The usage model for work files is characterized by the fact that they are always created in work volume sets which systems support makes temporarily available to the users. Systems support and users must agree upon the duration of the work volume set. After a defined period has elapsed, system support has the right to delete all the files on the work volume set and to export the work volume set.

The path names of work files are no different from those of permanent files.

4.1.3 Temporary files

Temporary files are job-specific. They cannot be accessed by other jobs running under the same user ID or by other users. On completion of the job, they are automatically deleted by LOGOFF handling if the owner has not already deleted them explicitly. If a spoolout job for a temporary file is still outstanding at LOGOFF, the file is not deleted until the job has been executed. If a job is aborted due to abnormal termination of the system, temporary files remain cataloged until systems support imports the pubset again, since no LOGOFF handling is executed in this case.

Temporary files can be processed with the access method ISAM, SAM, UPAM, FASTPAM, DIV or BTAM. By default, they are created in the user area of the system and DMS treats them, wherever permissible, just like permanent files of the same type. They can be created on tapes, but not on private disks.

Nonprivileged users can only create temporary files on the default pubset for their user IDs.

The user creates a temporary file by prefixing a special character to the file name. This special character (# or @) is defined by the TEMPFILE system parameter at system startup and can be queried using the SHOW-SYSTEM-PARAMETER command or the NSIOPT macro (see the “Executive Macros” manual [2 (Related publications)]). The user-defined name for a temporary file is then, for example, “#CUSTOMERLIST”. Under this name, the temporary file can be processed (opened, closed, deleted...).

Temporary files are maintained in the file catalog under an internal file name with the following format: S.<sysid>.<tsn>.filename, where

<sysid>	is the ID of the system within the network (three-digit number)
<tsn>	is the current task sequence number (TSN)

Example of the internal name of a temporary file: S.100.1QXR.TEMP.FILE.

In various macros as well as in system messages, the internal file name is output. These naming conventions ensure that identically named temporary files of concurrently active jobs of a user do not collide. If the user assigns a file name whose format matches that of a temporary file (name starting with “S.” followed by a three-digit number, a period and a four-digit combination of letters and numerals), this name is rejected by the system.

If the user explicitly creates a file whose name has all the above-mentioned attributes (S.<sysid>. <tsn>), this file is created as a temporary file. To process it, the user needs to enter the file name with the appropriate special-character prefix (see above). Once the task has terminated, this file is lost.

i The use of the “internal” file names which are rooted in programs or procedures makes it impossible to port these products to other versions of the operating system. The internal names of temporary files are not part of the user interface and may be changed at any time. For this reason they should not be used in programs/procedures.

If temporary files (with a prefix in their names) are created on tape, neither the prefix nor the character string “S.<sysid>.<tsn>” is included in the HDR1 or HDR3 labels (only the file name without a prefix). The file is flagged as temporary only in the catalog. If such a file is opened with OPEN INPUT or OPEN INOUT (see chapter “OPEN processing”), DMS does not check this character string as part of HDR1/HDR3 processing.

Temporary file generation groups are not permitted.

Recataloging a work file as a temporary file or vice versa is not possible.

In an SM pubset, renaming a temporary file as a permanent file or vice versa is rejected if changing the file attributes implies reallocation to another volume set (S0 migration).

EAM files

The SYSEAM area is a domain on public volumes which is maintained under the user ID TSOS. Files created here by the user or by the system are temporary files and can be processed with the access method EAM (see chapter “EAM – Event Access Method” (EAM - Event Access Method)). These EAM files are subject to different conventions than the other temporary files, e.g. the file naming: the first time an EAM file is opened, the system assigns a twobyte binary number (1 <= name <= 14000) as the file name and the user can use this number to address the file. In contrast to other temporary files, EAM files are not kept in the user catalog.

The system-created object module file is a special EAM file. It can be referred to as the * file or *OMF file. (Examples: specifying the ERASE * macro or the DELETE-SYSTEM-FILE *OMF command deletes the object module file.) The EAM macro also offers access to the object module file.

Logical system files

BS2000 uses system files for statement and data input to the operating system and for output of data and messages by the operating system. These input and output areas predefined by the system are known as logical system files; they are job-specific and therefore temporary files.

Just like other temporary files, the logical system files can be used without affecting the user's public-space allocation.

The total set of system files available to a job is called the SYSFILE environment. This is significant, for example, when writing checkpoints (see the WRCPT macro in the “Executive Macros” manual [2 (Related publications)]).

System files may be used as the input or output paths of the system, for example by means of the WRLST and RDATA macros (see the “Executive Macros” manual [2 (Related publications)]).

The Data Management System of BS2000 supports only the processing of permanent and temporary user files and of EAM files. Refer to the “Commands” [3 (Related publications)] and “Executive Macros” [2 (Related publications)] manuals for details of processing logical system files.

4.2 File names

Each file cataloged in BS2000 is uniquely identified by its path name. This consists of a file name specified by the user and a prefix assigned by the system; the prefix is equivalent to the access path in the pubset.

A distinction must be made between the following terms: path name, fully qualified file name, partially qualified file name and internal file name. The term “tempfile name” may also be used for the name of a temporary file.

4.2.1 Path name

The file name and the user ID are entered in the file catalog (separately). The path name is the name for a BS2000 file, including the catalog ID and the user ID. It uniquely identifies a permanent or temporary file, a file generation, or a file generation group.

Format

pathname = :catid:$userid.filename

The maximum total length is 54 characters.

catid

Catalog ID; identifier of the pubset in which the file is cataloged. Length: 1 to 4 characters.

Default catalog ID: the catalog ID assigned to the user ID in the user catalog, i.e. the ID of the pubset on which the user's files are created by default.

A catalog ID must always be enclosed between colons (*:catid:*).

userid

User ID. Length: max. 8 characters.

Default user ID: normally the user ID of the current job, i.e. the one specified in the SET-LOGON-PARAMETERS command; For “secondary read”, a default user ID at the system level can be used: all files which are to be accessible to all users are cataloged under this user ID (see section “System default user ID” and "Access via the system default user ID").

When referring to a file, the user ID must always be preceded by a “$” character and terminated with a period ($*userid.*).

filename

A fully qualified file name defined by the user; its length is dependent on the typeof file:

- for permanent files max. 41 characters
- for temporary user files max. 31 characters (including the prefix)
- for file generation groups max. 34 characters

If the catalog ID consists of more than one character, the maximum possible length is reduced if the total length of the catalog ID (including colons) and that of the user ID (including the “$” character and period) exceeds 13 characters (`:catid:$userid.`).

4.2.2 Fully qualified file name

A fully qualified file name is the file name defined by the user combined with the catalog ID and the user ID. A fully qualified file name may be split into partial names with the aid of periods.

Format

Fully qualified file name = name$_1$[.name$_2$[. ...]][()]{ absgen / version }

The total length is dependent on the type of file:

- for permanent files max. 41 characters
- for temporary files max. 31 characters
- for file generation groups max. 34 characters
- for file generations max. length of the name for a file generation group (see above) +7 characters for the absolute generation number

name$_1$

Partial name 1 (see “Rules for the formation of file names”)

name$_2$

Partial name 2 (see “Rules for the formation of file names”)

absgen

Absolute generation number;
valid range: 1 <= absgen <= 9999;
the generationnumber identifies a file generation within a file generation group. File generations may also be referenced via relative generation numbers, but these are not included in the catalog entry (see section “Generation numbers”).

The absolute generation number must always be specified preceded by an asterisk and enclosed in parentheses (*(*absgen)*).

version

Version ID for tape files; a simple name consisting of one or more characters(letters, digits or special characters).

Generally, cataloged tape files must have unique file names. The freely selectable version ID permits the user to catalog different files which would otherwise have identical file or path names.

The version ID is entered in the file catalog, but not in the file labels. It is thus merely an aid which uniquely identifies the catalog entry and permits implicit association of a file and a volume.

The version must always be enclosed in parentheses (*(version)*). DMS recognizes the version ID by the opening parenthesis “(“ and the following character, which must not be a plus, minus or asterisk sign (+ or - or * in parenthesis identifies a file generation). DMS ignores all following characters when it writes the file name into the tape labels. When tape labels are read, the file names are compared only up to the opening parenthesis, and all following characters are ignored.

Rules for the formation of file names

The user may use the following characters:

- all letters (A ... Z)
- all digits (0 ... 9)
- the special characters #, @, $, the hyphen (-) and the period (.).

A fully qualified file name must contain at least one letter; this may be anywhere within the character string.

The period divides a fully qualified file name into partial names (see “$name_1.name_2$”, above). It cannot, therefore, be the first or last character of a partial name.

The hyphen must not be the first or last character of a partial name.

The $ character is used to identify the user ID and must therefore not be used as the first character of a file name.

The characters “@” and “#” are used as the prefix for identifying temporary files. They should not, therefore, be used as the first character of a file name, even if the TEMPFILE function (i.e. temporary files are permitted) is not active in the system.

When assigning file names and group names, it is advisable to restrict their length to the maximum value of four positions for the catalog ID and eight positions for the user ID. This will ensure that the file set can be transferred to another pubset or another user ID without problems (e.g. when the user switches from pubset “A” to pubset “AB01”).

Example

```
/show-file-attributes # ———————————————————————————————————————————————  (1)
         3 :2OSG:$USERXYZ.S.152.500H.PROTOKOLL
         3 :2OSG:$USERXYZ.S.152.500H.VOLL.QUALIF.TEMPFILE.NAME
:2OSG: PUBLIC:      2 FILES RES=         6 FRE=         6 REL=        6 PAGES
/show-file-attributes  generations=yes ————————————————————————————————  (2)
         0 :G:$USERXYZ.ABCDEFGHIJKLMNOPQRSTUVWXYZ.1234567890 ——————————  (3)
         3 :G:$USERXYZ.ABDEF.134.$-@#
         3 :G:$USERXYZ.DATEINAME
         0 :G:$USERXYZ.DAT.GENGROUP (FGG) —————————————————————————————  (4)
         0 :G:$USERXYZ.DAT.GENGROUP(*0001)
         0 :G:$USERXYZ.DAT.GENGROUP(*0002)
         3 :G:$USERXYZ.DO.CMH.ASS —————————————————————————————————————  (5)
        12 :G:$USERXYZ.DVS.TAB2-2
        30 :G:$USERXYZ.PROTO.DATEINAMEN
         3 :G:$USERXYZ.VOLL.QUALIF.DATEI.NAME
         3 :G:$USERXYZ.1234567890X0987654321
:2OSG: PUBLIC:     11 FILES RES=        57 FRE=        47 REL=       36 PAGES
```

(1) The SHOW-FILE-ATTRIBUTES command lists the temporary user files (prefix: #) created by the current task (TSN=500H) on the default pubset (2OSG) under the calling user ID ($USERXYZ).

(2) The SHOW-FILE-ATTRIBUTES command lists the permanent files, file generation groups and file generations cataloged on the default pubset (2OSG) under the calling user ID ($USERXYZ).

(3) 3 examples of the permissible character set for the formation of file names.

(4) Example of a file generation group with 2 file generations.

(5) 5 examples of the permissible character set for the formation of file names.

4.2.3 Partially qualified file name

A file name is said to be partially qualified if it ends with a period. By forming a file name from several partial names, the user can map assignments or functions in the file name. A partially qualified file name can be used to address several files simultaneously. This means that partially qualified file names cannot be used when creating a file, but only when accessing existing files (deleting, importing, obtaining information, etc.).

Format

Partially qualified file name = $name_1$.[$name_2$.]...

$name_1$= partialname-1

$name_2$ =partialname-2

The last character of a partial name is always a period.

A partially qualified file name always refers to one or more existing/cataloged files. The rules for formation of $name_1$, $name_2$ are thus the same as explained in section “Fullyqualified file name”.

Example

Assembly and runtime listings for source programs written in various programming languages (e.g. Assembler, FORTRAN) are generated during the program testing phase, and finally load modules are generated. The file names could be structured as follows:

Source programs	Assembly listings	Runtime listings	Load modules
`Q.ASS.PROG1.V1`	`ASSLST.PROG1.V1`	`PROTO.PROG1.V1.1`	`L.PROG1.V3`
`Q.ASS.PROG1.V2`	`ASSLST.PROG1.V2`	`PROTO.PROG1.V1.2`	`L.PROG1.V2`
`Q.ASS.PROG1.V2A`			`L.PROG3.V3`
		`PROTO.PROG1.V2.1`	
`Q.ASS.PROG1.V3`			
`Q.ASS.PROG2.V1`			
`Q.FOR.PROG4.V1`			
`Q.FOR.PROG4.V2`			
`Q.FOR.PROG5.V3`			

With the `ERASE ASSLST.` macro or the `/DELETE-FILE ASSLST.` command, all files with $name_1$ = ASSLST generated during compilation can be deleted. With `ERASE` or `/DELETE-FILE RTLST.`, all of the runtime listing files can be deleted, etc. “RTLST.PROG1.V1.” lists all runtime listings for version 1 of PROG1. The `FSTAT S.` macro or the `/SHOW-FILE-ATTRIBUTES S.` command lists all source programs with the identifier S, while `FSTAT S.ASS.` or `/SHOW-FILE-ATTRIBUTES S. ASS.` would list all programs written in Assembler, etc.

4.2.4 System default user ID

At system startup, systems support uses the DEFLUID system parameter to define a default user ID which is valid throughout the system (usually $TSOS). The nonprivileged software products available to all users (e.g. utility routines, editors, compilers, etc.) are cataloged under this user ID.

A user may access these files without specifying a user ID by placing “$.” or the $ character before the file name. If the file name is divided into partial names by periods, the string “$.” must be placed in front of the name. If no such division exists, it is sufficient to place the $ character in front of it.

This system default user ID is also significant for secondary read access (see section“Access via the system default user ID”).

Example

```
Editor: $.EDT, $EDT
Compiler: $ASSEMB
Utility routines: $SORT, $PERCON, $DPAGE, $TSOSLNK, $LMS, $.ARCHIVE
```

4.3 Files larger than 32 GB

Because of the continual growth of disk storage capacity and of data that needs to be available online, the disk and file sizes have been increased from the approx. 32 GB previously supported in BS2000. The limits are as follows:

- The maximum capacity of a pubset or volume set is about 4 TB.
- The maximum capacity of a single disk is about 2 TB.
- The maximum file size is about 4 TB (which is the maximum size of a pubset or volume set minus SVL, F5 label and system files).

This means that within the operating system, 4-byte block numbers and 4-byte counters must be used systematically for file sizes and disk sizes.

Converting 3-byte fields to 4-byte fields affects all components, products and applications that work directly or indirectly with these fields. The TPR interfaces have been modified accordingly, but not all of the TU user interfaces can compatibly support these large files.

Preconditions

There are two pubset types for large objects (large files and volumes):

- LARGE-OBJECTS pubsets without large files:

  This pubset type allows large volumes, but limits the file size allowed to 32 GB. This means that the pubset may contain volumes >= 32 GB, but does not necessarily currently contain such volumes.

  From the point of view of the user, these pubsets behave (almost) as conventional pubsets.
- LARGE-OBJECTS pubsets with large files

  This pubset type allows large volumes and large files, but they do not necessarily exist.

  It allows large files to be separated from programs that use interfaces that are incompatible and supports the step-by-step introduction of large volumes and – at a second step – of large files.

Large objects are supported by both SF and SM pubsets. To achieve this, the following two attributes have been introduced:

LARGE_OBJECTS	Attribute for supporting volumes >= 32 GB
LARGE_FILES_ALLOWED	Attribut steuert, ob auf dem Pubset auch große Dateien (Dateien >= 32 GB) unterstützt werden

As for pubsets, an attribute has been introduced for volumes: LARGE_VOLUME. The LARGE_VOLUME attribute characterizes a logical volume and indicates that the volume has a gross capacity >= 32 GB. Just as the LARGE_OBJECTS characteristic is for pubsets, the LARGE_VOLUME characteristic is permanent and is stored in the SVL.

3-byte and 4-byte fields

The introduction of 4-byte fields for the following data stored in the catalog entry is a key aspect in the lifting of the 32-GB limit for volume and data sizes:

- FILE-SIZE – the storage space allocated for the file
- HIGHEST-USED-PAGE – the amount of storage that currently contains data

- LHP (Logical halfpage number) and PHP (physical halfpage number) of the individual extents in the extent list, that allocate “physical” halfpages of volumes to the logical halfpages

Block numbers and block counters are visible at various BS2000 user interfaces. Although 4-byte fields have been used exclusively in all new versions of these interfaces, some old interface versions may still use 3-byte fields. If files >= 32 GB exist, you may experience compatibility problems, as is also the case when "only" volumes >= 32 GB exist.

The introduction of 4-byte LHPs and 4-byte PHPs means that a new (additional) format also exists for the extent list. In order to achieve as much downward compatibility as possible, the current BS2000 versions support both formats of the extent list:

- In general, the “old” format with 3-byte block numbers will be used.
- Only in the case of large files or of files that can only be addressed using PHPs larger than X'FFFFFF' will the new format with 4-byte block numbers be used.

Extent lists therefore contain either extents with 3-byte block numbers or extents with 4-byte block numbers.

Restrictions for large files

The following restrictions for large files must be taken into account when working with nonprivileged applications:

- Support for files >= 32 GB is only possible on pubsets with LARGE_OBJECTS=TRUE, LARGE_FILES_ALLOWED=TRUE.
- Support for files >= 32 GB is only possible for non-PAMKEY files: Files with BLKCTRL=PAMKEY are not supported since the logical page number is stored as a 3-byte field in the system section of the PAMKEY.
- EAM files >= 32 GB are not possible, because the SYSEAM container file cannot be a large file.

Summary of the DMS pubset management interfaces affected by 32 GB objects

Interface	Change
Nonprivileged commands	
ADD-FILE-LINK	New operand EXCEED-32GB for specifying the permitted file size (larger or smaller than 32 GB)
SHOW-FILE-LINK	Outputs the “large file” file attribute
SHOW-FILE-ATTRIBUTES	Extended output for various output fields
Macros	
FCB	New operand LARGE_FILE for large files
FILE	New operand EXC32GB for large files
FSTAT	Effort involved in check and conversion if VERSION=0/1
OPEN	Take account of semantic problems

RDTFT	Outputs the “large file” file attribute
DIV	New operand LARGE_FILE for large files; Extended range of values for the BLOCK and SPAN operands
FPAMACC	Extended range of values for BLOCK operand
FPAMSRV	New operand LARGE_FILE for large files

Table 5: Summary of the DMS pubset management interfaces affected by 32 GB objects

Executability of programs in configurations with files larger than 32 GB

It cannot be assumed that all programs have been prepared for accessing large objects, i.e. that they can address 4-byte block numbers and block counters, and it must be borne in mind that the interfaces available to user applications only relate to accessing and processing files and their metadata. The following description is therefore confined to large files. Program behavior can thus be classified as follows:

Class A:	A program is able to process large files without restrictions. This behavior is defined as LARGE_FILES-capable.
Class B:	A program has not been prepared for processing large files and/or their metadata. It is, however, able to perform defined rejections of corresponding access attempts that it regards as illegal or, alternatively, there is no access to files or their metadata within the program. This behavior is defined as LARGE_FILES-compatible.
Class C:	A program has not been prepared for processing large files and cannot perform a defined rejection of corresponding access attempts. This behavior is defined as LARGE_FILES-incompatible.

In configurations that contain large files, programs that are compatible with or capable of LARGE_FILES are required. LARGE_FILES compatibility is to be regarded as the norm. Growth over 32 GB will initially be limited to a relatively small number of files; Only the programs that access these require LARGE_FILES capability.

For a presentation of all the considerations associated with the use of large volumes and files, refer to the “Files and Volumes larger than 32 GB” manual [18 (Related publications)].

4.4 File attributes for outputting files

The file attribute CODED-CHARACTER-SET identifies the file’s character set via the CCS name. This character set defines the presentation of the file content on output media (for example screen, printer) for characters and the collating sequence.

The file attribute CODED-CHARACTER-SET is, for example, evaluated by the COMPARE-DISK-FILES, MAIL-FILE, PRINT-DOCUMENT and SHOW-FILE commands and by the software products EDT and openFT.

A default user character set can be entered for a user ID. When the file attribute CODED-CHARACTER-SET is defined, it can be specified that the default user character set from the user entry should be accepted.

A default system character set can be entered in the system parameter HOSTCODE. When a default user character set is defined, it can be specified that the default system character set should be accepted.

Details are provided in the "XHCS" manual [21 (Related publications)].

In addition, there is the NET-CODED-CHAR-SET file attribute for node files, which identifies the character set with which the file is saved on Net-Storage. This character set is necessary to process SAM node files (see section “Character set for node files”).

The standard for node file character sets is defined in the NETCODE system parameter. This value can be transferred to the user entry as NET-CODED-CHAR-SET (see commands ADD-USER and MODIFY-USER-ATTRIBUTES).

4.5 Comparing disk files

The COMPARE-DISK-FILES command and the COMPFIL macro compare two disk files block by block (UPAM) or record by record (SAM, ISAM) and inform the user of the result of this comparison.

Temporary files or work files can also be compared. The files may reside on public volumes, Net-Storage or private disks.

Files with the following properties **cannot** be compared:

- empty files
- opened files
- locked files (e.g. SECURE lock)
- REPAIR-NEEDED identifier set
- NO-DMS-ACCESS identifier set

Entire file generation groupd cannot be compared, although individual file generations can.

PLAM libraries can be compared block by block. The members they contain cannot be compared.

5 File and catalog management

5.1 File catalog

Files in BS2000 are usually cataloged under the user ID of the job in which they are created. (Exceptions: Co-owners, see "Defining co-ownership (co-owners)", and users possessing the TSOS privilege can catalog files under user IDs other than their own.) The catalog entry describes file attributes and volumes; only cataloged files are available to the system (exception: foreign files on tape).

5.1.1 Creates a catalog entry

The creation of a catalog entry is not dependent on the creation of a file (with space allocation). Catalog entries can be created by means of macros and commands.

Macros

Catalog entries can be generated with the CATAL and FILE macros. They have the following functions:

- CATAL creates an entry in the user catalog, taking into account the protection attributes assigned by the user, but it does not reserve storage space.
- FILE creates a catalog entry and allocates storage space to the file (exception: files on Net-Storage volumes), but uses only the default values for the file protection attributes. These default values can then be modified by means of the CATAL macro. Other file attributes in the catalog entry (the access method used for file creation, block length, record length, etc.) are recorded only when the file is opened for the first time. Values indicating the logical file size (= highest page number occupied) etc. are recorded when the file is closed.

  To create a node file, NFTYPE=NODE-FILE must be specified explicitly in the FILE macro. If it is not, the Net-Storage file is created as a BS2000 file.

Commands

Catalog entries can be created with:

- the CREATE-FILE command for files
- the CREATE-FILE-GENERATION command for file generations
- the CREATE-FILE-GROUP command for file generation groups

These commands create an entry in the file catalog, taking into account the protection attributes assigned by the caller and any storage space reservation specified for CREATE-FILE.

To create a node file, FILE-TYPE=*NODE-FILE must be specified explicitly in CREATE-FILE. If it is not, a Net-Storage file is created as a BS2000 file.

Some of the file attributes (the access method used for file creation, block and record length, etc.) are recorded only when the file is opened for the first time. Values indicating the logical file size etc. are recorded when the file is closed. The user can use the utility routine SPCCNTRL (SPACE-CONTROL) and the SHOW-FILE-ATTRIBUTES command to display the catalog entries.

Files and file generations on private volumes and files on Net-Storage volumes whose catalog entries have been deleted or which were created in other systems can be cataloged (= imported) using the IMPORT or FILE macro, STATE=FOREIGN operand, or the IMPORT-FILE command. In the case of disk files, only the first volume containing the file need be mounted at this time, since the catalog entry is generated from the F1 label of this volume or the catalog entry of a Net-Storage volume. Node files are cataloged with the IMPNFIL macro or with the IMPORT-NODE-FILE command. For tape files, the volume sequence must be described in the macro or command (see section “Non-cataloged tapefiles”).

A file can only be imported if there is an entry for its user ID in the user catalog. If the user ID of the file is not the same as that of the current job, the file must be shareable (except in the event of co-ownership, see "Defining co-ownership (co-owners)"). If a file is imported from another system or from another user ID, the old catalog entries for this file should be erased.

5.1.2 Update the catalog entry.

Each time file processing occurs the fields of the catalog entry which have changed because of the processing are updated. Such fields are, for example, the ACC-DATE (date of last access) and ACC-COUNT (access counter) fields for input files. Such fields are, e.g., the fields ACC-DATE (“date of last access”) and ACC-COUNT (access counter) for input files.For output files, the file and processing attributes are written into the catalog entry, and the volume list, the extent list, and the file size (for disk files: HIGH-US-PA; for tape files: BLK-COUNT) are updated.

For disk files which are not locked, the storage space allocation can be changed at any time with the FILE macro (operand SPACE) or the MODIFY-FILE-ATTRIBUTES command. Occupied pages cannot be released. If a disk file is not locked, its protection attributes can be changed with the CATAL macro or the MODIFY-FILE-ATTRIBUTES command. In the case of tape files, the protection attributes are written into the labels when the file is opened for the first time and cannot be changed freely after this.

5.1.3 Displaying information from the catalog entry

With the aid of the FSTAT macro or the SHOW-FILE-ATTRIBUTES command, the user can obtain information from the catalog entries of his own files (whether permanent or temporary) at any time. (S)he can also obtain information from the catalog entries of shareable files which belong to other users.

The information output can be controlled using the various functions of the macro or command:

- select certain parts of the catalog entry
- use selection criteria to control the information output on the basis of the attributes stored in the catalog entry
- direct output to various output media: SYSOUT, SYSLST, file, printer

The user can also use the utility routine SPCCNTRL to obtain information from catalog entries (see the “Utility Routines” manual [13 (Related publications)]).

5.1.4 Deleting a catalog entry

The catalog entry is automatically deleted when the related file is erased. However, erasing the file contents and deleting the catalog entry can be carried out separately with the various functions of the appropriate macro or command. Erasing the file contents without deleting the catalog entry is also called “logical erasure” of the file (see "Deleting a catalog entry"), while deleting the catalog entry without releasing the storage space is called “file export” (see "Deleting a catalog entry").

The following tables show overviews of the functions of the ERASE macro and the DELETE-FILE and EXPORT-FILE commands for deleting catalog entries.

ERASE macro

Operands	**Macro version number**	**Destroy data**[1]	**Deleting a catalog entry**	**belegten Speicherplatz freigeben**	**Delete logically** [2]	**Comments**
„pathname“	0 – 3		yes	yes		
SPACE-CATALOG	1 – 3		yes	yes		Default function
DESTROY	0 – 3	yes	yes	yes		Physical data destruction
DATA	0 – 3				yes	File is “logically erased”
DATA-KEEP-ATTR	3				yes	File-specific attributesare retained
SPACE	0 – 3			yes		Only for pubset files
CATALOG	0 – 3		yes			“Private” file is exported
CATALOG, VOLUME	0		(yes)	(yes)		Volume is exported
DELETE-OR-EXPORT	1 – 3		(yes)	(yes)		Effect depends on volume type
DELETE-OR-EXPORT, VOLUME	1 – 3		(yes)	(yes)		Volume is exported

[1] If the catalog entry contains DESTROY=NO.

[2] “Logical erasure” = Resetting the Last-Page Pointer

Table 6: Functions of the ERASE macro

DELETE-FILE command

Operands	Destroy data[1)]	Deleting a catalog entry	Release allocated storage space	Delete logically [2)]	Comments
FILE-NAME		yes	yes		
OPTION=ALL		yes	yes		Default function
OPTION=DESTROY-ALL	yes	yes	yes		
OPTION=DATA				yes	File is “logically erased”
OPTION=SPACE			yes		Only for pubset files
OPTION=DATA-KEEP-ATTRIBUTES				yes	File-specific attributes are retained
[1)] If the catalog entry contains DESTROY=NO. [2)] “Logical erasure” = Resetting the Last-Page Pointer					

Table 7: Functions of the DELETE-FILE command

EXPORT-FILE command

Operands	Deleting a catalog entry	Comments
FILE-NAME	yes	Datei auf Privatplatte oder Net-Storage-Volume wird exportiert

Table 8: Functions of the EXPORT-FILE command

EXPORT-NODE-FILE command

Operands	Deleting a catalog entry	Comments
FILE-NAME	yes	Node file (on Net-Storage volume) is exported

Table 9: Functions of the EXPORT-NODE-FILE command

Node files are cataloged with the IMPNFIL macro or with the IMPORT-NODE-FILE command.

Erasing a file

The default function of the ERASE macro and the DELETE-FILE command depends on the type of volume. For disk files, the default value is “release allocated space and delete catalog entry”; in addition, the user can have the released space overwritten, see section“Data protection by “data destruction” (DESTROY option)”). For tape files, the default is “delete catalog entry”, which is equivalent to exporting the volume.

Destruction of data when the file is erased can be requested in the macro or command call, or can be specified previously in the catalog entry:

- Operand DESTROY = YES in the CATAL macro
- Operand PROTECTION...DESTROY-BY-DELETE in the CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands

“Logically erasing” a file

The catalog entry and the allocated storage space are retained; the pointer to the last PAM page which was written (highest-used page) is simply reset to 0. The data still exists on the disk, but the user can no longer read it, since physical access to volumes is not permitted. The contents of the fields CRE-DATE and HIGH-US-PA in the catalog entry for a logically erased file are the same as for a file newly created by means of FILE (macro) or CREATE-FILE (command). A file may also be regarded as “logically erased” if it is (re-)opened with OPEN OUTPUT and then immediately closed again: the old file contents still exist, but the user can no longer access them. However, both in the case of OPEN OUTPUT and when erasing with the operand value DATA-KEEP-ATTR the file-specific attributes in the catalog entry are retained.

File export

The catalog entry is deleted, but the storage space and the data are retained. Files on private volumes or Net-Storage volumes can be removed from a system in this manner. They can be imported again, into the same or another system, at any time (by means of the IMPORT macro or the IMPORT-FILE command; see section “Creates a catalog entry”). Node files are imported with the IMPNF macro or with the IMPORT-NODE-FILES command.

Files are exported with the ERASE macro (DELETE-OR-EXPORT or CATALOG operand) or the EXPORT-FILE command. If the VOLUME operand is specified together with the macro operands or the command (permitted only for disk files), all files on this volume are exported.

5.2 File location

5.2.1 Requesting storage space

The user can influence storage space allocation on public disks by means of the SPACE operand of the FILE macro or the CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands.

i In the case of files on Net-Storage the SPACE operand is accepted and the FILE-SIZE is specified accordingly in the catalog entry. The storage space for the file is only occupied in the file system when the file is stored with user data of the corresponding size.

If the SPACE operand is not specified when a file is cataloged by means of the FILE macro or CREATE-FILE command, the system default values are used. If a file is cataloged by means of the CATAL macro, only the catalog entry is created, i.e. no space is allocated to the file.

The SPACE operand defines the primary and/or secondary allocations.

Storage space requests of up to 4 TB are possible. This is the maximum capacity of a pubset minus pages used for the management data of the system (TSOSCAT, user catalog, F5 label...). The theoretical maximum file size that can be represented in the catalog is
2,147,483,647 PAM pages.

Primary allocation

This storage space is immediately reserved or released for the file. The value specified in the SPACE operand is rounded up to a multiple of “k”.
(“k“ being the number of 2-Kbyte blocks per allocation unit, the smallest memory management unit; see "Requesting storage space").

- If the value in the macro is positive or is specified in the commands with the operand SPACE=*RELATIVE or SPACE=*ABSOLUTE, DMS reserves the corresponding number of PAM pages for the file.
- If the value in the macro is negative or is specified in the commands with the SPACE=*RELEASE operand, the corresponding number of free PAM pages are released, provided this is possible.

Secondary allocation

In contrast to the primary allocation, the secondary allocation has no effect when a file is created; it is only used if the reserved storage space is insufficient when creating or extending the file. The system then automatically increases the space allocation for the file by the number of PAM pages defined as the secondary allocation (SHOW-FILE-ATTRIBUTES command, output field S-ALLOC).

The value for the secondary allocation is doubled after each successful extension. Doubling the value is discontinued when the maximum value set in the system is reached. If required, the defined number of PAM pages is rounded up to a multiple of “k” when allocating space. This process may be repeated until the total storage space available to the user is exhausted.

Allocation of contiguous storage areas (extents)

For both primary and secondary allocation of storage space, DMS attempts to reserve contiguous storage areas and to avoid splitting a file over several disks. The resulting file sections are called extents. The FSTAT macro and the SHOW-FILE-ATTRIBUTES
command show the user how many extents there are in a file and on which volumes or disks these extents are stored.

Files on Net-Storage always have (only) one extent.

Storage space is reserved in “allocation units”, the smallest memory management unit. An allocation unit comprises a specific number of 2-Kbyte blocks; currently available sizes are 6K, 8K and 64K. Systems support can define the size of an allocation unit at pubset generation, subject to certain restrictions depending on the disk format. Pubsets including key disks, for instance, accept 6K units only.

The table below shows the interdependencies of rounding factor k, allocation unit size and corresponding number of PAM pages:

“k”	Unit size in Kbytes	Number of PAM pages
6	6	3
8	8	4
64	64	32

File processing speed can be increased by preventing files from being distributed over too many extents. For instance, files can be reorganized by copying (COPFILE macro or COPY-FILE command).

Physical allocation

Physical allocation is the explicit specification of the storage location for a file; this can be done in any of the following ways:

1. specification of a volume set
2. specification of a volume
3. Absolute allocation
   This means specifying which and how many PAM pages are to be reserved for a file by entering the start address and the size of the area to be reserved in the SPACE operand.

Absolute allocation is not possible for Net-Storage volumes.

Types b) and c) of physical allocation may generally be used for private disks. For pubsets, physical allocation can be used in the following cases:

- for files on public volumes, provided the pubset was made available with the appropriate attribute (see output of the command SHOW-MASTER-CATALOG-ENTRY ...,INFORMATION=*USER)
- for files on public volumes, provided the user has been granted the physical allocation right (see output of the command SHOW-USER-ATTRIBUTES ...,INFORMATION=*PUBSET-ATTRIBUTES).
- for work files in SM pubsets.

Insufficient storage space

If the system cannot satisfy a request for storage space, it proceeds as follows:

- either the macro or command issued for primary allocation is rejected or only part of the requested space is allocated

- in the case of secondary allocation, the program may be aborted.

DMS rejects a space request in the following cases:

- The request would exceed the public space limit available to the user on this pubset and the user has no authorization to exceed his/her public space limit (PUBLIC-SPACE-EXCESS=NO in the user entry).
- Because there is insufficient free space on the pubset or private disk in question, further allocation of storage space is not possible.
- The request was submitted for Net-Storage and the use of Net-Storage is not permitted in the user entry (NET-STORAGE-USAGE=*NOT-ALLOWED). By default the use of Net-Storage is permitted, but no limit can be specified.

If the new request would make the number of necessary extents too large for the extent list in the catalog entry or if the space request exceeds the free capacity of a private disk, DMS executes a partial allocation.
If a request for storage space would exceed the maximum file size that can be represented in the catalog entry, only the maximum possible partial allocation is made.

5.2.1.1 Peculiarities of pubsets

With the exception of physical allocation, the user has no influence on the selection of the disks on which storage space is allocated (if the pubset consists of more than one disk). The FSTAT macro, the SHOW-FILE-ATTRIBUTES command or the SPCCNTRL utility routine provide the relevant information about the individual disks on which the file is residing.

Each pubset contains the following information:

- an entry for each user who may access this pubset, implicitly excluding all other users
- an upper space limit, the public space limit, for each user with access rights
- An entry for each user with access rights specifies whether he/she may exceed his/her public space limit (PUBLIC-SPACE-EXCESS=*NO in the user entry).

A user who is not authorized to access a pubset cannot request memory space on this pubset.

If a user has access authorization for a pubset, he/she can request storage space until his/her public space limit is exhausted. Only when he/she may exceed his/her public space limit may he/she also request storage space in excess of this.

> i Storage space which a user occupies on Net-Storage is not counted in his/her public space limit.

The operator receives a message if a user exceeds his/her public space limit. If this limit is still exceeded at the end of the job, DMS issues a corresponding message to the user during LOGOFF processing.

The storage space limit is set by systems support. The user can display his limit by means of the SHOW-USER-ATTRIBUTES command.

Extending storage space

If the requested storage space is insufficient, it can be extended. This can be done either explicitly, for example, using the MODIFY-FILE-ATTRIBUTES ...,SUPPORT= *PUBLIC-DISK(SPACE=*RELATIVE(...)) command or the FILE ...,SPACE=... macro, or implicitly during processing by DMS. When allocating additional storage space, DMS tries as far as possible to avoid splitting extents by assigning extents that are contiguous to the extent occupied last.

Releasing storage space

The user can release storage space using the ERASE macro (SPACE operand) or the DELETE-FILE command (operand OPTION=SPACE). The catalog entry of the affected files still exists, and looks just like a catalog entry created by means of the CATAL macro or the CREATE-FILE command.

The user can release reserved storage space which has not been used by means of the FILE macro (SPACE=<-n>) or the MODIFY-FILE-ATTRIBUTES command (operand SPACE=*RELEASE(...)). If the catalog entry indicates DESTROY=YES, the space released in this manner is overwritten. As no unit boundaries are observed during overwriting, the remaining data may possibly be overwritten (backwards) as far as the last occupied page.

5.2.1.2 Peculiarities of private disks

Just like the storage space on public volumes, space on private disks is requested by means of the FILE macro or the CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands. DMS makes the private disk available to the participating jobs for the duration of the requests.

Any amount of storage space can be requested on private disks, limited only by the capacity of the disk. The user may use relative space allocation with primary and secondary allocations for a private disk. Furthermore He/she can specify the disk on which space is to be allocated. The user can also use the absolute allocation to specify which PAM pages are to be reserved.

Extending storage space

Storage space is extended by means of the FILE macro or the MODIFY-FILE-ATTRIBUTES command. If the user does not specify VSNs in the VOLUME operand, the system attempts to reserve space on the last disk which contains the file. If there is not enough space on the disk to meet the entire request, space is reserved on other private disks recorded in the catalog.

If VSNs are specified in the VOLUME operand, the system attempts to reserve storage space on the specified volumes. If there is not enough space free, a partial allocation is made. Disks whose VSNs are recorded in the catalog but not specified in the current macro or command call are ignored.

If a file which, until now, has existed on one private disk is to be extended on a second private disk, the operands VOLUME, DEVICE and SPACE must be specified.
If a large amount of additional space is needed, it is better to request it in several small units, which will probably be assigned on different disks. If a file extends over several private disks, all of these disks must be mounted when the file is opened.

Releasing storage space

The user can release storage space that has been reserved or not allocated by means of the FILE macro (operand SPACE=<-n>) or via the operand SPACE=*RELEASE in the MODIFY-FILE-ATTRIBUTES command. If the catalog entry indicates DESTROY=YES, the space released in this manner is overwritten. In this case, the private disk must be accessible when the macro or command is issued to release the space.

5.2.2 Determining file location on SM pubsets

In an SMS environment, an SM pubset containing volume sets with different attributes is the equivalent of various SF pubsets with different attributes made available to the users of a computer center for distributing their files. As with SF pubsets, the predefined catalog ID (catid) in the path name of a file determines which pubset it is stored on. Thus, selecting a suitable SF pubset corresponds to selecting a suitable volume set of an SM pubset.

The location of files in an SM pubset can be influenced by various instruments such as direct specification of file attributes, storage classes and physical allocation. These can be used simultaneously within an SM pubset.

- The file attributes performance, structure, availability and work file enable users to specify their requirements directly. The system then selects the suitable volume set in accordance with the specified file attributes.
- Storage classes are named combinations of individual attributes. They offer a more convenient way of specifying the desired file attributes. As with direct specification of file attributes, the system then selects the suitable volume set in accordance with the specified storage classes.
- Physical allocation enables users to define a specific location within the public disks of an SM pubset.

The following sections first describe an environment which exclusively uses direct attribute specification. Then storage classes are presented as an extended form of direct attribute specification. Finally, the part physical allocation can play in the utilization of an SM pubset is described.

5.2.2.1 Direct attribute specification

User requirements with regard to performance, availability and file structure are specified via the file attributes already used with SF pubsets.

The user command SHOW-PUBSET-FILE-SERVICES enables users to make efficient use of the benefits of an SM pubset. It lists the services that are logically available in an SM pubset. It provides information about what combinations of the file attributes AVAILABILITY, FORMAT, PERFORMANCE, USAGE, DISK-WRITE (and WORK-FILE) which are relevant to the storage location and that are possible on an SM pubset, about impossible combinations, and about combinations that are possible but cannot be optimally implemented. The determination of available services takes neither the current occupancy rate of individual volume sets nor user authorizations or user contingents into account.
The following file attributes relevant for file location are used to specify user requirements:

Preformat

The meaning of the structure attributes (K, NK2, NK4) is the same for SM pubsets and SF pubsets. However, the mechanisms used to assign structure attributes to a file are different. One feature of SM pubsets that is particularly important is that information about the file format can be supplied at the time storage space is reserved for a new file or a file that has already been cataloged but not yet been allocated any storage space. So that the system can take this information into account when selecting the volume set. The format information is supplied via the file attribute PREFORMAT, which is assigned to a file at initial storage space allocation via the commands CREATE-FILE and MODIFY-FILE-ATTRI-BUTES (FILE-PREFORMAT operand). (N.B.: “Initial storage space allocation” includes the allocation of storage space to a file that has been cataloged but not yet allocated any storage space.) For files created via the FILE macro, the preformat is derived, among other things, from the user specification for BLKCTRL and BLKSIZE made at file creation (i.e. at the time of storage space allocation). The term “preformat” indicates that this is a preliminary format. The final file format is not defined until the file is opened for creation. The volume set on which storage space has been reserved for the file may be found to be incompatible with the desired file format at OPEN time. In this case, OPEN processing implicitly relocates the file to a suitable volume set, if available. To keep system overhead low and to avoid the risks involved in relocation (memory bottlenecks etc.), it is highly advisable to ensure that a suitable preformat is specified for files of an SM pubset at the time storage space is allocated. If no preformat is explicitly specified for a file, a default preformat is assumed. (For information on assigning a default, see section “Defaults forlocation-relevant file attributes”.)

Performance

The file attributes PERFORMANCE, USAGE, DISK-WRITE for SF pubsets are also used to specify performance requirements for files on SM pubsets. However, these performance attributes have a more general meaning with SM pubsets, since they then not exclusively refer to cache utilization. The initial effect of the attributes statically assigned via the commands CREATE-FILE and MODIFY-FILE-ATTRIBUTES is that a suitable volume set is selected at file creation. This is not necessarily a volume set equipped with a cache because there are other means to improve performance. The statically assigned values may be reduced for current file processing via dynamic modification of the performancerelevant file attributes PERFORMANCE and USAGE. This has no effect on the location of a file; it only refers to the utilization of the cache of the volume set where the file resides at OPEN time. If the volume set has no cache, dynamic modification of the performance attributes has no effect.

Availability

The file attribute AVAILABILITY was introduced to specify to what extent a file is expected to be failsafe (CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands, AVAIL operand in the FILE and CATAL macros). In contrast to the performance attributes, this specification has a binding effect, i.e. the attribute is guaranteed. The specification AVAILABILTY=*HIGH is rejected for work files and temporary files. It is likewise rejected if no suitable volume set is available or if the contingents assigned to the user ID are exceeded. If AVAIL-ABILTY=*HIGH is requested for a file that is already occupying storage that only meets AVAILABILITY=*STD requirements, the file is relocated to a suitable volume set.

Work file

Work files are files that are assigned the file attribute WORK-FILE.

This is not possible unless a work volume set exists on the SM pubset (see the SHOW-PUBSET-DEFINITION-FILE command). A description of work files is given on "Work files".

Prospective file size

The prospective file size influences the selection of a suitable volume set with regard to the allocation unit size. It cannot be explicitly specified by the user and is deduced from the values specified for the primary and secondary allocation instead. Users can simply specify whether or not their files may grow to become so-called “large files” (file size >= 32 GB) (see "Files larger than 32 GB").

Validity of location-relevant file attributes

The file attributes performance, availability, preformat and work file, which are relevant for the location of a file, are assigned when the file is allocated storage space for the first time. Both space allocation and attributes remain valid until either modified by the user or until the space allocation is cancelled. If a file is cataloged but not allocated any storage space (not even on any background volumes), the file attributes relevant for the location of the file (with the exception of the performance attributes) are undefined. They are determined on the basis of the user specifications made when the user (again) requests storage space for the file (MODIFY-FILE-ATTRIBUTES command).

Adapting file location in accordance with modified requirements

The MODIFY-FILE-ATTRIBUTES command enables the user to modify the static performance attributes PERFORMANCE, USAGE, DISK-WRITE as well as the AVAILABILITY attribute of an existing file, thus informing the system of modified performance and availability requirements. The preformat cannot be modified. A higher availability cannot be requested unless covered by the service range of the SM pubset. Note that the volume set on which the file is currently residing may prove to be less than optimally suitable for or even incompatible with the new performance or availability attributes.

If the new file attributes are incompatible with the current volume set, the file is implicitly and immediately relocated to a compatible volume set, if available. If the current volume set is no longer ideally suited, the file remains on the volume set, even if user requirements are not fully met or resources not optimally used. Systems support then has to restore optimum distribution of files over the individual volume sets by means of pubset reorganization. Implicit reorganization occurs whenever a file is recalled to the processing level from a background level, since the selection of a volume set in this case is based on the current file attributes.

Being performed at the physical level, the activities involved in file relocation are (largely) invisible to the user. In particular, they have no effect on file addressing and do not require any adaptation of the user JCL.

5.2.2.2 Storage classes

A storage class represents a specific combination of values for the file attributes AVAIL-ABILITY, PERFORMANCE, USAGE, DISK-WRITE, FILE- PREFORMAT and WORK-FILE. In addition, systems support can link storage classes to a predefined list of volume sets to be given priority (but not to be considered exclusively) by the system during storage space allocation for a file. The use of storage classes can be restricted by assigning guards profiles.

Information about the storage classes created by systems support and available to users as well as their characteristics (with the exception of the associated list of volume sets, if any) is supplied by the SHOW-STORAGE-CLASS command. The STORAGE-CLASS operand of the commands CREATE-FILE and MODIFY- FILE-ATTRIBUTES or the STOCLAS operand of the macros FILE and CATAL can be used to inform the system of the requirements for a file by assigning it a suitable storage class (instead of specifying file attributes directly).

In comparison with direct attribute specification, storage classes enable a convenient user interface to be provided which is tailored to the services offered by a specific SM pubset. Using storage classes rather than direct attribute specification permits JCL procedures to be kept concise and easy to understand.

Storage classes are defined on a pubset-specific basis.

To make use of the benefits offered by a particular SM pubset, users require information about available storage classes and their meaning; knowledge of the physical configuration implemented by the services is not required.

Modifying the service requirements for a file

The user can make allowance for changing requirements for a file by assigning it a new storage class. This implicitly assigns the file attributes specified in the storage class definition to the file (with the exception of the preformat). As a result of changed requirements for a file, the volume set on which the file is currently residing may no longer be suitable because it does not (fully) comply with the new file attributes or is not included in the list of volume sets associated with the new storage class.

Immediate relocation of the file to another volume set takes place when the new file attributes are not compatible with the file’s current storage location. Immediate relocation can also take place when the volume set to which the file belongs does not belong to the volume set list of the new storage class. Where immediate relocation of the file is not required, the changed requirements will not take effect until the next recall of the file from a background level to the processing level. This could take place, for instance, as a result of pubset reorganization initiated by systems support. The user can modify individual attributes for a

file assigned to a storage class via direct attribute specification. In doing so he/she can overrule the previous storage class assignment. This fact must be taken into account, in particular if the storage class is linked with a volume set list.

Modifying the characteristics of a storage class

Modifications of volume set lists automatically take effect even for existing files, since they will be taken into account by subsequent recalls. Modifications of the combination of attributes represented by a storage class, on the other hand, do not automatically apply to existing files; this can be remedied by systems support or the user explicitly updating the storage class (command MODIFY-FILE-ATTRIBUTES (STORAGE-CLASS=*UPDATE)).

5.2.2.3 Physical allocation

The TSOS privilege implies the general authorization for physical allocation on all user IDs. Systems support can assign individual user IDs the right to physical allocation of their files (see also "Physical allocation" (Requesting storage space)). Physical allocation is performed via the operands VOLUME-SET, VOLUME and SPACE=*ABSOLUTE(..) of the commands CREATE-FILE and MODIFY-FILE-ATTRIBUTES. The VOLUME-SET operand is relevant for SM pubsets only.

The following level of detail may be specified:

1. Physical allocation at block level The caller specifies both the volume and the blocks within that volume on which storage space is to be reserved.
2. Physical allocation at volume level The caller specifies the volumes of the pubset on which the file is to be stored. Where exactly on the volumes storage space is reserved is irrelevant for the caller.
3. Physical allocation at volume set level The caller specifies the volume set on which the file is to be stored. Where exactly on the volume set storage space is reserved is irrelevant for the caller.

> i Physical allocation is possible for Net-Storage volumes at volume level. The privilege for physical allocation is not required in this case.

Information functions for users with the right to physical allocation

Functions providing information which permit a view of the physical configuration are available to users with the right to physical allocation.

- The SHOW-PUBSET-DEFINITION-FILE command provides detailed information on the pubset configuration and the pubset characteristics of SM pubsets.

  For the individual volumes it displays the utilization type (STANDARD, HSMS-CONTROLLED, WORK), allocation constraints (NEW-FILE-ALLOCATION) and performance attributes. The performance and availability characteristics rely on the definitions made by systems support, a more detailed description of the underlying physical features (cache type, DRV) is not given.
- The SHOW-PUBSET-RESTRICTION command shows the volume sets of the SM pubset, the volumes which belong to it and – as for SF pubsets – the device type and allocation constraints for the individual volumes.

Example

```
/SHOW-PUBSET-DEFINITION-FILE PUBSET=SMS1,VOLUME-SET=VS01——————————————————————— (1)
-----------------------------------------------------------------------------
COMMAND: SHOW-PUBSET-DEFINITION-FILE
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
PUBSET SMS1: TYPE = SYSTEM-MANAGED, VOLUMESETS = 3, DEFAULT FILE FORMAT = NK4
---- VOLUME-SET INFORMATION --------- + ------------------------------------
VOLUME-SET VS01: NORMAL-USE, NK4-FORMAT
---- GLOBAL ATTRIBUTES      --------- + ------------------------------------
 AVAILABILITY                         | STANDARD
 USAGE                                | STANDARD
 FORMAT                               | NK4-FORMAT
 MAXIMAL I/O LENGTH                   | 48  HP
 ALLOCATION UNIT SIZE                 | 4   HP
 DRV-VOLSET                           | NO
 NEW FILE ALLOCATION                  | NOT RESTRICTED
 VOLUME SET ACCESS                    | NOT RESTRICTED
---- PERFORMANCE ATTRIBUTES --------- + -----------------------------------
 PERFORMANCE                          | STANDARD
 WRITE-CONSISTENCY                    | BY-CLOSE
-----------------------------------------------------------------------------
```

(1) The physical volume set/volume configuration of SM pubset SMS1 is displayed.

```
/SHOW-PUBSET-RESTRICTION PUBSET=SMS1————————————————————————————————————————  (2)
-----------------------------------------------------------------------------
COMMAND: SHOW-PUBSET-RESTRICTION
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
PUBSET SMS1: TYPE = SYSTEM-MANAGED, VOLUMESETS = 3, DEFAULT FILE FORMAT = NK4
---- PHYSICAL CONFIGURATION --------- + ------------------------------------
---- VOLUME-SET INFORMATION --------- + ------------------------------------
VOLUME-SET VS00: VOLUMES = 2
 VOLUME CONFIGURATION:
  VOLUME    DEVICE     ALLOCATION       VOLUME    DEVICE     ALLOCATION
  VS00.0    D3435      NOT RESTR        VS00.1    D3435      NOT RESTR
VOLUME-SET VS01: VOLUMES = 1
 VOLUME CONFIGURATION:
  VOLUME    DEVICE     ALLOCATION       VOLUME    DEVICE     ALLOCATION
  VS01.0    D3435      NOT RESTR
VOLUME-SET VS02: VOLUMES = 1
 VOLUME CONFIGURATION:
  VOLUME    DEVICE     ALLOCATION       VOLUME    DEVICE     ALLOCATION
  VS02.0    D3435      NOT RESTR
-----------------------------------------------------------------------------
```

(2) The global volume set attributes (availability, utilization type, allocation constraints, etc.) and the performance attributes (performance profile and write consistency) of the volume set VS01 in SM pubset SMS1 are displayed.

For SM pubset entries, the SHOW-MASTER-CATALOG-ENTRY command supplies information about global pubset states (e.g. accessible, inaccessible, local, remote, see the “HIPLEX MSCF” manual [11 (Related publications)]). For volume set entries, the command supplies the associated pubset. The structure of SM pubsets must also be account where the STAMCE macro interface is used:

- Global pubset states (accessible, inaccessible, etc.) are indicated via the MRSCAT entry for the pubset (as for SF pubsets).
- With SM pubsets, configuration data concerning cache assignments, structure, format, etc. is volume-set-specific and is therefore supplied in the MRSCAT entries for the volume sets. These entries also contain information about the pubset to which the volume set belongs.

Detailed information down to the level of individual blocks is also supplied for SM pubsets by the utility routine SPCCNTRL (DISPLAY function), see the “Utility Routines” manual [13 (Related publications)].

Taking file migration inhibits into account

The file attributes S0-MIGRATION and MIGRATE determine whether a file is allocated a fixed storage location or not. The user can explicitly assign these attributes to a file e.g. via the commands CREATE-FILE and MODIFY-FILE-ATTRIBUTES. They serve as an interface between the user and systems support or the system.

They permit the special requirements of user-controlled files to be recognized and taken into account with little effort when the activities “pubset reorganization” and “migration” (using software product HSMS) are performed which are normally organized by systems support. This is done, for instance, by HSMS excluding files with the attribute MIGRATE-FORBIDDEN from migration and recalling files with the attribute S0-MIGRATION-FORBIDDEN residing on a background level to their original volume set.

Files on Net-Storage are never migrated to a background level.

Physically allocated files and file migration inhibits

A close relationship exists between physical allocation and determining the storage location of a file; it must therefore be taken into account when files are relocated (using software product HSMS) both within the S0 level as between S0 and the background levels. Physical allocation is usually coupled with the request to fix a specific storage location for a file. If initial storage space allocation (i.e. not the extension of an existing file) is requested via physical allocation without explicitly specifying the location-fixing attributes,
S0-MIGRATION and MIGRATE are implicitly preset. The presettings are dependent on the level of detail used for defining the file location:

- If the file location was specified at volume set level (VOLUME-SET operand), the implicit presettings are S0-MIGRATION=*FORBIDDEN and MIGRATE=*ALLOWED. The system assumes that the file is to reside on the specified volume set throughout the entire processing at S0 level. Relocation within the volume set is permissible. The file may be migrated to a background level during migration or pubset reorganization; its recall to the original volume set is guaranteed.
- In the event of physical allocation at volume or block level, the implicit presettings are S0-MIGRATION=*FORBIDDEN and MIGRATE=*FORBIDDEN. Migration to a background level is not permissible in this case; the location of the file as specified at file creation is retained.

As mentioned earlier, file migration inhibits may be modified explicitly. Since this has a similarly strong effect on a pubset's space utilization as physical allocation, the right to physical allocation is required to explicitly set S0-MIGRATION=*FORBIDDEN and MIGRATE=*FORBIDDEN.

Relation between physical allocation and file attributes

The performance, availability, format and work file attributes serve to specify the requirements for a file. The system takes these requirements into account when selecting a suitable volume set. With physical allocation, the file attributes are irrelevant for determining the file location, since the volume set is in this case specified by the user. However, the file attributes have other functions which are relevant for physically allocated files as well:

- If a volume set is equipped with a cache, the performance-related file attributes control cache utilization.
- Usually, the format of files residing on a volume set (K, NK2, NK4) is not unequivocally defined by that volume set. Additional and more detailed specifications are required.
- The file attributes serve to determine the user contingents “debited” by the storage space occupied by a file.

Files are consequently assigned valid values for the file attributes PERFORMANCE, IO-USAGE, DISK-WRITE, AVAILABILITY, PREFORMAT and USAGE even if they are created by means of physical allocation. However, physical allocation and non-physical allocation differ in certain aspects such as default values and consideration of explicit user specifications:

- Attributes already assigned as a result of the volume set selected are taken into account and entered.
- If any user-specified attributes are incompatible with the characteristics of the volume set selected, the user request is rejected.
- Storage classes and physical allocation cannot be used in combination.

Relation between S0 migration inhibit and file attributes

As with physical allocation, it must be ensured that contingent monitoring cannot be ignored when modifying the attributes of a file for which an S0 migration inhibit is set or when setting the migration inhibit for a file. Therefore, current or user-specified attributes may be modified automatically in these cases as well. If, for instance, a user creates a file with the attribute AVAILABILITY=*HIGH, subsequently changing it to AVAILABILITY=*STD, S0-MIGRATION=*FORBIDDEN by means of a MODIFY-FILE-ATTRIBUTES command, the file is implicitly reassigned the original attribute AVAILABILITY=*HIGH.

5.2.2.4 Determining the preformat

Determining the file format (K, NK2, NK4) is more complex with SM pubsets than with SF pubsets. During initial allocation, the system defines a preformat for the file with a view to optimal file location selection, and enters it in the catalog entry. The final file format is not determined until the file is opened, when it is derived from the preformat, the actual file location and the OPEN parameters (see section “Determining the file format”). As mentioned earlier (see section “Direct attribute specification”), it is still advisable to inform the system of the desired file format during initial allocation. This section is designed to show what options are available for this and what defaulting mechanisms are provided by the system.

The system offers two global defaulting mechanisms:

- pubset default format
- user-specific default storage class

Pubset default format

In analogy to SF pubsets whose format is implicitly defined by the format of their disks, a default format for SM pubsets is determined by systems support at pubset generation. This default format may change during a pubset session as a result of reconfiguration measures taken by systems support.

Information about the pubset default format can be obtained via the commands SHOW-PUBSET-DEFINITION-FILE and SHOW-PUBSET-FILE-SERVICES PUBSET=..., FILE-FORMAT=*BY-PUBSET-DEFAULT, or by means of the STAM macro.

User-specific default storage class

Systems support can define a specific default storage class for each user on each SM pubset. The FILE-PREFORMAT value entered in this storage class (or, with *BY-PUBSET-DEFAULT, the resulting value) is copied to the file's catalog ID under certain conditions (see below).

The command SHOW-USER-ATTRIBUTES PUBSET=...,INFORMATION=*PUBSET-ATTRIBUTES indicates whether or not a user-specific default storage class exists. The command SHOW-STORAGE-CLASS supplies information about the contents of the storage class.

The ways in which the system determines the preformat are the complete direct attribute specification or the defaulting mechanisms:

1. Complete direct attribute specification

   In this case an unequivocal preformat can be deduced from the user's attribute specification. This is true in the following cases:

Specified attributes	Resulting FILE-PREFORMAT
FILE-PREFORMAT=K	K
FILE-PREFORMAT=NK2	NK2
FILE-PREFORMAT=NK4	NK4
BLKCTRL=PAMKEY	K

BLKCTRL=DATA2K	NK2
BLKCTRL=DATA4K	NK4

BLKCTRL=DATA / NO
and
BLKSIZE=(STD,n)
(n = odd)
	NK2

BLKCTRL=DATA / NO
and
BLKSIZE=(STD,n)
(n = even)
	NK4

2. Defaulting mechanisms

   Where direct attribute specification is not complete, the preformat is determined via various defaulting mechanisms. The sequence in which the various mechanisms are employed by the system is shown in the table below:

User specifications		**Resulting FILE-PREFORMAT**
1.	Physical allocation	According to volume set characteristic[1)]
2.	Storage class specified with FILE-PREFORMAT not equal *BY-PUBSET-DEFAULT	According to preformat in the specified storage class [1)]
3.	No specification of any storage-class-relevant attributes [2)] and a default storage class exists with FILE-PREFORMAT not equal *BY-PUBSET-DEFAULT	According to preformat in the default storage class [1)]
4.	Other	According to pubset default format [1)]

1) If the resultant preformat is K and *DATA or *NO is specified for BLKCTRL, NK2 isset instead of K.

2) The following are storage class-relevant attribute specifications:

- Command interface (CREATE-FILE/MODIFY-FILE-ATTRIBUTES): Specification of *NONE for STORAGE-CLASS, or of one of the operands AVAILABILITY, DEVICE-TYPE, DISK-WRITE, VOLUME, VOLUME-SET, WORK-FILE, PERFORMANCE or USAGE
- Macro interface (FILE): for any of the operands IOPERF or IOUSAGE, or a value other than of *NONE for STOCLAS, or of any of the operands AVAIL, DEVICE, DISKWR, VOLUME, VOLSET or WORKFIL.

5.2.2.5 Determining the file format

The final file format is reflected by the attributes BLOCK-CONTROL-INFO (or BLKCTRL) and BUFFER-LENGTH (or BLKSIZE). These are not determined until the file is opened (OPEN macro, OPEN mode OUTPUT or OUTIN). The following factors are decisive for the file format:

- the values specified for FCBTYPE, BLKCTRL and BLKSIZE; the TFT, FCB and the catalog entry are used as sources from which the decisive input values are obtained during OPEN processing (see chapter "OPEN processing")
- the format of the actual file location (K, NK2, NK4)
- the preformat: For files with CREATION-DATE=*NONE, this is the value of the FILE-PREFORMAT attribute. For files already possessing a CREATION-DATE, this is the current file format (derived from the existing attributes FCBTYPE, BLKSIZE and BLKCTRL).

Usually, the specified attribute values are taken to determine the file format, or the following defaults are assumed:

File preformat	Default for BLKSIZE
K / NK2	(STD,1)
NK4	(STD,2)

File preformat	FCBTYPE	Default for BLKSIZE
K	PAM / SAM / ISAM	PAMKEY
NK2 / NK4	SAM	DATA
NK2 / NK4	PAM	NO
NK2	ISAM	DATA2K
NK4	ISAM	DATA4K 1)

1) If FCBTYPE=ISAM and BLKCTRL=DATA4K, any specification of BLKSIZE=(STD,n) with an odd number for "n" will be rejected. The default value is always (STD,2).

The following modifications are applied during OPEN processing if the resulting file format is incompatible with the file location, or if any specifications are incompatible with the access method:

	Old file preformat	New file preformat
BLKCTRL=PAMKEY	NK2 / NK4	K
BLKCTRL not equal PAMKEY and BLKSIZE=(STD,n) where n is odd	NK4	NK2

FCBTYPE	Old BLKCTRL value	New BLKCTRL value
SAM	DATA2K / DATA4K / NO	DATA
PAM	DATA2K / DATA4K	NO
ISAM	DATA / NO	• DATA4K,if the resulting preformat is NK4 • Otherwise DATA2K

5.2.2.6 Defaults for location-relevant file attributes

User specifications for file attributes are mapped onto actual values by means of defaulting mechanisms; these supply any values not explicitly specified by the user and may even modify explicitly specified values, where necessary. The following defaulting mechanisms exist:

- defaults for individual location-relevant attributes via storage classes and default storage classes (STORAGE-CLASS attribute)
- default for the file format (FILE-PREFORMAT attribute)
- default for the availability (AVAILABILITY attribute)
- default for the performance-related attributes (PERFORMANCE attribute)
- default for file type “work file” (WORK-FILE attribute)
- default for the S0-MIGRATION attribute
- default for the MIGRATE attribute

These mechanisms are explained below for the command interface (mostly CREATE-FILE and MODIFY- FILE-ATTRIBUTES). The comments apply analogously to the program interfaces (macros CATAL and FILE).

Assigning defaults for the individual location-relevant attributes via storage classes and default storage classes

Storage classes must be defined by systems support. The values defined in a storage class are used to determine the location-relevant file attributes WORK-FILE, PERFORMANCE, USAGE, DISK-WRITE, FILE-PREFORMAT and AVAILABILITY. The performance value may be reduced by the system, taking the user authorization DMS-TUNING-RESOURCES into account.

Default storage classes (user- and pubset-specific)

Systems support can define a specific default storage class for each user on each SM pubset. This causes the attribute settings of the user-specific storage class to be assumed for each initial storage space allocation on the pubset under that user ID unless any of the following specifications is made:

- Explicit specification of a storage class
- PERFORMANCE *)
- STORAGE-CLASS=*NONE (*)
- USAGE *)
- WORK-FILE=*NO *)
- DISK-WRITE *)
- WORK-FILE=*YES
- AVAILABILITY *)
- Physical allocation

*) If there is no authorization to perform physical allocation then the default storage classoverrides the specifications marked with *) (and not vice versa).

The command SHOW-USER-ATTRIBUTES PUBSET=...,INFORMATION=*PUBSET-ATTRIBUTES indicates whether or not a user-specific default storage class exists.

The user can define a default storage class for each file generation group in an SM pubset (STOR-CLASS-DEFAULT operand of the CREATE-FILE-GROUP command). During initial storage space allocation for a file generation, the attribute settings of this storage class are used subject to the same conditions as for files. The default storage class of a file generation group can be specified explicitly, or the user-specific default storage class (STOR-CLASS-DEFAULT=*STD) of the SM pubset where the file generation group is created is assigned.

Assigning a storage class

A storage class can be assigned to a file (or file generation) either explicitly (by specifying STORAGE-CLASS=<composed-name>) or via the default (STORAGE-CLASS=*STD). A file or file generation cannot be assigned a storage class until it is allocated storage space. At that time the location-relevant attributes are assigned as well.
Until a file has been allocated storage space, the values of the location-relevant file attributes AVAILABILITY, WORK-FILE and FILE-PREFORMAT as well as the storage class assignment are undefined. This also applies if a file was allocated space and the entire space has been released (commands DELETE-FILE or MODIFY-FILE-ATTRI-
BUTES...,SPACE=*RELEASE(...)).

The following must be taken into account when cancelling a storage class assignment for a file (MODIFY-FILE-ATTRIBUTES...,STORAGE-CLASS=*NONE):

The name of the storage class is removed from the catalog entry. Only location-relevant attributes specified explicitly are modified. To ensure that contingents, if any, are not ignored, the PERFORMANCE and AVAILABILITY values for files with the (derived) attribute S0-MIGRATION=*FORBIDDEN may, however, be modified in accordance with the corresponding values of the volume set on which the file is residing.

Default file preformat

The commands CREATE-FILE and MODIFY-FILE-ATTRIBUTES, as well as the commands for creating and modifying storage classes permit the value *BY-PUBSET-DEFAULT, which is also the default value, to be specified for the PREFORMAT attribute. The effect of this default value differs, depending on whether the file was created via physical allocation or not:

- A file created via physical allocation is assigned the format of the volume set on which it is created as preformat.
- A file not created via physical allocation is assigned the default file format of the SM pubset as preformat. The preformat is taken into account for the selection of a suitable volume set.

In addition, the preformat has an influence on the final file format assigned when the file is opened for creation: If the user opens the file without specifying a format explicitly (ADD-FILE-LINK or FCB), the preformat is used as default.

See the sections on "Determining the preformat" and "Determining the file format" for more details about determining the file preformat and the final file format.

Default availability

*STD is the default availability value for files not created via physical allocation. For files created via physical allocation, the attribute value is determined on the basis of the user specifications as follows:

AVAILABILTY value of the volume set on which the file is created	User-specified file attribute AVAILABILITY	Resulting value of the file attribute AVAILABILITY

HIGH	not explicitly specified HIGH STD	HIGH HIGH HIGH
STD	not explicitly specified HIGH STD	STD Incompatible specification STD

If S0-MIGRATION=*FORBIDDEN is set (explicitly via the MODIFY-FILE-ATTRIBUTES command or implicitly via physical allocation) and the file occupies storage space on a volume set with the attribute AVAILABILITY=*HIGH, the same value AVAILABILITY=*HIGH is set for the file (even if the user explicitly specifies AVAILABILITY=*STD).

Defaults for static performance-related attributes

A distinction is made between static and dynamic values of performance attributes (see section “Sequence of OPEN processing”). Static values are stored in the catalog entry, dynamic values are either stored in the TFT or specified in the TU FCB. Explicit user specifications or values taken from the storage classes may be reduced by the system, taking the DMS-TUNING-RESOURCES entry for the file owner into account.

To ensure that contingent monitoring cannot be ignored when creating a file via physical allocation, the CREATE-FILE command may replace user-specified values or existing settings for the static PERFORMANCE attribute by higher values depending on the performance characteristic of the predefined volume set.

PERFORMANCE values of the volume set	User-specified file attribute PERFORMANCE	Resulting file attribute PERFORMANCE with physical allocation
STD [,HIGH] [,VERY-HIGH]	not explicitly specified STD HIGH VERY-HIGH	STD STD HIGH VERY-HIGH
HIGH [,VERY-HIGH]	not explicitly specified STD HIGH VERY-HIGH	HIGH HIGH HIGH VERY-HIGH
VERY-HIGH	not explicitly specified STD HIGH VERY-HIGH	VERY-HIGH VERY-HIGH VERY-HIGH VERY-HIGH

The higher settings effected by CREATE-FILE may even exceed the file owner's DMS-TUNING-RESOURCES authorization.

The value for static file performance determined by the MODIFY-FILE-ATTRIBUTES command is dependent on the value of the S0-MIGRATION attribute after execution of the command:

- If command execution results in S0-MIGRATION=*FORBIDDEN, the user-specified performance (in the event of explicit specification) or the static performance hitherto assigned to the file may be modified automatically. The rules for creating a file via physical allocation apply in this case as well. Again, the resulting static performance value may exceed the file owner's DMS-TUNING-RESOURCES authorization.

- If command execution results in S0-MIGRATION=*ALLOWED, the performance explicitly specified by the user will be reduced to avoid exceeding the DMS-TUNING-RESOURCES. In the absence of any explicit performance specification, the current performance value is retained.

Default for file type “work file”

By default, files are created as permanent files (WORK-FILE=*NO). A work file can be created using the following mechanisms:

- explicit specification of WORK-FILE=*YES
- physical allocation (explicit specification of a disk that belongs to a work volume set by means of the VOLUME operand or explicit specification of a work volume set via the VOLUME-SET operand)
- specification of a storage class for which WORK-FILE=*YES applies.

No file already occupying storage space can be redefined as a work file.

Default for the S0-MIGRATION attribute

The CREATE-FILE command sets the default value for this attribute. With physical allocation, this results in the allocation S0-MIGRATION=*FORBIDDEN, otherwise in S0-MIGRATION=*ALLOWED.

In the MODIFY-FILE-ATTRIBUTES command, the specification S0-MIGRATION=*UNCHANGED sets the default value for this attribute. For files already occupying storage space, this causes the current value to be retained even if the command simultaneously extends the file via physical allocation. For files which do not yet occupy any storage space and are assigned storage space using the MODIFY-FILE-ATTRIBUTES command, the S0-MIGRATE value is determined as for a newly created file.

Default for the MIGRATE attribute

The CREATE-FILE command sets the default value for this attribute. With physical allocation at volume or block level, this results in MIGRATE=FORBIDDEN. For files not created via physical allocation, a distinction is made between permanent and temporary files: for temporary files MIGRATE=INHIBITED is assigned, and for permanent files MIGRATE=ALLOWED.

In the MODIFY-FILE-ATTRIBUTES command, the specification MIGRATE=*UNCHANGED sets the default value for this attribute.

The effect is different for files already occupying storage space and those not yet occupying storage space:

- For files already occupying storage space, the current value of MIGRATE is retained unless the file is simultaneously converted from a temporary to a permanent file.
- Temporary files are automatically assigned MIGRATE =INHIBITED. Any extension of the file via physical allocation has no effect on the setting of the MIGRATE value.
- For files not yet occupying any storage space and being assigned storage space via the MODIFY-FILE-ATTRIBUTES command, the MIGRATE value is determined as for a newly created file.

5.2.3 Determining the file location on SF pubsets

SF pubsets are built by combining volumes whose attributes should match. These should have identical features since differences between the volumes are not taken into account when the system allocates storage space to files. Caches are assigned at pubset level, not at volume level. (Homogeneous) pubsets are thus containers for files that can be characterized by a specific attribute profile (K or NK2 pubset, DRV pubset, etc.).

Different attribute profiles must be implemented by a set of different SF pubsets. It is the users' task, within the limits of their authorizations, to select those pubsets as storage locations for their files that are best suited to their requirements. To be able to select the appropriate pubsets, users require information about the computer center's configuration. The following interfaces are provided to obtain information about installed pubsets and their attributes:

- The information blocks SUMMARY and PHYSICAL-CONFIGURATION output by the SHOW-PUBSET-CONFIGURATION command supply information about the pubset type (SF or SM), the volumes of the pubset, their device type as well as any allocation constraints that may apply to the volumes (unrestricted allocation, only physical allocation permitted, allocation prohibited).
- The SHOW-MASTER-CATALOG-ENTRY command and the STAM macro indicate the global features of an SF pubset with respect to cache assignment, availability, structure, allocation unit, etc.

Once the user has selected a suitable pubset for a file, the selection must be communicated to the system. This is done via the path name of the file which includes the catalog ID of the pubset on which the file is to reside. The catalog ID need not be specified explicitly, i.e. file names without a catalog ID specification are permissible also. In this case, the system determines the full path name automatically on the basis of the user-specific default catalog ID.

Determining the file format

In contrast to SM pubsets, no preformat is required. The file format as reflected in the attributes BLOCK-CONTROL-INFO (or BLKCTRL) and BUFFER-LENGTH (or BLKSIZE) is determined analogously. The attributes are not determined until the file is opened (OPEN macro, OPEN mode OUTPUT or OUTIN). The following factors are decisive for the file format:

- the attribute values for FCBTYPE, BLKCTRL and BLKSIZE
- the format of the actual file location

Usually, the specified attribute values are taken to determine the file format, or the following defaults are assumed:

Disk format	Default for BLKSIZE
K / NK2	(STD,1)
NK4	(STD,2)

Disk format	FCBTYPE	Default for BLKCTRL
K	any	PAMKEY
NK2 / NK4	SAM	DATA
NK2 / NK4	PAM	NO
NK2	ISAM	DATA2K

NK4	ISAM	DATA4K

The following modifications are applied during OPEN processing if any specifications are incompatible with the access method:

FCBTYPE	Old BLKCTRL value	New BLKCTRL value
SAM	DATA2K / DATA4K / NO	DATA
PAM	DATA2K / DATA4K	NO
ISAM	DATA / NO	• DATA4K if the disk format is NK4 • Otherwise DATA2K

Any specification for BLKCTRL and BLKSIZE that is incompatible with the disk format will be rejected, e.g.:

- BLKSIZE=(STD,1) and BLKCTRL=DATA4K
- BLKSIZE=(STD,1) for a file residing on an NK4 disk
- BLKCTRL=DATA4K, BLKSIZE not specified, FCBTYPE=ISAM for a file residing on a a K or NK2 disk.

5.3 Access to cataloged files

- Access via the path name
- Access via the system default user ID
- Access via the file link name

5.3.1 Access via the path name

Each file access by DMS is preceded by an access to the catalog entry for the file to be processed. When specifying a file name in a macro or command, the user can generally use various combinations of the catalog ID, user ID and file name. Temporary files are normally addressed with the prefix and file name only (the default catalog ID and/or default user ID may be specified).

pathname = [:catid:][$userid.]filename

filename

DMS searches for the file in the catalog belonging to the user ID of the job issuing the request. If this is unsuccessful, the function “secondary read” is performed for several commands (or if the operand OPTION=GLODEF is specified in the FCB macro; see next section). Otherwise, DMS issues an appropriate error message.

$userid.filename

DMS searches for the file in the default catalog assigned to the specified user ID. If the user ID of the job issuing the request is neither the owner nor the co-owner of the file nor TSOS, the file must be shareable. If the file is not found, DMS issues the appropriate error messages.

:catid:$userid.filename

DMS searches for the file in the specified catalog under the specified user ID. If no entry for the file is found there, DMS issues the appropriate error message.

:catid:filename

DMS searches for the file in the named catalog under the user ID of the job which issues the request. Assumed, of course, that there is a corresponding entry in the user catalog for the user ID in the named catalog. A secondary read is possible (see next section). If the access attempt is unsuccessful, DMS issues an appropriate error message.

The catalog entry indicates how and where the file is stored (volume list and position information).

5.3.2 Access via the system default user ID

Utility routines and similar programs are generally cataloged under the system default user ID. If a user wishes to use such routines or procedures, he can bypass the access to his own user catalog by placing this ID or its abbreviation (the $ character) in front of the file name. In the second case, he must note the following:

- If the file name is split into partial names by means of periods, the system default user ID must be specified as “$.”.
- If the file name is not subdivided, the $ character does not need to be separated from the file name by a period.

If a user wishes to access a file under the system default user ID, but does not place “$.” or “$” before the file name, DMS first searches for this file in the user's own user catalog, i.e. the user catalog belonging to the default user ID of the job. If this is unsuccessful, DMS automatically executes a secondary read (a second read access to the catalog belonging to the system default user ID) for commands such as START-PROGRAM and CALL-PROCEDURE.

A secondary read can also be defined for OPEN processing by means of the operand OPTION=GLODEF in the FCB macro.

Wherever possible, the user should avoid using the secondary read function, and should specify the abbreviation for the system default user ID ($ or $.), since this reduces the number of accesses to catalogs.

Example 1: Macro access to files under the system default user ID / secondary read

Cataloged files:	`$USER1.DAT1`
	`$USER2.DAT1`
	`$DEFL.DAT1`
	`$DEFL.DAT2`
	`$DEFL.DAT.V.1`
Standard label:	`$DEFL`

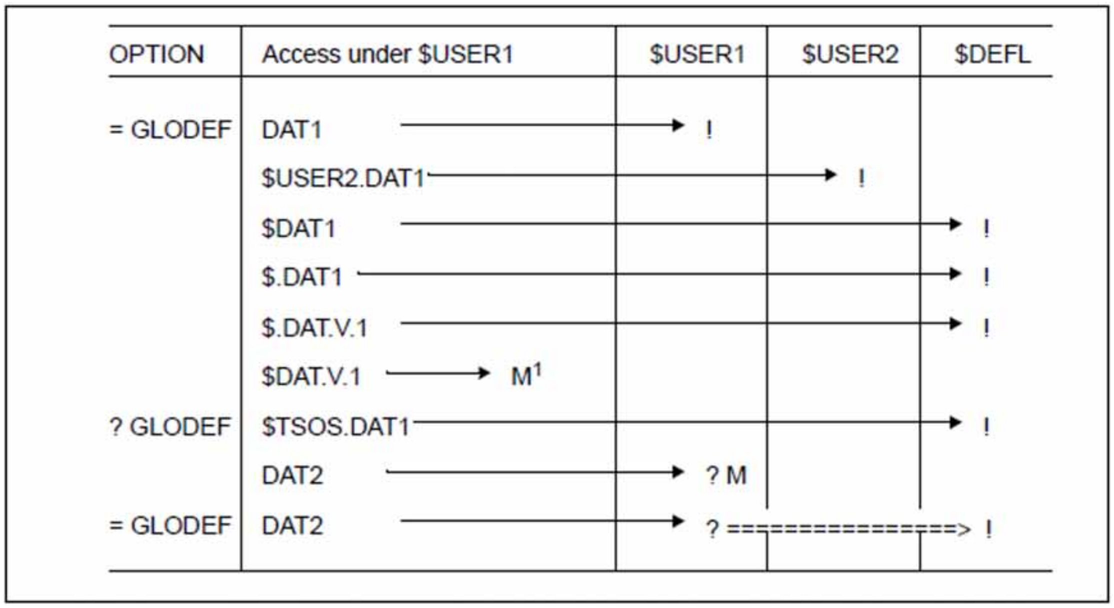


Table 10: Catalog access / secondary read (macro)

Key:

`---->`	First catalog access
`====>`	Secondary read
`?`	Unsuccessful catalog access
`!`	Hit
`M`	Error message
`M`^{1}	Error message: user ID $DAT. does not exist

Example 2: Command access to files under the system default user ID / secondary read

Cataloged files:	`$USER1.DAT1`
	`$USER2.DAT1`
	`$DEFL.DAT1`
	`$DEFL.DAT2`
	`$DEFL.DAT.V.1`
Standard label:	`$DEFL`

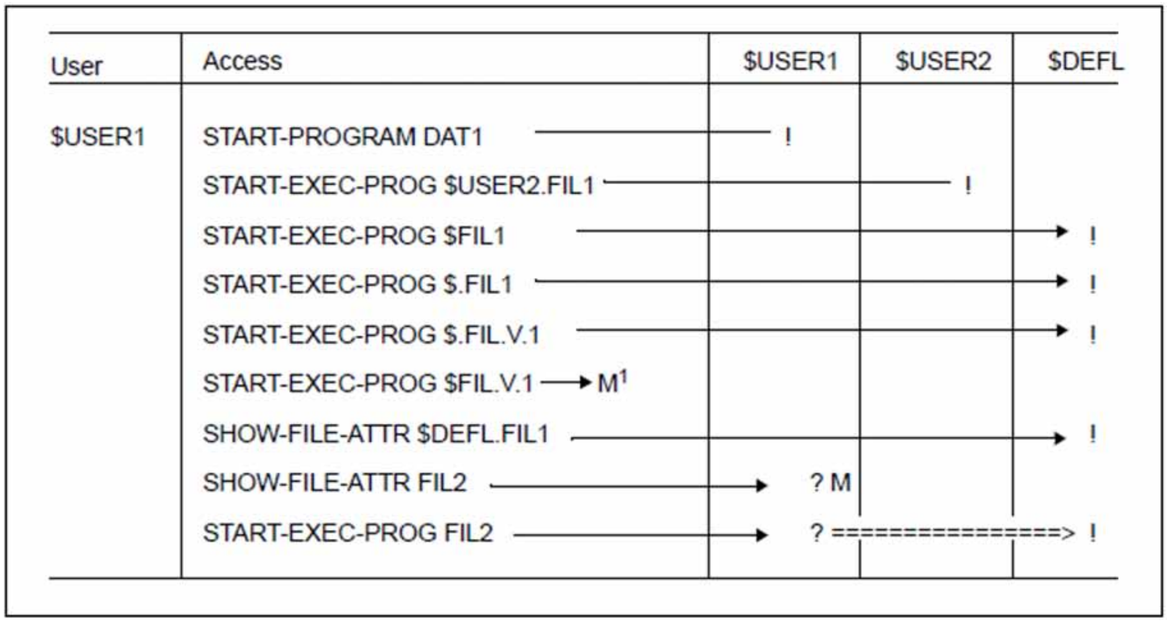


Table 11: Catalog access / secondary read (command)

Key:

`---->`	First catalog access
`====>`	Secondary read
?	Unsuccessful catalog access
!	Hit
M	Error message
M1	Error message: user ID $DAT. does not exist

Example 3: Command access to files under the system default user ID / secondary read

```
/show-file-attributes file-name=edt ———————————————————————————————————  (1)
%       3 :2WR3:$A1234.EDT
%:3WR3: PUBLIC:     1 FILE  RES=         3 FRE=        2 REL=       0 PAGES
/start-exec-program from-file=edt   ————————————————————————————————————  (2)
%  BLS0518 INVALID FORMAT OF LOAD MODULE. COMMAND REJECTED
%  NRTT101 ABNORMAL JOBSTEP TERMINATION BLS0518
/start-exec-program from-file=$edt ————————————————————————————————————  (3)
%  BLS0500 PROGRAM 'EDT', VERSION '17.0A10' OF '2009-03-06' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU TECHNOLOGY SOLUTIONS   2009. ALL RIGHTS
RESERVED
:
%  EDT8000 EDT TERMINATED
/show-file-attributes $edt ————————————————————————————————————————————  (4)
%       6 :3WWW:$TSOS.EDT
%:3WWW: PUBLIC:     1 FILE  RES=         6 FRE=        2 REL=       0 PAGES
```

(1) A file “EDT” is cataloged under the user ID $A1234.

(2) The file “EDT” does not contain an executable program, so the command START-EXECUTABLE-PROGRAM is rejected.

(3) The file editor $EDT, which is cataloged under the system default user ID, is called.

(4) The command SHOW-FILE-ATTRIBUTES displays the catalog entry of the file editor $EDT.

5.3.3 Access via the file link name

If a program is to process files, DMS must be able to establish a connection between the program and the file(s). This is done either by means of a file name which is specified in the program or via a file link name within the program, which must then be assigned to an actual file before the file is opened. Access via a file name specified in a program is carried out as described on "Access via the path name". However, this method has certain drawbacks: the program loses flexibility, and the source code has to be modified if the file name is changed. Output files created in this manner must be renamed or recataloged to prevent them from being overwritten the next time the program is run. If input files are specified in this manner, the program can process only the specified file(s) unless they are repeatedly recataloged. Furthermore, the programmer needs to know the actual file name when he writes the program, a problem that is made worse by the fact that a program is often written for different users and/or different applications.

A file and a program should therefore be linked flexibly with the aid of an internal name within the program, the file link name. A file is assigned a file link name (= link name) by means of an entry in the job-specific task file table (TFT); this entry is created by means of the FILE macro (LINK operand) or the ADD-FILE-LINK command (LINK-NAME operand). This assignment can be canceled (RELTFT macro or REMOVE-FILE-LINK command) or changed (FILE/CHNGE macros or the CHANGE-FILE-LINK/ADD-FILE-LINK commands) at any time, provided the TFT entry is inactive.

TFT entries can be protected against deletion by means of the LOCK-FILE-LINK command; this will cause any REMOVE-FILE-LINK command or RELTFT macro to be suspended for these entries. Locked TFT entries can be unlocked again by means of the DROPTFT macro or the UNLOCK-FILE-LINK command. Any REMOVE-FILE-LINK command or RELTFT macro for this entry which has been suspended is now processed, i.e. the TFT entry will be deleted as specified in the command/macro and the associated private volumes will be released.

The fields in the TFT entry describe the file and processing attributes for the file currently being processed. All fields whose associated operands are specified in the macro or command which creates or uses the TFT entry are supplied with data.

When the file is opened, specifications in the TFT entry override the corresponding fields in the file control block created by means of the FCB macro. Fields in the TFT entry which contain the value “*BY-CAT” or “*BY-CATALOG” (so-called “NULL operands”, see below) are treated specially.

All attributes which are neither specified in the TFT entry nor declared in the file control block are set to default values by DMS.

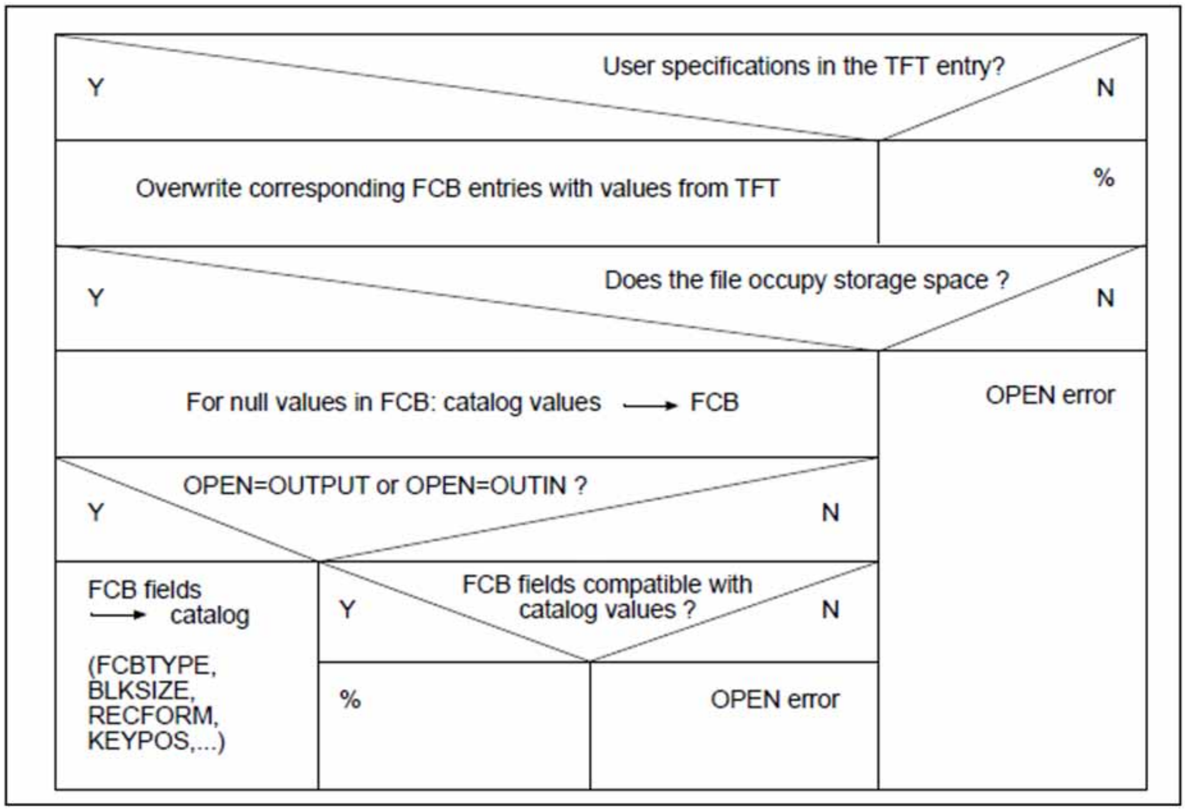

User specifications in the TFT entry?
Y
N
Overwrite corresponding FCB entries with values from TFT
%
Does the file occupy storage space ?
Y
N
For null values in FCB: catalog values → FCB
OPEN error
OPEN=OUTPUT or OPEN=OUTIN ?
Y
N
FCB fields → catalog
(FCBTYPE, BLKSIZE, RECFORM, KEYPOS,...)
FCB fields compatible with catalog values ?
Y
N
%
OPEN error

Figure 4: Compiling file attributes

The user can display the structure and the contents of the TFT entries at any time by means of the RDTFT macro or the SHOW-FILE-LINK command. The default function of the macro and the command tells the user which TFT entries were created for the current job. However, the user can also request information about specific TFT entries.

TFT entries are stored in the order in which they were created by the macros and/or commands and are searched sequentially whenever the table is accessed. Entries which are no longer needed should therefore be deleted, as the number of TFT entries affects system performance.

i Software products such as utility routines, file editors, etc. often use their own file link names; these are described in the appropriate manuals.

Null operands

When processing existing files, the user can use the null operand function. This function ensures that the file control block contains the correct information at OPEN time. Many of the operands in the FILE/ FCB macros and the ADD-FILE-LINK command which describe file attributes may be specified as null operands in the macro (i.e. with a null string as the operand value) or in the command (by means of the operand value “*BY-CATALOG”). The corresponding field of the TFT entry then contains the value “BY-CAT”. When the file is opened, DMS transfers the corresponding values from the catalog entry to the file control block.

If a command or macro specifies specific file attributes for an existing file, the new entries must be compatible with those in the catalog. Compatibility between the FCB and the catalog does not mean that the corresponding values must be the same. For example, it is possible for SAM/ISAM files to be processed using the UPAM access method (access method specified in the FCB is PAM).

Example 1: File link name – sorting with the SORT utility routine

The contents of the file DATEN.UNSORT are to be sorted with the SORT utility and the output file is to be called DATEN.SORT. This file is then to be printed by the program DRUCK as an edited listing with the name LISTE. By default, the SORT utility uses the file link names SORTIN and SORTOUT, while the program DRUCK uses the file link names LISTEIN and LISTAUS.

```
/show-file-attributes daten.unsort ———————————————————————————————————  (1)
%      12 :2OS2:$USER1.DATEN.UNSORT
%:2OS2: PUBLIC:      1 FILE  RES=      12 FRE=       7 REL=      6 PAGES
/add-file-link link-name=sortin,file-name=daten.unsort ————————————————  (2)
/add-file-link link-name=sortout,file-name=daten.sort —————————————————  (3)
/show-file-link   ————————————————————————————————————————————————————  (4)
%
%-- LINK-NAME --------- FILE-NAME -------------------------------------------
%   SORTIN              :2OS2:$USER1.DATEN.UNSORT
%   SORTOUT             :2OS2:$USER1.DATEN.SORT
/start-sort  —————————————————————————————————————————————————————————  (5)
%  BLS0523 ELEMENT 'SRT80', VERSION '079', TYPE 'L' FROM LIBRARY
':1OSH:$TSOS.SYSLNK.SORT.080' IN PROCESS
%  BLS0524 LLM 'SRT80', VERSION '08.0A00' OF '2015-02-03 13:07:17' LOADED
%  BLS0551 COPYRIGHT (C) 2014 FUJITSU TECHNOLOGY SOLUTIONS GMBH. ALL RIGHTS
RESERVED
%  SRT1001  2017-03-03/18:19:43/000000.00 SORT/MERGE STARTED, VERSION
08.0A00/BS2000V20.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
*sort fields=(10,5,a,ch)
*record length=100
*end
%  SRT1222  WARNING: RECORD SIZE OF INPUT FILE NOT EQUAL TO <LENGTH1>.
            <LENGTH1> WILL BE IGNORED
%  SRT1016  SORT/MERGE INPUT RECORDS:..........................101 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:............................101
%  SRT1002  09:49:11/000000.16 SORT/MERGE COMPLETED
/show-file-link ————————————————————————————————————————————————-————  (6)
%
%-- LINK-NAME --------- FILE-NAME -------------------------------------------
%   SORTOUT             :2OS2:$USER1.DATEN.SORT
/change-file-link link-name=sortout,new-name=listein ———————————-————  (7)
/add-file-link link-name=listaus,file-name=liste   ———————————————————  (8)
/show-file-link   —————————————————————————————————————————————————————  (9)
%
%-- LINK-NAME --------- FILE-NAME -------------------------------------------
%   LISTAUS             :2OS2:$USER1.LISTE
%   LISTEIN             :2OS2:$USER1.DATEN.SORT
/start-prog from-file=druck ——————————————————————————————————————— (10)
%  BLS0517 MODULE 'DRUCK' LOADED
:
: *** Create output list in DRUCK program ***
:
*** PROGRAM DRUCK TERMINATED ***
/remove-file-link link-name=listein ——————————————————————————————— (11)
/remove-file-link link-name=listaus ——————————————————————————————— (12)
/show-file-link ——————————————————————————————————————————————————— (13)
%  DMS05E1 TASK FILE TABLE (TFT) NOT AVAILABLE OR SPECIFIED FILE NOT IN
           'TFT'. OPERATION NOT PROCESSED
```

(1) The file “DATEN.UNSORT” contains data .

(2) Link the input file to the SORT utility routine.

(3) Link the output file to the SORT utility routine.

(4) Display the TFT entries.

(5) Call the SORT utility.

“*” Enter the sort statements.

"% SRT..." messages from the SORT utility routine.

(6) Display the TFT entries: the TFT entry for the SORT input file, SORTIN, has been released by the SORT utility.

(7) Change the link name: the output file from the SORT utility routine (SORTOUT) is renamed as the input file (LISTEIN) for the DRUCK program.

(8) Create a TFT entry for the PRINT output file.

(9) Display the TFT entries.

(10) Start the PRINT program.

(11) Delete the TFT entries for the DRUCK input file (LISTEIN).

(12) Delete the TFT entries for the DRUCK output file (LISTAUS).

(13) The SHOW-FILE-LINK command now shows that all the TFT entries for this job have been deleted.

Example 2: Transfer of applicable values from the catalog entry

```
/sh-f-attr lst.bsp.2,inf=par(org=yes)
%0000000003 :2OS2:$USER1.LST.BSP.2
% ------------------ ORGANIZATION -------------------------------------------
% FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
% IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
% REC-FORM   = (V,M)       REC-SIZE   = 0
% AVAIL      = *STD
% WORK-FILE  = *NO         F-PREFORM  = *K          S0-MIGR    = *ALLOWED
%:2OS2: PUBLIC:      1 FILE  RES=        3 FRE=         2 REL=        0 PAGES
/add-file-link link=edtsam,file-name=lst.bsp.2,access-method=*by-cat,
rec-form=*by-cat,buffer-length=*by-cat,block-contr-info=*by-cat
/show-file-link link=edtsam,inf=all
%-- LINK-NAME --------- FILE-NAME -------------------------------------------
%   EDTSAM              :2OS2:$USER1.LST.BSP.2
%  -------------------- STATUS --------------------------------------------
%  STATE      = INACTIVE    ORIGIN     = FILE
%  -------------------- PROTECTION ----------------------------------------
%  RET-PER    = *BY-PROG    PROT-LEV   = *BY-PROG
%  BYPASS     = *BY-PROG    DESTROY    = *BY-CAT
%  -------------------- FILE-CONTROL-BLOCK - GENERAL ATTRIBUTES ------------
%  ACC-METH   = *BY-CAT     OPEN-MODE  = *BY-PROG    REC-FORM   = *BY-CAT
%  REC-SIZE   = *BY-PROG    BUF-LEN    = *BY-CAT     BLK-CONTR  = *BY-CAT
%  F-CL-MSG   = STD         CLOSE-MODE = *BY-PROG
%  -------------------- FILE-CONTROL-BLOCK - DISK FILE ATTRIBUTES ----------
%  SHARED-UPD = *BY-PROG    WR-CHECK   = *BY-PROG    IO(PERF)   = *BY-PROG
%  IO(USAGE)  = *BY-PROG    LOCK-ENV   = *BY-PROG
%  -------------------- FILE-CONTROL-BLOCK - TAPE FILE ATTRIBUTES ----------
%  LABEL      = *BY-PROG   (DIN-R-NUM  = *BY-PROG,   TAPE-MARK  = *BY-PROG)
%  CODE       = *BY-PROG    EBCDIC-TR  = *BY-PROG    F-SEQ      = *BY-PROG
%  CP-AT-BLIM = *BY-PROG    CP-AT-FEOV = *BY-PROG    BLOCK-LIM  = *BY-PROG
%  REST-USAGE = *BY-PROG    BLOCK-OFF  = *BY-PROG    TAPE-WRITE = *BY-PROG
%  STREAM     = *BY-PROG
%  -------------------- FILE-CONTROL-BLOCK - ISAM FILE ATTRIBUTES ----------
%  KEY-POS    = *BY-PROG    KEY-LEN    = *BY-PROG    POOL-LINK  = *BY-PROG
%  LOGIC-FLAG = *BY-PROG    VAL-FLAG   = *BY-PROG    PROPA-VAL  = *BY-PROG
%  DUP-KEY    = *BY-PROG    PAD-FACT   = *BY-PROG    READ-I-ADV = *BY-PROG
%  WR-IMMED   = *BY-PROG    POOL-SIZE  = *BY-PROG
%  -------------------- VOLUME --------------------------------------------
%  DEV-TYPE   = *NONE       T-SET-NAME = *NONE
%  VSN/DEV    = GVS2.5/D3435
```

The operands ACCESS-METHOD, RECORD-FORMAT, BUFFER-LENGTH and BLOCK-CONTR-INFO are specified as “BY-CATALOG” in the ADD-FILE-LINK command, since the applicable values are to be taken from the catalog entry when the file is opened. During processing, these values are copied from the catalog entry to the file *LST.BSP.2* via the link name *EDTSAM*. The TFT entry has the value “*BY-CAT” in the corresponding fields.

5.4 Access to non-cataloged files

Non-cataloged files are always files on private volumes which have been exported by means of ERASE or created in other computer centers. Non-cataloged tape files may even have been created in other operating systems.

5.4.1 Non-cataloged disk files

Before non-cataloged files stored on private disk or Net-Storage volumes can be processed, they can be imported using either the IMPORT or FILE macro (with the STATE=FOREIGN operand) or with the IMPORT-FILE command. The volume on which the file begins must then also be specified so that DMS can create the catalog entry for the file from the F1 label of the private volume or the catalog of the Net-Storage volume.

It is possible to import several files at the same time or even all files which begin on the volume specified in the macro or command. The catalog entry is created automatically from the F1 label of the volume or the catalog of the Net-Storage volume.

If the files to be imported are file generations, the group entry must be created before the generations are imported: The user must first either import the volume containing the group entry or create a group entry using the CATAL macro (with the STATE=FOREIGN operand) or with the CREATE-FILE-GROUP command (with the VOLUME operand). For details see section “Importing file generation groups and generations”.

Files that have been stored in a user-specific directory of a Net-Storage volume by open systems, can be added as node files to the catalog of the Net-Storage volume and the TSOSCAT of the corresponding pubset by using the IMPNFIL macro or the IMPORT-NODE-FILE command. The files must comply with the BS2000 naming conventions. The user can choose whether the file is to be cataloged as a PAM or SAM file. PAM files are cataloged using BLKSIZE=(STD,1) and SAM files using BLKSIZE=(STD,16). The CCS and NETCCS character sets are preset, following the specifications in the user entry.

5.4.2 Non-cataloged tape files

If a user wishes to process a non-cataloged tape file, he/she must describe the file and the associated volumes precisely in the macro or command.

If the file was created with standard labels, the user does not need to specify file attributes such as RECFORM (RECORD-FORMAT), RECSIZE (RECORD-SIZE), BLKSIZE (BUFFER_LENGTH), and CODE since these attributes are taken from the HDR2 label.

If the file has no labels or uses nonstandard labels, all file attributes must be specified explicitly by means of the FILE macro or the ADD-FILE-LINK command as otherwise default values are used. Full access protection is assured for files which have standard labels.

The following tables provide an overview of the operands of the FILE macro and of the ADD-FILE-LINK command which should be used for access to non-cataloged files.

FILE macro

Operand	**Label specification**		
	LABEL=(STD,n) Standard labels	**LABEL=NSTD Non-standard labels**	**LABEL=NO No labels**
filename	+	+	+
LINK	(1)	(1)	(1)
DEVICE	+	+	+
VOLUME	+	+	+
MOUNT	x	x	x
STATE=FOREIGN	+	+	+
SECLEV	x	x	x
FCBTYPE	x	x	x
BLKCTRL	(2)	(2)	(2)
RECFORM	x	+	+
RECSIZE	x	+	+
BLKSIZE	x	+	+
BUFOFF	x	+	+
LABEL	x	+	+
TPMARK		x	x
FSEQ	(3)		(3)
CODE	x	+	+

Table 12: Operands of the FILE macro for non-cataloged tape files

+ The operand is mandatory.

x The operand is optional.

(1) The operand must be specified so that a TFT entry can be created and file attributes such as BLKSIZE, RECFORM etc. can be evaluated.

(2) The operand must be specified if the file is an NK-ISAM file which has been copied to tape.

(3) The operand need not be specified if the file is the first file of a file set or of an MF/MV set.

ADD-FILE-LINK command

Operand	**Label specification**		
	LABEL=(STD,n) Standard labels	**LABEL=NSTD Non-standard labels**	**LABEL=NO No labels**
FILE-NAME	+	+	+
LINK-NAME	(1)	(1)	(1)
ADD-CATALOG-VOLUME	+	+	+
NUMBER-OF-PREMOUNTS	x	x	x
PROTECTION-LEVEL	x	x	x
ACCESS-METHOD	x	x	x
BLOCK-CONTROL-INFO	(2)	(2)	(2)
RECORD-FORMAT	x	+	+
RECORD-SIZE	x	+	+
BUFFER-LENGTH	x	+	+
BLOCK-OFFSET	x	+	+
LABEL	x	+	+
TAPE-MARK		x	x
FILE-SEQUENCE	(3)		(3)
CODE	x	+	+

Table 13: Operands of the ADD-FILE-LINK command for non-cataloged tape files

+ The operand is mandatory.

x The operand is optional.

(1) The operand must be specified so that a TFT entry can be created and file attributes such as BUFFER-LENGTH, RECORD-FORMAT etc. can be evaluated.

(2) The operand must be specified if the file is an NK-ISAM file which has been copied to tape.

(3) The operand need not be specified if the file is the first file of a file set or of an MF/MV set.

5.4.3 Non-BS2000 tape files

Tape files which were not created under BS2000 are treated just like non-cataloged
BS2000 files. However, due to the often widely different principles underlying the operating systems, the full degree of access protection is not always possible. For volumes and files with standard labels in accordance with DIN 66029, owner protection is assured if the VOL1 label contains the owner identifier and neither the volume nor the files is/are shareable.

5.5 ACS: accessing files via an alias catalog system

With ACS (Alias Catalog Service) it is possible to access files and JVs under names which are, within certain limits, freely selectable. Users thus have the option of defining aliases for the files/JVs they require and of storing them in special catalogs, the alias catalogs, together with the assignment to the real file/JV. Only the alias need then be specified when processing files/JVs. Alias catalogs are managed on a task-local basis.

Overview of ACS functions and their advantages

The Alias Catalog Service (ACS) includes three basic functions:

- Alias definition
- Catalog ID insertion for temporary SPOOL files
- Prefix insertion

Alias definition

ACS commands can be used to define aliases for files/JVs within a task. These definitions are placed in a (temporary) catalog known as the alias catalog and can be optionally saved in permanent files and subsequently reloaded into the alias catalog (even by other tasks) if required. When a file/JV is accessed, the actual name is substituted for the alias. Alias definitions are task-local. The alias catalog is deleted on completion of the task.

This function provides independent access to software products without regard to their versions or computer center IDs. The information required for this purpose, however, must be supplied by systems support in the form of an alias catalog system file. Users are thus free to define file/JV names, user IDs, and catalog IDs without the constraints of computer center conventions and version changes; their procedures will still be portable. Files/JVs that belong to unbundled software products can always be addressed with the same alias in an alias catalog regardless of versions and computer center IDs.

Catalog ID insertion for temporary SPOOL files

It can be used to define a system-wide catalog ID (catid) for the temporary files/JVs generated by the SPOOL task. Such files/JVs are then no longer stored on the default pubset of the user task. This function is only relevant for the ACS administrator.

As far as nonprivileged users are concerned, this function is only evident in the output fields of the SHOW-FILE-ATTRIBUTES command.

Prefix insertion

This function can be used within a task to define a specific prefix that is to be implicitly placed before the file name according to appropriate conventions whenever a file name is entered.

The function of prefix insertion offers a convenient way of structuring file/JV names within a user ID. Files can be grouped together from a functional point of view. This allows users to set up a form of sub-catalog within their user ID, for use as a local working environment. This allows the user to set up a form of subcatalog within his/her user ID for use as a local working environment. Under a shared user ID, files/JVs can be addressed under a prefix which designates the name, product or version.

5.5.1 Overview of ACS commands and processing sequence

Command	Meaning
ADD-ALIAS-CATALOG-ENTRY	Adds an entry to the alias catalog of the current task and defines where the substitution applies.
HOLD-ALIAS-SUBSTITUTION	Interrupts the ACS function for the current task. No alias substitution occurs following this command.
LOAD-ALIAS-CATALOG	Transfers entries saved in an AC file to the alias catalog.
MODIFY-ACS-OPTIONS	Modifies task-local options.
MODIFY-ALIAS-CATALOG-ENTRY	Modifies an existing entry in the alias catalog.
PURGE-ALIAS-CATALOG	Deletes the alias catalog of the current task, but retains the options that where set for it.
REMOVE-ALIAS-CATALOG-ENTRY	Deletes an existing entry from the alias catalog.
RESUME-ALIAS-SUBSTITUTION	Resumes the alias substitution function
SET-FILE-NAME-PREFIX	Defines a prefix for file/JV names and specifies where the prefix applies.
SHOW-ACS-OPTIONS	Displays options (for local task).
SHOW-ACS-SYSTEM-FILES	Shows the predefined alias catalog system files.
SHOW-ALIAS-CATALOG-ENTRY	Displays selected alias catalog entries (on SYSOUT).
SHOW-FILE-NAME-PREFIX	Shows the current prefix and its scope.
STORE-ALIAS-CATALOG	Stores the alias catalog in a file.

Table 14: ACS command overview (user commands)

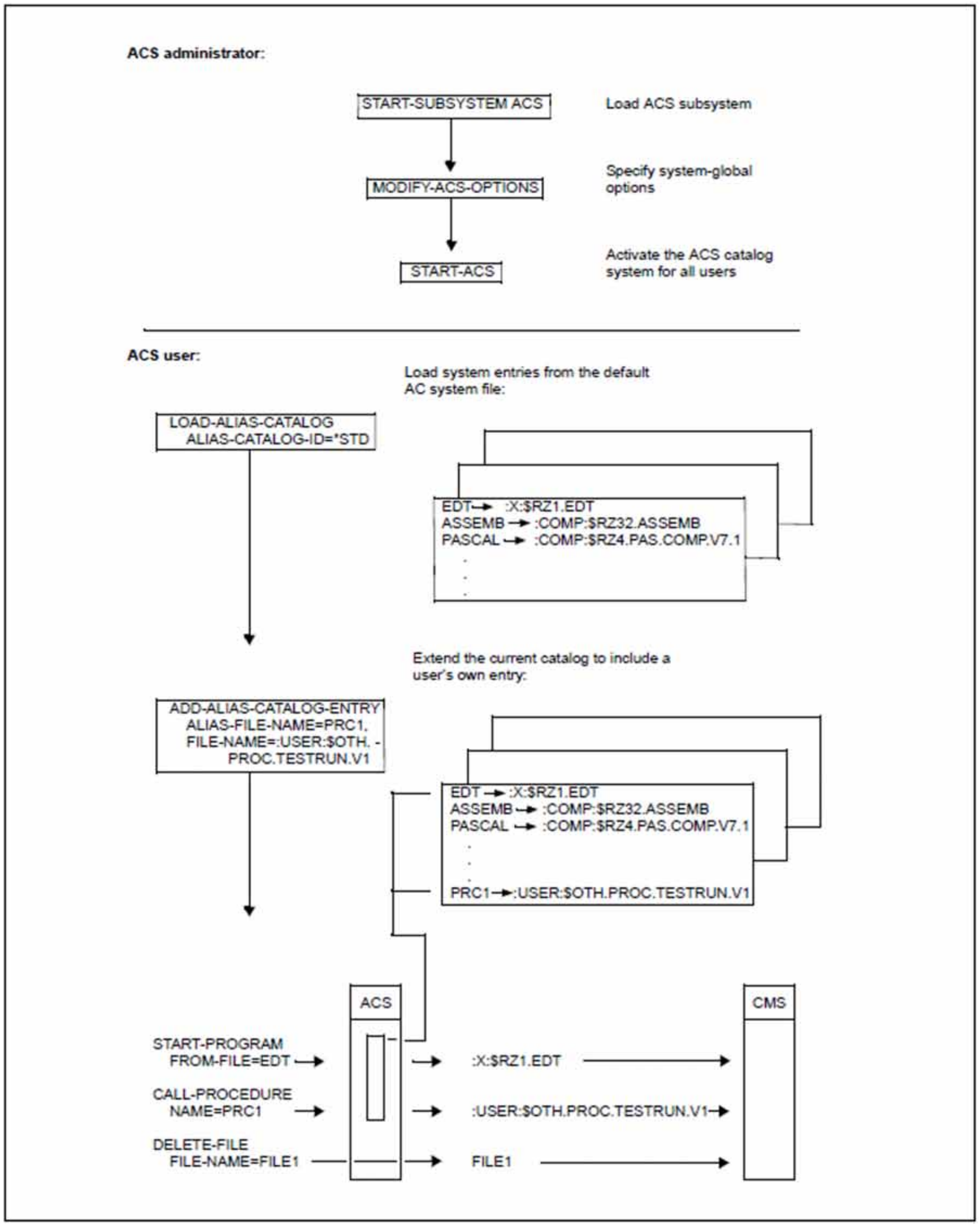

ACS administrator:
START-SUBSYSTEM ACS
Load ACS subsystem
MODIFY-ACS-OPTIONS
Specify system-global options
START-ACS
Activate the ACS catalog system for all users
ACS user:
Load system entries from the default AC system file:
LOAD-ALIAS-CATALOG ALIAS-CATALOG-ID=*STD
EDT → :X:$RZ1.EDT
ASSEMB → :COMP:$RZ32.ASSEMB
PASCAL → :COMP:$RZ4.PAS.COMP.V7.1
Extend the current catalog to include a user's own entry:
ADD-ALIAS-CATALOG-ENTRY ALIAS-FILE-NAME=PRC1, FILE-NAME=:USER:$OTH. - PROC.TESTRUN.V1
EDT → :X:$RZ1.EDT
ASSEMB → :COMP:$RZ32.ASSEMB
PASCAL → :COMP:$RZ4.PAS.COMP.V7.1
PRC1 → :USER:$OTH.PROC.TESTRUN.V1
ACS
CMS
START-PROGRAM FROM-FILE=EDT
:X:$RZ1.EDT
CALL-PROCEDURE NAME=PRC1
:USER:$OTH.PROC.TESTRUN.V1
DELETE-FILE FILE-NAME=FILE1
FILE1

Figure 5: Function of the ACS (“Alias definition”)

5.5.2 The ACS administrator

The ACS administrator is the owner of the ACS-ADMINISTRATION system privilege (if SECOS is not available, the privilege is linked to the TSOS user ID). The ACS administrator has the following ACS options which apply throughout the system:

- Start and parameterization of the AC system (i.e. assignment of system-wide defaults)
- Assignment or restriction of permissions
- Provision and maintenance of the AC system files

The ACS administrator is provided with special enhancements of the ACS user commands and additional administrator commands to perform these tasks.

5.5.3 Alias definition

The “alias definition” function of the Alias Catalog Service enables cataloged files/JVs to be accessed under supplementary file names or aliases, which can be freely defined within certain limits. The files/JVs can also be accessed under their original file names. Any number of aliases may be defined for a file/JV.

Each file name that is passed to DMS by means of a command or macro is checked to determine whether it is defined as an alias in the context of the current task. If the specified file name has been defined as an alias, it is replaced by the corresponding actual file name and then supplemented with the catalog ID and user ID if required. This constitutes a complete path name, which uniquely designates a file/JV. If the specified name is not defined as an alias, it is interpreted as an actual file name.

Alias catalog (AC)

The definition of an alias which can be used instead of an actual file/JV name is stored in the so-called alias catalog (AC). This catalog contains the assignments of aliases to actual file names. Each task has its own alias catalog, and this catalog can only be created, extended, or modified by special ACS commands (see the ACS command on "Overview of ACS commands and processing sequence").

Each entry in the alias catalog consists of the alias, the corresponding actual file name, a RANGE indicator for the scope, and some special attributes. Both user entries and system entries may be contained in the alias catalog of a task.

The RANGE operand in the ADD-ALIAS-CATALOG-ENTRY command determines the scope of alias substitution.

The following settings are possible:

*STD	Alias substitution is performed with the value for the ACS option STANDARD RANGE (default is *BOTH) which is valid for the task.
*FILE	Alias substitution applies only for files.
*JV	Alias substitution applies only for JVs.
*BOTH	Default: Alias substitution applies only for files and JVs.

As an entry in the alias catalog is defined precisely via the alias, there can only be precisely one entry for a particular alias. Any earlier definition for an alias is replaced by the new definition for the same name, even if the two definitions have different scopes.

The alias catalog, which is unique to each task, can only be supplied with entries by means of ACS commands. The catalog is created with the first LOAD-ALIAS-CATALOG-ENTRY or ADD-ALIAS-CATALOG-ENTRY command. It is deleted at the end of the task.

Alias catalog file (AC file)

The entries of an alias catalog can be stored in a file. Only the ACS administrator can store system entries in a file. Such a file is referred to as an alias catalog file (AC file). This file is a SAM file in which the contained entries are represented internally.

The entries stored in an alias catalog file can be reloaded into the alias catalog with appropriate commands. Nonprivileged users can only store the user entries of his or her catalog. Loading alias catalog entries from an alias catalog file essentially recreates user entries.

User entries

These can be created independently by users.

The user entries are however, subject to specific restrictions with respect to the conventions for defining alias names. They are essentially a means of implementing individual file name substitutions.

System entries

System entries can only be created by a ACS administrator. Users can generate them only by loading them from a privileged AC system file. System entries are stored in an AC system file The AC system file is an alias catalog file created by the ACS administrator and users can load catalog entries from it into their own alias catalogs. For data security reasons, users must address the AC system file via a special ID that is defined by the ACS administrator.

The system entries describe the system software configuration (compiler, linkage editor, utilities, etc.) predefined by the ACS administrator. System entries cannot be modified by users or stored in an alias catalog file.

Notes on version dependency

- AC files which were created in earlier versions can be called in the current version of BS2000 without any problem.
- AC files which were created in the current version of BS2000 can also be called in earlier versions.

Constructing aliases

- An alias is an alternative file/JV name that can be used instead of the actual name. Aliases have the same syntax as normal file/JV names (see “Rules for the formation offile names” (Fully qualified file name)).
- An alias may include a catalog ID and/or user ID, provided this is allowed by the ACS administrator.
- Aliases may not be partially qualified and may not be defined using wildcards.
- Generation and version specifications are illegal both in aliases and the associated actual file names.
- Alias definitions containing the user ID TSOS or SYS* are not permitted. "$" names, however, are permitted ($ and $. are essentially distinct from $TSOS, even if DEFLUID=$TSOS, so $EDT and $.EDT are allowed as aliases!). The names “$MYUSERID.FILE.123” and “FILE.123” are considered distinct even if MYUSERID is the user ID of the current task.
- Any specification of the default user ID (file names starting with “$” and without a period) is interpreted as equivalent with names starting with “$.”
- The names of temporary files/JVs (“#”) cannot be defined as an alias, and no alias may be defined for a temporary file/JV. The identifier (first character) for temporary files/JVs cannot be used for aliases.
- If a file name is specified with the generation number in a macro or command, the generation entry is separated from the name before the alias catalog is searched. It is appended to the associated actual file name if an appropriate alias definition is found. If the resulting name is too long (i.e. > 54 characters after completion), it is treated as illegal and rejected.
- Names of temporary system files (“S.” files) are not converted.

ACS does not restrict the effectiveness of **file protection** in BS2000 at all, since the protection attributes of a file (passwords, BASIC-ACL, GUARDS, etc.) are only checked by BS2000 after the TSOSCAT entry has been read via a complete path name. The ACS call is executed before this occurs, and no further alias substitution takes place between the authorization checks and the actual processing of the file.

Restrictions on the use of aliases

- Alias definitions are local to each task and are only effective during the execution of that task. They are not stored in the catalog entry of the file/JV, which means that a file/JV can be accessed via any number of different aliases, and the same alias can be used by various tasks to identify different files/JVs. No aliases can therefore be defined for file names which are passed from one task to another.
- In order to prevent inconsistencies, aliases for task families which use the same program libraries must be defined by means of LOGON procedures.
- The same name should always be used with all calls for a file/JV. If a substituted and completed file name is used after the SHOW-FILE-ATTRIBUTES command, inconsistencies may occur. The completed file name may be found as an alias for another file/JV in the alias catalog and thus cause some other file/JV to be addressed. To prevent inadvertent duplicate substitutions, only file names without catalog IDs should be defined as aliases by the user or ACS administrator.
- If an alias catalog has been created for a task or if file name completion with a prefix is defined, any subsequent comparison between specified names and file names returned by the SHOW-FILE-LINK command or the RDTFT macro or supplied by OPEN in the TU FCB can be expected to contain discrepancies with respect to files/JVs for which alias addressing was not explicitly restricted.
- As far as possible, an alias should be defined without a user ID or be defined completely with both a catalog ID and a user ID. The ACS administrator can globally prohibit the use of a catalog ID/user ID in aliases for all users (but cannot control this selectively for individual users). In addition, duplicate substitutions are more likely to occur if complete path names are used as aliases.
- The interrogation of information in the user's own alias catalog can also be completely blocked by the ACS administrator. If this is done, commands involving the catalog will only be executed if the task has ACS administrator privileges. The commands affected in this case are ADD-ALIAS-CATALOG-ENTRY, MODIFY-ALIAS-CATALOG-ENTRY, and LOAD-ALIAS-CATALOG FROM-FILE=...
- Before deleting a file/JV which can be addressed by using aliases or which is supplemented with a prefix, DELETE-FILE or DELETE-JV generally displays the actual name of that file/JV in a message prompt unless CHECK=NO has been specified.
- REPAIR-DISK-FILE: No prefix insertion and no ACS substitution occurs when the error file S.<filename1>. REPAIR is generated.
- Catalog IDs for which an RFA connection exists are not allowed in alias catalog entries and prefixes. The insertion of a prefix into a file name intended for a remote job (RFA) is never permitted.
- Aliases which correspond to the structure of SERSLOG files cannot be used for SERSLOG files.
- If the file associated with a specified alias is not available for ELFE, SERSLOG, etc., the alias is output in the corresponding error message.

- The following **must not** be addressed via aliases:
  - alias catalog files (ACFs and ACSFs) in the commands STORE-ALIAS-CATALOG and LOAD-ALIAS-CATALOG
  - USER-RESOURCES-FILEs in the PRINT-DOCUMENT command
  - all DSSM files (subsystem libraries, REP files, syntax files, DSSM catalogs,...)
  - UDS database files (catalog, realm, AFIM file, SLF file, status file, backout file, tempo file, hashlib)
  - files processed by TCP-IP
  - SATUT files for selected information
  - SESAM database files (ZD file, ORG file, catalog file, logging and cursor files)
  - files that are to be processed with LEASY or by File Transfer
  - files used at startup

5.5.4 Prefix insertion

ACS can be used to define a prefix which precedes the name for all file/JV names which are not recorded in the alias catalog.

This allows a user to create a separate “subcatalog” under his or her user ID and to create, process, and delete files /JVs in it independent of the other files which belong to the same user ID but are not part of the subcatalog.

A subcatalog of this type is identified by a partially qualified file name (the last character must be a period). This partially qualified name is inserted as a prefix for the file names entered with all commands and macros, provided these file names do not contain a user ID and have not been recorded in alias catalog entries.

One of the typical applications of the prefix function is when several users share the same BS2000 user ID. By using an appropriate prefix rule (e.g. prefix=name/project), each user can work in a separate subcatalog and make use of commands and programs for file processing without having to manually insert the prefix before each file name. The same procedures with “fixed” file names can also be used without changes on any subcatalog.

The rules for prefix insertion are defined as follows:

If the file/JV name has been entered as an alias in the alias catalog of the task, no prefix is inserted. In other words, such files/JVs can still be accessed without a user ID (e.g. `START-PROGRAM FROM-FILE=EDT`), but no prefix will be inserted.

No prefix is inserted if the file/JV name contains a user ID which differs from that of the current task.

If the file/JV name contains the user's own user ID, the prefix will be inserted only if it consists of just a catalog ID and if the file name itself does not contain a catalog ID.

A single catalog ID can be defined as a prefix (in the form “:catid:”). It is then inserted before all file/JV names associated with the user's own user ID in the current task. This is effectively equivalent to a temporary, task-local change to the default catalog ID. Only catalog IDs on the system-local (pubsets of the home system) may be specified; RFA is not supported by ACS.

If the file name and the prefix contain a catalog ID, the prefix is not inserted.

File names with the user ID TSOS are never supplemented with a prefix.

If the file/JV name contains a catalog ID (but not a user ID), the prefix is inserted immediately after the catalog ID.

If the file/JV name becomes too long (i.e. > 54 characters) as a result of inserting the prefix, the file name is considered invalid and rejected.

An option can be used to control whether a prefix is to be inserted even if a file/JV name already begins with the same string that is defined as the prefix.

Restrictions when inserting a prefix

When a prefix for file names is defined for a task, the prefix is only inserted before file names which are specified without a user ID and not recorded in the alias catalog.

> ! Warning in the case of system files!
> `START-EXECUTABLE-PROGRAM FROM-FILE=EDT` is converted to: `START-EXECUTABLE-PROGRAM FROM-FILE=<prefix.>EDT`

Any file/JV name that is to be supplied with a prefix must not be contained as an alias in the alias catalog of the task.

The inserted prefix is a component of the name under which the file/JV is cataloged in the TSOSCAT file catalog. It must therefore be defined with ACS or explicitly specified as a component of the file/JV name whenever the file is subsequently accessed.

The specified file/JV name must comply with naming conventions even if another prefix is added later.

If DAMP files are created under the user's own user ID and prefix insertion is used, all internal files of DAMP are also addressed with the preceding prefix.

5.6 File lock management

The SHOW-FILE-LOCKS command displays the locks that are currently active for a file. File locks may be set as a result of the following states/operations:

- The file is currently open.
- The file has been explicitly reserved (SECURE-RESOURCE-ALLOCATION command).
- A print lock has been defined for the file (with the PRINT-DOCUMENT ...,LOCK-FILE=*YES command, the PRNT macro or the MODIFY-PRINT-JOB-ATTRIBUTES command).
- The file has been reserved for file transfer (TRANSFER-FILE command).
- A SYSLST file is waiting to be printed following an EXIT-JOB command.
- The file is currently being processed by a CCOPY job.
- The file has been reserved for an EAM access.
- A temporary connection failure in a computer network or a system error on the local system has led to “unauthorized file locks” (see below).

The SHOW-FILE-LOCKS command does not display any locks set by catalog management.

SHOW-FILE-LOCKS enables the file owner and systems support to diagnose the reason for a processing problem caused by an active file lock. The command may be entered by the file owner, other users authorized to access the file and under any user ID with the TSOS privilege.

Resetting unauthorized file locks

So-called unauthorized file locks may prevent your (continued) work with a file. Such locks may be caused, for example, by a temporary connection failure in a computer network or a system error on the local system that prevents the resetting of file locks.

The REPAIR-FILE-LOCKS command makes it possible for file owners and systems support to reset unauthorized file locks.

5.7 PFA: performant file access

The performance-related file attributes PERFORMANCE, USAGE and DISK-WRITE were introduced in conjunction with the hiperfile concept:

- PERFORMANCE Specifies the desired performance of I/O behavior for file access
- USAGE Indicates whether the performance is requested for read access only, for write access only or for both types of access
- DISK-WRITE Determines at which point during high-performance file processing data consistency for write operations is required.

PFA with SF pubsets

With SF pubsets, the performance-related file attributes control cache utilization: any file with higher performance requirements is automatically processed with caching. In this case, the USAGE and DISK-WRITE specifications and the cache type (VOLATILITY) assigned to the pubset determine whether read and/or write caching are performed.

If the file attributes PERFORMANCE=*HIGH, USAGE= *READ-WRITE and DISK-WRITE=*IMMEDIATE are selected for a file stored on an SF pubset with a non-secure cache (e.g. main memory cache), then only read caching is performed since the conditions for secure input/output write caching are not fulfilled. It is the user's task to ensure that the file attributes he/she specifies are compatible with the cache characteristics.

The output line CACHE-MEDIUM supplied by the command SHOW-MASTER-CATALOG-ENTRY INF=*USER indicates whether the pubset has been assigned a nonvolatile cache medium (CACHE-MEDIUM =NONVOLATILE). Only if this is the case do the file attributes USAGE=*WRITE and DISK-WRITE=*IMMEDIATE take effect.

i From BS2000 OSD/BC V11.0 onwards, non-volatile cache media are no longer supported.

For SF pubsets that have not been assigned a cache, the attributes PERFORMANCE, USAGE and DISK-WRITE are irrelevant. The performance behavior is exclusively determined by the location of the file.

These examples show that the user requires very detailed knowledge of the characteristics of the various media to be able to obtain the desired performance behavior.

PFA with SM pubsets

With SM pubsets, the performance-related file attributes are used to select the volume set that is best suitable as location of a particular file: When a new file is created, its performance attributes are used automatically to determine the volume set whose performance profile is best suited to meet the file requirements.

The performance profile of a volume set can be displayed by means of the SHOW-PUBSET-DEFINITION-FILE command. It is derived from the performance range of a volume set and the constraints on secure writing as a result of higher performance values. The complexity is a result of the fact that the performance profile has to characterize different cache types and disk types. The actual performance-relevant attributes of a volume set are based on the following considerations:

For instance, a volume set equipped with a cache offers standard performance (STD) for file processing without cache, and higher performance (HIGH or VERY-HIGH) for file processing with cache. This means that the performance range offered by a volume set equipped with a cache extends from STD to HIGH up to VERY-HIGH.

i From BS2000 OSD/BC V11.0 onwards, non-volatile cache media are no longer supported.

The SHOW-PUBSET-DEFINITION-FILE command supplies a list of performance values for volume sets. Note, however, that with nonsecure (volatile) caches, higher performance can only be offered for file processing that does not require data consistency after each write operation. This constraint with respect to higher performance is taken into account by the performance-relevant attribute WRITE-CONSISTENCY:

WRITE-CONSISTENCY=*BY-CLOSE indicates that higher performance can be offered only at the expense of secure writing.

Systems support has to ensure that the hardware or cache configurations that are appropriate for the performance profile of a volume set are made available (using the MODIFY-PUBSET-CACHE-ATTRIBUTES command).

5.7.1 The hiperfile/PFA concept

The “hiperfile concept” encompasses numerous system software and hardware enhancements in BS2000 that are designed to accelerate file access and prevent a possible I/O bottleneck by “caching” files. High-speed semiconductor memory with short access times is used as a buffer (or cache) that enables adjustments to be made between the time required to access main memory and the (slower) access to external hard disks.

The HIPERFILE concept in BS2000 provides caching both via the command interface of the associated subsystem and via a uniform command interface that is integrated into DMS:

- ADM-PFA (Administrator Performant File Access): Caching in the medium main memory via privileged DAB commands
- PFA (User Performant File Access): Caching by embedding the hiperfiles in DMS

ADM-PFA caching designates the use of the concepts, methods and commands of the DAB cache handling system (START-DAB-CACHING) in order to differentiate this terminologically from the embedding of hiperfiles in DMS, i.e. the so-called PFA concept. ADM-PFA caching is described in the “DAB” manual [5 (Related publications)].

Within the PFA concept, BS2000 supports main memory storage as a cache medium. The driver software necessary to operate this medium has to be the DAB subsystem as cache handler:

Embedding of hiperfiles in DMS is achieved by enabling each pubset to be assigned a cache medium, and by allowing the user to declare his or her files as hiperfiles using appropriate attributes for high-performance processing.

5.7.2 Selecting pubsets

The SHOW-MASTER-CATALOG-ENTRY command (INFORMATION=*USER operand) can be used by the user to obtain information about the currently applicable and activated cache assignment (no cache or cache) for imported SF pubsets.

The SHOW-PUBSET-DEFINITION-FILE command provides information about the performance profile of volume sets of SM pubsets. In addition, the SHOW-PUBSET-FILE-SERVICES command enables the user to verify whether the pubset offers the services appropriate for the selected file attributes.

The use of hiperfiles is subject to the condition that systems support has granted the user the required authorization (DMS-TUNING-RESOURCES) and, for SM pubsets, that the user contingent (HIGH-PERF-SPACE or VERY-HIGH-PERF-SPACE) has been assigned a value greater than zero. Information on these settings is supplied by the SHOW-USER-ATTRIBUTES command.

The following caching authorizations can be assigned to the individual user by systems support:

- NONE: the user does not receive any authorization to cache files.
- CONCURRENT-USE: the user is authorized to cache files, but only in competition with all other users that have the same authorization. This means that if memory in the cache area is scarce, parts of the user file may be swapped to disk.
- EXCLUSIVE-USE: the user is authorized to make exclusive use of a cache for file processing. In this case, even if memory is scarce, the system attempts to maintain the user's file entirely in the cache area at exactly the point when the user requests this by means of an appropriate file attribute.

The effects of these authorizations and the interaction between them and the user-specified file attributes are explained in the form of a table immediately after the next section “Selecting files”.

5.7.3 Selecting files

If systems support has granted a user the right to cache his or her files, the user can identify his or her I/O-critical files by means of appropriate file attributes. Files to be identified in this way should be selected with the specific aim of reducing I/Os, i.e. the files selected should be those which exhibit an above average share of DMS I/Os (the SM2 monitoring system can be used as a decision-making tool here).

Users can assign the attributes PERFORMANCE, USAGE and DISK-WRITE to their selected files by means of the commands CREATE-FILE, MODIFY-FILE-ATTRIBUTES, and ADD-FILE-LINK, or the FILE and CATAL macros.

The PERFORMANCE file attribute describes the weight (i.e. importance) of the required improvement in I/O performance and can be assigned the following values:

- STD
  The file is to be processed without caching.
- HIGH
  The file is to be processed using a cache, but may be partially migrated if memory is scarce. This specification will only be effective if CONCURRENT-USE or EXCLUSIVE-USE authorizations have been entered for the user ID in the user catalog of the pubset.
- VERY-HIGH
  The file is to be permanently maintained in the cache even if memory is scarce. This specification is only effective if EXCLUSIVE-USE authorization has been entered for the user ID in the user catalog of the pubset.
- USER-MAXIMUM
  The file is to be given the highest I/O attribute that is entered for the user in the user catalog.

The file attribute USAGE indicates the caching mode and can be specified with the following values:

- READ
  The enhanced performance requirements are for read operations only (read cache).
- WRITE
  The enhanced performance requirements are for WRITE operations only (write cache).
- READ-WRITE
  The requirements are for both READ and WRITE operations (read-write cache).

The file attribute DISK-WRITE specifies the point in time at which data must be in a consistent state following a write operation. This operand allows the user to implicitly indicate that his or her files are to be processed in a secure, nonvolatile cache medium.

- IMMEDIATE
  The data in the file must be in a consistent state as soon as a write operation is completed, which implicitly means that the file should not be processed via a volatile write cache. The data is to be written back to a secure, nonvolatile medium. Immediate data consistency after every write operation should therefore be requested for files which contain important data.

  i From BS2000 OSD/BC V11.0 onwards, non-volatile cache media are no longer supported.

- BY-CLOSE
  Data consistency for the file is not required until after CLOSE processing; the user forgoes processing in a nonvolatile storage medium.
  It should be noted in this case that when a file is processed via a volatile write cache, the data contained in the file will not be in a consistent state until after CLOSE processing. System errors that occur during the processing phase could thus lead to inconsistencies.
  Therefore only files which are relevant to the current application only should be assigned this attribute (e.g. compilation result lists).

The user can assign the required file attributes either statically or dynamically. File attributes are assigned statically via the CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands or the CATAL and FILE macros. Only users with the file owner's access rights in the user catalog can create the appropriate catalog entry. If the required authorization is not available, the file is assigned the maximum performance attribute that is possible for the user ID. Another option available to the user is to disable the attribute stored in the catalog entry for special file processing without modifying the catalog entry itself (dynamic assignment of file attributes). The weight of the PERFORMANCE and USAGE file attributes can be reduced by the ADD-FILE-LINK command or via the FCB interface. Here the file attributes stored in the file catalog are left unchanged.

The current performance attributes are evaluated by DMS when opening the file. Specifications in the catalog entry, dynamically defined values, and the authorization in the user catalog are all compared at this time. Simultaneously a further check is made to ensure that the user's specifications are compatible with the applicable cache assignment of the pubset in question. If all values are compatible, the file is processed in the cache; otherwise, no cache is used. Inappropriate authorization or incompatible configurations do not result in the OPEN being rejected.

The following table shows the interactions between systems support specifications for the user ID in the user catalog and those of the user for his/her files:

Cached files	**Authorization in the data pubset**	**Performance attributes permitted to the user on this data pubset**
BY-USER-SELECTED	NONE	None
	CONCURRENT-USE	File attribute PERFORMANCE=HIGH
	EXCLUSIVE-USE	File attribute PERFORMANCE=HIGH/VERY-HIGH
ALL / BY-SYSTEM	NONE	File attribute PERFORMANCE=HIGH

The START-FILE-CACHING command can be used to start caching for files that are already open. The PERFORMANCE and USAGE operands are used to determine the file's performance level and cache mode characteristics. The preconditions for the activation of caching for files that are already open are:

- the caller must possess access authorization to the file (file owner or systems support)
- the pubset on which the file is located must
  - possess a valid, activated cache allocation
  - be locally accessible
- the command must be issued on the same system if the cache assigned to the pubset is operated on the local system
- the necessary cache authorization must have been granted to the caller by systems support

If the file is already open or is already being processed with caching then the START-FILE-CACHING command is rejected.

The STOP-FILE-CACHING command terminates caching for an open file or a file for which data is still present in the cache. Data that is still present in the cache is written back (exception: read-only cache) and the cache contents are invalidated. The same conditions as for START-FILE-CACHING apply to command execution.

5.7.4 Errors when closing a hiperfile

The term “hiperfile”, as used below, indicates a file residing on a PFA pubset and possessing the valid performance attribute HIGH or VERY-HIGH, or a file residing on a PFA pubset that possesses the cache attribute CACHED-FILES=*ALL.

When a hiperfile is closed (CLOSE processing), all of the file blocks written by the application which have not yet been saved on the appropriate disks of the pubset by the cache handler or a cache controller are written explicitly to disk.

If an error occurs during this save operation because a disk device being addressed is defective, CLOSE processing returns the following error code:

```
DMS0E27  Error when closing file. I/O operation terminated due to hardware
         error.
```

However, the data is still readable, the cache is not cleared, and the data still in the cache can be transferred to the disk without outside intervention once the disk has been repaired. The file can normally be opened again and processed even though data in the cache could not be written back to the disk. Such files can be displayed with the command: SHOW-FILE-ATTRIBUTES ...,STATUS=*PAR(CACHE-NOT-SAVED=*YES).

The operating system flags a file which does not have a consistent image on disk due to the fact that a cache is being used as a “file with data in the cache”. These files (hiperfiles) can be displayed with the following command: SHOW-FILE-ATTRIBUTES ..., STATUS=*PAR(CACHED=*YES)

This list includes files that are currently being cached (including those with read-only cache) and which are currently open.

The user can exploit the fact that blocks which could not be written to disk successfully can still be read in cache in the following ways:

A file can be reopened at any time. A file can be “rescued” by writing it to a different volume with the COPY-FILE command. In order to do this, a destination file must be created with an explicit CREATE-FILE command before the COPY-FILE command is executed.

A DELETE-FILE command can be used to logically delete such a file. This invalidates any data belonging to the file which is still located in the cache.

If, even after servicing by a service technician, it is not possible to save the data remaining in the cache to disk, the cache can be destroyed without saving the data by means of the FORCE-DESTROY-CACHE command.

The hiperfiles of the pubset which have been flagged as being a “file with data in the cache” are marked in the catalog during the next IMPORT-PUBSET in such a way that they can no longer be opened (owing to possible data inconsistency). A prerequisite for this is, however, that the file has been assigned the performance attribute DISK-WRITE=*IMMEDIATE. The user can list the set of hiperfiles which can no longer be opened due to possible data inconsistency with the following command:
SHOW-FILE-ATTRIBUTES STATUS=*PAR(OPEN-ALLOWED=*NO)

These files should be deleted. A REPAIR-DISK-FILES command (only possible with the TSOS privilege) resets the OPEN-ALLOWED=NO attribute and it is again possible to open the file, but it is impossible to obtain information about data consistency. If a disk defect cannot be eliminated, the entire pubset may have to be reinitialized.

5.7.5 Examples

The following two examples, one for SF pubsets and one for SM pubsets, are intended to summarize the typical execution sequence from the user's point of view.

Example 1: SF pubset

The imported data pubset DAT1 (SF pubset) is buffered in the main memory.

```
/SHOW-MASTER-CATALOG-ENTRY CATALOG-ID=DAT1, INFORMATION=*USER ─────────  (1)
PUBSET DAT1: SINGLE-FEATURE, PUBRES-UNIT=FEE8, LOCAL-IMPORTED, NK-FORMAT
             EXTRA-LARGE-CATALOG
---- CURRENT PUBSET-PARAMETERS      ----------------------------------------
 MAXIMAL I/O LENGTH                   | 240 HP
 ALLOCATION UNIT SIZE                 | 3   HP
 PHYSICAL ALLOCATION                  | BY ADMINISTRATOR
 SPEEDCAT MODE                        | SCA RUNNING
---- CURRENT CACHE-CONFIGURATION    ----------------------------------------
 CACHE MEDIUM                         | VOLATILE
 CACHE SIZE                           | 80 MB
 DOUBLE BUFFERING                     | NO
/SHOW-USER-ATTRIBUTES USER2, PUBSET=DAT1,INFORMATION=*PUBSET-ATTRIBUTES ──(2)
SHOW-USER-ATTRIBUTES --- PUBSET DAT1 - USER USER2         2017-03-20 12:56:15
-----------------------------------------------------------------------------
USER-ID                    USER2           PUBLIC-SPACE-EXCESS     *NOT-ALLOWED
CODED-CHARACTER-SET       EDF03IRV        DMS-TUNING-RESOURCES    *CONCURRENT
NET-CODED-CHAR-SET        *ISO            PHYSICAL-ALLOCATION     *NOT-ALLOWED
                                        NET-STORAGE-USAGE       *ALLOWED
FILE-NUMBER-LIMIT           16777215      JV-NUMBER-LIMIT            16777215
LIMITED FILES                     27      JOB-VARIABLES                     0
PERM-SPACE-LIMIT            16777215      TEMP-SPACE-LIMIT         2147483647
PERM-SPACE-USED               424572      TEMP-SPACE-USED               12000
-----------------------------------------------------------------------------
SHOW-USER-ATTRIBUTES           END OF DISPLAY FOR USER USER2  ON PUBSET DAT1
/SHOW-FILE-ATTRIBUTES FILE-NAME=:DAT1:USERFILE ────────────────────────  (3)
0000012000 :DAT1:$USER2.USERFILE
  -------------------------------- ORGANIZATION ------------------------------
  FILE-STRUC = NONE        BUF-LEN    = NONE        BLK-CONTR  = NONE
  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
  REC-FORM   = NONE        REC-SIZE   = 0
  AVAIL      = *STD
  WORK-FILE = *NO          F-PREFORM = *NK2         S0-MIGR = *ALLOWED
:DAT1: PUBLIC:      1 FILE  RES=     12000 FRE=     12000 REL=    12000 PAGES
/MODIFY-FILE-ATTRIBUTES FILE-NAME=:DAT1:USERFILE,
     IO-ATTRIBUTES=*PARAMETERS(PERFORMANCE=*HIGH,
     USAGE=*READ),DISK-WRITE=*IMMEDIATE  ──────────────────────────────── (4)
```

(1) The command SHOW-MASTER-CATALOG-ENTRY enables the user to retrieve information on the cache activated for pubset DAT1, indicating whether a cache is active. In this example that is the case.

(2) The SHOW-USER-ATTRIBUTES command allows the user to check his or her currently applicable pubset-specific authorization to process files using a cache. The displayed authorization indicates the highest permissible authorization (CONCURRENT(-USE)) for the user.

(3) The command SHOW-FILE-ATTRIBUTES enables the user to check which file attributes he or she has already assigned to the USERFILE.

(4) The user assigns the USERFILE the following performance-related file attributes:

- The file is to be processed via a cache (PERFORMANCE=*HIGH). This specification will take effect, since the authorization CONCURRENT(-USE) has been entered for the user ID in the user catalog.
- The request for high-performance file processing is to apply to read operations only (USAGE=*READ).
- If the file is opened in write mode, the cache will not be used, because DISK-WRITE=*IMMEDIATE requires an always consistent write processing of the file.

Example 2: SM pubset

The volume set VS01 of the imported data pubset SMS1 (SM pubset) is buffered in the main memory.

```
/SHOW-PUBSET-DEFINITION-FILE PUBSET=SMS1,VOLUME-SET=VS01 ——————————————  (1)
 -------------------------------------------------------------------------
 COMMAND: SHOW-PUBSET-DEFINITION-FILE
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 PUBSET SMS1: TYPE=SYSTEM-MANAGED, VOLUMESETS=3, DEFAULT FILE FORMAT=NK2
 ---- VOLUME-SET INFORMATION --------- + ----------------------------------
 VOLUME-SET VS01: NORMAL-USE, NK2-FORMAT
---- GLOBAL ATTRIBUTES      --------- + -----------------------------------
 AVAILABILITY                        | STANDARD
 USAGE                               | WORK
 FORMAT                              | NK2-FORMAT
 MAXIMAL I/O LENGTH                  | 240 HP
 ALLOCATION UNIT SIZE                | 3   HP
 DRV-VOLSET                          | NO
 NEW FILE ALLOCATION                 | NOT RESTRICTED
 VOLUME SET ACCESS                   | NOT RESTRICTED
---- PERFORMANCE ATTRIBUTES --------- + -----------------------------------
 PERFORMANCE                         | STANDARD
 WRITE-CONSISTENCY                   | IMMEDIATE
```

(1) The SHOW-PUBSET-DEFINITION-FILE command enables the user to retrieve information about the performance profile of volume set VS01 of pubset SMS1.

```
/SHOW-USER-ATTRIBUTES USER2,PUBSET=SMS1,INFORMATION=*PUBSET-ATTRIBUTES ———(2)
SHOW-USER-ATTRIBUTES --- PUBSET SMS1 - USER  USER2        2017-03-21 11:53:55
-----------------------------------------------------------------------------
USER-ID                  USER2         PUBLIC-SPACE-EXCESS     *NOT-ALLOWED
CODED-CHARACTER-SET      EDF03IRV        DMS-TUNING-RESOURCES    *CONCURRENT
NET-CODED-CHAR-SET       *ISO            PHYSICAL-ALLOCATION     *NOT-ALLOWED
DEF-STORAGE-CLASS        *NONE           NET-STORAGE-USAGE       *ALLOWED
FILE-NUMBER-LIMIT          16777215      JV-NUMBER-LIMIT           16777215
LIMITED FILES                     1      JOB-VARIABLES                    0
                                   PERM-SPACE      TEMP-SPACE      WORK-SPACE
TOTAL-SPACE              LIMIT      2147483647      2147483647      2147483647
                         USED                0               0               3
S0-LEVEL-SPACE           LIMIT        16777215
                         USED                0
HIGH-PERF-SPACE          LIMIT        *MAXIMUM        *MAXIMUM        *MAXIMUM
                         USED                0               0               0
VERY-HIGH-PERF-SPACE     LIMIT        *MAXIMUM        *MAXIMUM        *MAXIMUM
                         USED                0               0               0
HIGH-AVAILABLE-SPACE     LIMIT               0
                         USED                0
-----------------------------------------------------------------------------
SHOW-USER-ATTRIBUTES          END OF DISPLAY FOR USER USER2    ON PUBSET SMS1
```

(2) The SHOW-USER-ATTRIBUTES command allows the user to check his or her currently applicable pubset-specific authorization to process files using a cache. The displayed authorization indicates the highest permissible authorization (CONCURRENT(-USE)) for the user. At the same time, the command indicates that the contingents for HIGH-PERF-SPACE and VERY-HIGH-PERF-SPACE have a value greater zero and that the user has not yet exceeded the contingents assigned to him or her.

```
/SHOW-PUBSET-FILE-SERVICES PUBSET=SMS1, SELECT=*BY-ATTRIBUTES(FILE-
ATTRIBUTES=*PARAMETERS( IO-
ATTRIBUTES=*PARAMETERS(PERFORMANCE=*HIGH,USAGE=*READ-WRITE), DISK-WRITE=*BY-
CLOSE))————————————————————————————————————————————————————————————————  (3)
 WORK-F  AVAIL F-FORM   IO(PERF)  IO(USAGE) DISK-WRITE   SUPPORT-QUALITY
------+------+------+----------+----------+----------+-------------------
  YES    STD    NK2        HIGH READ-WRITE   BY-CLOSE          OPTIMAL
/CREATE-FILE FILE-NAME=:SMS1:USERFILE, IO-
ATTRIBUTES=*PARAMETERS(PERFORMANCE=*HIGH, USAGE=*READ-WRITE),DISK-WRITE=*BY-
CLOSE ——————————————————————————————————————————————————————————————————  (4)
```

(3) The SHOW-PUBSET-FILE-SERVICES command enables the user to verify whether pubset SMS1 offers the file service requested.

(4) The user assigns the USERFILE the following performance-related file attributes:

- The file is to be processed via a cache (PERFORMANCE=*HIGH). This specification will take effect, since the authorization CONCURRENT(-USE) has been entered for the user ID in the user catalog.
- The request for high-performance file processing is to apply to both read and write operations (USAGE=*READ-WRITE).
- By specifying DISK-WRITE=*BY-CLOSE, the user decides that the data are only stored in a consistent state at the time the file is closed. Therefore, the file can be processed in a cache in the main memory. Only files with easily restorable data should be given this attribute (e.g. a list file during a compilation), because a system error can lead to inconsistencies of these files.

5.7.6 The HIPERBATCH concept

HIPERBATCH (high performance batch processing) is the term given to the use of a special new variant of PFA caching.

Batch processing often involves a series of processing steps for individual files. For example, in one processing step a (temporary) file is created which in a subsequent step is read again as an input file and used as such. Between two such processing steps, the file is usually closed and reopened.

When a file buffered with PFA is closed

- in the case of a write cache, the data belonging to the file in the cache is written back to the disk
- the cache management data of this file is released by the software cache handler DAB (i.e. cache in the main memory)
- the data belonging to this file which is still in the cache is invalidated.

As long as there is no follow-up processing for this file, this is a highly effective approach. In a subsequent access, however, the file must first be paged back into the cache in a “transient phase” until the application profits from read hits. In the case of batch processing, there is no real need to write the data back to disk for reasons of data security, since the run can be repeated in the event of an error.

This is where the HIPERBATCH concept comes in. A new CLOSE parameter (ADD-FILE-LINK command or CLOSE macro) can be used to specify that at the time of closure the data in the cache should not be written back and – more importantly – not invalidated. A subsequent open operation on the same file can use the data in the cache immediately. The effect is a noticeable acceleration of batch processes with file follow-up processing steps.

The CLOSE parameter can be specified as follows:

- Command: ADD-FILE-LINK ...,CLOSE-MODE=*KEEP-DATA-IN-CACHE
- Macro: CLOSE <fcb>,*KEEP-DATA-IN-CACHE

6 File and data protection

This chapter describes the functions used by DMS to support file and data protection. File and data protection are implemented with the aid of various mechanisms:
protection against unauthorized reading, modification or destruction of an existing file by restricting the access rights and access mode, by defining a retention period, by assigning passwords, etc.
When files are deleted the data they contain can be overwritten with X'00'. This operation is known as physical deletion.
Protection against unauthorized access is guaranteed by default. If the user makes his/her file shareable by other users, he/she can limit their access rights, e.g. by means of passwords.

File encryption with a crypto password enables the contents of a file to be protected against unauthorized access – even against people with TSOS privilege. However, file encryption does not protect against deletion, overwriting or destruction of the file contents and cannot replace file protection and backup.

DMS guarantees comprehensive file protection for all volumes (in the case of tapes, provided they are equipped with standard labels). Access protection for pubsets takes effect as soon as the relevant access authorization is defined in the user catalog entry by systems support. Only users authorized to access a given pubset can access files on this pubset.
At the file level, the user can assign protection attributes.

Each access to a file (displaying the contents, processing with an editor, executing the commands in a procedure file,...) is implemented via one of the three access modes:

READ	read access
WRITE	write access
EXECUTE	execute access

The specified access mode may therefore be subject to the corresponding access protection mechanism.

6.1 Protection mechanisms

The following file protection mechanisms are available in BS2000:

- Restricting pubset access
  Protection against unauthorized access to files on other pubsets can be provided by assigning different user IDs to different pubsets.
- Standard access control (USER-ACCESS/SHARE and ACCESS), (see "Standard access control (USER-ACCESS/SHARE and ACCESS)")
- Basic Access Control List (BACL)
  Permanent files and file generations on shared volumes can be protected using an Access Control List, whereas tape files and temporary files cannot (see "Basic Access Control List, BACL").
- Definition of an access profile using GUARDS
  GUARDS is a component of the SECOS software product (see "GUARDS - file protection via a special access profile (guard)").
- Password
  Files can be protected by defining file-specific read, write and execute passwords. A password-protected file cannot be processed unless the appropriate password for the desired access is specified. Passwords can be encrypted, if desired (see "Assigning a file password").
- File encryption
  The content of files is encrypted (see "File encryption").
- Retention period
  The user can define a retention period for a file during which no modification to the file will be permitted (see "Specifying a retention period").

Notes

- A file protected with GUARDS can be accessed only when the defined access profile exists and the GUARDS subsystem is loaded. The SHOW-SUBSYSTEM-STATUS command supplies information on loaded subsystems.
- For files on Net-Storage and in particular for node files, with respect to their access protection it must be borne in mind that the mechanisms described here are only effective in BS2000. Please also observe the notes in section

6.1.1 Hierarchy of protection mechanisms

All files can be secured by one or more protection mechanisms.

If the ACCESS/USER-ACCESS, BACL and GUARDS protection mechanisms are all used for the same object then only one of them is effective in accordance with the following hierarchy:

1. Guards
   If object protection is defined by means of Guards then only the access conditions defined in the Guards apply. Any BACL defined for the object is ignored as are the ACCESS/USER-ACCESS protection attributes.
2. Basic Access Control List, BACL
   If Guards protection is not defined for an object but a BACL is, then the protection settings specified in the BACL apply. The ACCESS and USER-ACCESS protection attributes are ignored.
3. Standard access control (USER-ACCESS/SHARE and ACCESS)
   If an object is not protected by Guards or by a BACL then the ACCESS and USER-ACCESS protection attributes are employed as the protection mechanism.

Restricted pubset access, password protection, file encryption and the retention period apply additionally in all cases.

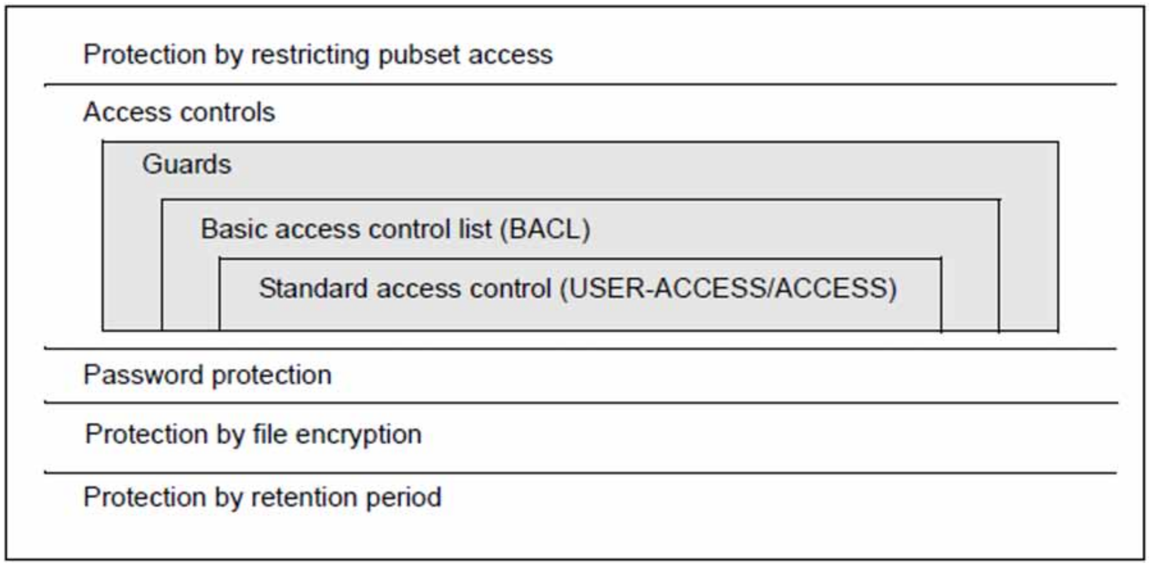

Protection by restricting pubset access
Access controls
Guards
Basic access control list (BACL)
Standard access control (USER-ACCESS/ACCESS)
Password protection
Protection by file encryption
Protection by retention period

Figure 6: Protection mechanisms for files in a pubset

6.1.2 Interdependency of the various protection mechanisms

Standard access control (USER-ACCESS/SHARE and ACCESS)

When a catalog entry is created (via the CATAL macro or CREATE-FILE command), file protection via standard access control is set by default. The specifications for standard access control are always entered into the file catalog regardless of whether or not a BACL has been activated or whether a GUARDS entry exists for the file. If a higher access protection level (using a BACL or GUARDS) is defined at the same time, the standard access control values will be entered into the file catalog, but ignored for the current access operation.

Basic Access Control List, BACL

Users who wish to use the basic access control list (BACL) for file protection must activate it themselves (by using the CREATE-FILE or MODIFY-FILE-ATTRIBUTES command or the CATAL macro). Specifications for the BACL are entered in the file catalog. If a higher protection level is defined at the same time as the BACL (GUARDS), the BACL values will be ignored for access purposes.

If an existing basic access control list (BACL) is deleted (command MODIFY-FILE-ATTRIBUTES; macro CATAL) and no higher access protection exists, any standard access control values specified previously or at the same time are activated again, i.e. standard access control is reinstated as the protection mechanism.

GUARDS

If a file has a GUARDS entry then access control is regulated by GUARDS alone.

Any existing BACL protection will be reinstated after GUARDS has been deactivated.

If no BACL protection is defined, the USER-ACCESS/ACCESS protection attributes are reinstated after GUARDS deactivation.

6.1.3 Protection mechanisms for disk and tape files

The following table provides an overview of the available protection mechanisms:

Object	**Protection mechanism**	1. Restricted pubset access 2. ACCESS, USER-ACCESS 3. BACL 4. Password 5. File encryption 6. Retention period 7. GUARDS						
		1	**2**	**3**	**4**	**5**	**6**	**7**
File	Public	+	+	+	+	+	+	+
	Temporary	-	-	-	-	+	-	-
	Private	-	+	+	+	-	+	-
	Tape	-	+	-	+	-	+	-
File generation group	Index public, FGen public	-	+	+	+	+	+	+
	Index public, FGen tape	-	+	+	+	+	+	+
	Index private, FGen private	-	+	+	+	-	+	-

Table 15: Possible uses of the protection mechanisms (+ =can be used, - =cannot be used)

The scope of the protection mechanisms for disk and tape files depends on the file type: the attributes for “standard access control”, “retention period”, “password” and “DESTROY option” apply either fully or only partially.

Disk files

The following tables show the permissible value ranges for the various operands in a macro and in a command.

Macro

File type	**Standard access control**		**BACL**	**GUARDS**	**RETPD**	**RDPASS, WRPASS, EXPASS**	**DESTROY**
	ACCESS	**SHARE**					
Permanent	Y	Y	Y	Y	Y	Y	Y
Temporary	WRITE	NO	N	N	Ignored	NONE	Y
Generation	Y	Y	Y	Y	Y	READ/ WRITE	Y

Table 16: Protection mechanisms for disk files (macro)

Command

File type	Standard access control		BACL	GUARDS	EXPIRATION-DATE	READ-, WRITE-, EXEC-PASSWORD	DESTROY
	ACCESS	SHARE					
File type	Y	Y	Y	Y	Y	Y	Y
Permanent	WRITE	OWNER-ONLY	N	N	Rejected	*NONE	Y
Temporary	Y	Y	Y	Y	Y	READ/ WRITE	Y

Table 17: Protection mechanisms for disk files (command)

Tape files

The following tables show the permissible value ranges for the various operands in a macro and in a command

Macro

File type	Standard access control		RETPD	RDPASS, WRPASS, EXPASS	DESTROY
	ACCESS	SHARE			
File type	Y	Y	Y	Y	Y
File type	Y	Y	Ignored	Y	Y
Permanent	Y	Y	Y	READ/ WRITE	Y

Table 18: Protection mechanisms for tape files (macro)

Command

File type	Standard access control		EXPIRATION-DATE	READ-, WRITE-, EXEC-PASSWORD	DESTROY
	ACCESS	SHARE			
File type	Y	Y	Y	Y	Y
File type	Y	Y	N	Y	Y
Permanent	Y	Y	Y	READ/ WRITE	Y

Table 19: Protection mechanisms for tape files (command)

Y:	the operand may be used with all possible operand values

N:	the operand is not permissible
other entries:	these are default values which cannot be changed

6.2 Defining protection attributes

If a file is cataloged by means of the CATAL macro or CREATE-FILE command without explicit protection attributes, then, by default, only the file owner can access this file.

The **file owner** is considered to be the user under whose user ID the file is cataloged, any co-owners of this file, if defined (see "Defining co-ownership (co-owners)") , as well as – subject to restrictions – systems support (i.e. user IDs with the TSOS privilege, see "Restrictions on TSOS co-ownership").

Unless otherwise specified, the permitted access mode is write access (ACCESS=WRITE). When a file generation is cataloged, DMS uses the protection attributes specified in the group entry (see "File generation groups (FGGs)" for further information on file generations).

Passwords and retention periods are not automatically defined by DMS; they can be assigned only by the owner. Write access is likewise not prohibited unless otherwise specified.

If a file is to be copied, COPY-FILE PROTECTION=*SAME or COPFILE PROTECT=*SAME can be specified; the copy is thus assigned the same protection attributes as the original file, i.e. the same passwords, retention period etc. but neither the file monitoring attribute (AUDIT) nor the locks against releasing storage space.

Only the owners of a file (see above for definition) may assign access rights or define, change or delete its protection attributes. In the case of disk files, the protection attributes may be redefined at any time by means of the CATAL macro or MODIFY-FILE-ATTRIBUTES command, provided that the file is not locked by some other access. The protection attributes of tape files can be changed only before the file is opened for the first time. That is because they are written into the tape labels when the file is processed and can then no longer be changed by means of CATAL.

Users can restrict the group of authorized co-users as well as the type of access (with standard access control, BACL or GUARDS) for each of their files.

Access authorization for files

Systems support can restrict access to a pubset: only those users who have been granted explicit access authorization by means of an entry in the user catalog (ADD-USER or MODIFY-USER-ATTRIBUTES command) may access files on the pubset.

A user can restrict access to particular user classes (OWNER, GROUP and OTHERS). User groups cannot be defined unless SECOS is being used.

If SECOS is not used, existing entries in the GROUP user class are ignored and only the entries in the OWNER and OTHERS classes are evaluated.

i *Note on the GROUP user class*

All users who are not assigned to any explicitly created group are automatically members of the implicitly created group *UNIVERSAL. This is especially true when no groups have been explicitly created. In this case, all system users are members of the same group. Consequently, when a BACL is evaluated, all users attempting access – with the exception of the object owner himself/herself – receive the access rights specified in the GROUP entry and not those defined in the OTHERS entry.

You are therefore very strongly recommended to assign identical access rights for the GROUP and OTHERS user classes when dealing with the *UNIVERSAL group.

6.2.1 Standard access control (USER-ACCESS/SHARE and ACCESS)

Access to a file can be made dependent on the access mode and the access rights.

The file owner can use the ACCESS operand of the CATAL macro and the commands CREATE-FILE, MODIFY-FILE-ATTRIBUTES, CREATE-FILE-GROUP and MODIFY-FILE-GROUP-ATTRIBUTES to define the type of file access to be permitted.

ACCESS=WRITE	Read and write access are permitted.
ACCESS=READ	Only read access is permitted.

With standard access control, execute access can only be restricted indirectly by assigning an EXECUTE password (see section “Access to password-protected files”).

The SHARE operand of the CATAL macro is used to define which user IDs may access the file using the access modes defined above:

SHARE=NO	Only the file owner can access the file.
SHARE=YES	All user IDs may access the file with the exception of user IDs which possess the privilege HARDWARE-MAINTENANCE but not the STD-PROCESSING privilege (see note).
SHARE=SPECIAL	All user IDs may access the file.

In the commands CREATE-FILE and MODIFY-FILE-ATTRIBUTES, the USER-ACCESS operand is used to define which user IDs may access the file using the access modes defined above:

USER-ACCESS=*OWNER-ONLY	Only the file owner can access the file.
USER-ACCESS=*ALL-USERS	All user IDs may access the file with the exception of user IDs which possess the privilege HARDWARE-MAINTENANCE but not the STD-PROCESSING privilege (see note).
USER-ACCESS=*SPECIAL	All user IDs may access the file.

i *Note on user IDs with the HARDWARE-MAINTENANCE privilege*

By default, these user IDs do **not** belong to the set of users designated by *ALL-USERS.
User IDs with the HARDWARE-MAINTENANCE privilege are used by the manufacturer for hardware maintenance purposes whereas all other user IDs are intended for the customer.

6.2.2 Basic Access Control List, BACL

Files can also be protected by means of the basic access control list (BACL).

Only permanent files and work files on disks can be protected with a BACL. BACL protection does not cover temporary files and tape files. When using a BACL, any desired combinations of access rights can be defined independently of each other for various user classes. Subsequently these combinations are entered in the BACL.

The BACL takes effect for an object if no Guards protection has been defined for it. Password protection and the retention period remain effective.

User classes

In connection with access protection via a basic access control list, a distinction is made between three classes of users:

OWNER:	The owner of an object, i.e. the user ID under which the file is cataloged, together with co-owners defined using the co-owner protection mechanism (see "Defining co-ownership (co-owners)").
GROUP:	All user IDs in the user group to which the owner belongs with the exception of the owner himself /herself and any co-owners; the group structure of the home pubset is checked.
OTHERS:	All other users with the exception of the co-owners.

Access rights

A BACL defines nine access authorizations for a file. Each of the user classes OWNER, GROUP and OTHERS can be separately assigned three access rights:

- Read (R): The file may be read and copied.
- Write (W): Data may be written to the file and the file may be overwritten. In contrast to the ACCESS attribute, BACL write access authorization does not automatically imply read access (data protection).
- Execute (X): The file (program, procedure) may be executed.

None of these access rights subsumes any of the others.

Each of the attributes R, W, X may be set or not set. They are independent of each other and only applicable to the specified access mode. In particular, “R” does not need to be set as a prerequisite for “X”. Specifying NO-ACCESS for all user classes revokes all access rights for the owner, the owner's user group and all other user IDs.

The following table shows which values must be entered in the basic access control list for a file in order to achieve the same effect as for the various values which can be specified for the SHARE and ACCESS operands of the CATAL macro or by means of the ACCESS and USER-ACCESS operands of the CREATE-FILE command.

Standard access control (CATAL macro)		**Basic control list (BACL)**		
ACCESS	**SHARE**	**OWNER**	**GROUP**	**OTHERS**
WRITE	NO	R W X	- - -	- - -
WRITE	YES	R W X	R W X	R W X

WRITE	SPECIAL	R W X	R W X	R W X
READ	NO	R - X	- - -	- - -
READ	YES	R - X	R - X	R - X
READ	SPECIAL	R - X	R - X	R - X

Table 20: BACL values in accordance with standard access control attributes in the CATAL macro

Standard access control (CREATE-FILE command)		**Basic control list (BACL)**		
ACCESS	**USER-ACCESS**	**OWNER**	**GROUP**	**OTHERS**
WRITE	OWNER-ONLY	R W X	- - -	- - -
WRITE	ALL-USERS	R W X	R W X	R W X
WRITE	SPECIAL	R W X	R W X	R W X
READ	OWNER-ONLY	R - X	- - -	- - -
READ	ALL-USERS	R - X	R - X	R - X
READ	SPECIAL	R - X	R - X	R - X

Table 21: BACL values in accordance with standard access control attributes in the CREATE-FILE command

Evaluation of the Basic Access Control List

1. If the user ID requesting access is the (co-) owner of the object or systems support then the access rights stored under OWNER apply.
2. If the user ID belongs to the owner's user group then the access rights stored under GROUP apply.
3. For all other user IDs, the access rights stored under OTHERS apply.

Example

OWNER	GROUP	OTHERS
R W X	R W -	R - -

The (co-)owner(s) of this file have read, write and execute access to the file. Members of the file owner's group have read and write access to the file. All other users have only read access to the file.

Notes

The software product SECOS must be used if all entries for all three user classes are to be evaluated (see the “SECOS” manual [8 (Related publications)]).

User groups cannot be defined unless SECOS is being used. If SECOS is not used, existing entries in the GROUP user class are ignored and only the entries in the OWNER and OTHERS classes are evaluated.

However, with an eye to the possibility of using SECOS later on, it is a good idea to make the same entries for the GROUP user class as for the OTHERS user class.

Activating BACL protection

BACL protection is activated when there is a BACL entry for at least one user class.

In a Basic Access Control List, the READ, WRITE, EXECUTE access types (abbreviated to R, W and X) can be assigned for user classes as follows:

The operands BASACL, OWNERAR, GROUPAR and OTHERAR of the CATAL macro are used to set BACL protection on the macro level. Any specification of an operand without an operand value is ignored.

On the command level, BACL is activated by the BASIC-ACL operand of the CREATE-FILE(-GROUP) and MODIFY-FILE(-GROUP)-ATTRIBUTES commands.

Deactivating BACL protection

Existing BACL protection can be canceled by means of an explicit specification (in the CATAL macro or MODIFY-FILE(-GROUP)-ATTRIBUTES command). If a higher protection level (GUARDS) is activated, the protection provided via the BACL is ignored.

6.2.3 GUARDS - file protection via a special access profile (guard)

Access protection is provided by means of special access profiles known as GUARDS. This file protection mechanism is effective only if the GUARDS (Generally Usable Access contRol aDministration System) function unit of the software product SECOS has been loaded (see also the “SECOS” manual [8 (Related publications)]).

Access to a file is controlled by a guard, i.e. a special access profile that contains all the conditions under which access to the file may be granted or denied (e.g. date, time, specific time period, user ID). Each such access profile is created by means of an appropriate GUARDS command or macro and is stored as a “guard entry” in the “guard catalog” under a “guard name” assigned by the user. Every pubset has a guard catalog that is maintained independently of user files.

As far as file processing is concerned, a distinction must be made between associating a file with a guard entry and accessing a file protected by such an entry. A file is linked with a guard entry by entering a guard name in the appropriate operand of the macro or command used (CATAL; CREATE-FILE(-GROUP) or MODIFY-FILE(-GROUP)-ATTRIBUTES). A file protected by a guard can only be accessed if the conditions defined in the guard entry are fulfilled.

Activating GUARDS protection

GUARDS protection is activated only if at least one access mode has been linked with a guard entry (the operand value *NONE for READ/WRITE/EXEC in a macro or command is also considered to be a guard entry in this sense). It is not necessary for the access profile to have been defined in the guard catalog at this point. Each of the three access modes (read, write, execute) can be protected by means of a separate guard entry.

i Unlike the ACCESS attribute, write access authorization with GUARDS does not imply read access authorization.

When GUARDS protection is activated for a file, all access modes which have not been explicitly defined are set to *NONE. The file **cannot** be accessed via these access modes.

It is only at the time of accessing a file protected with GUARDS that checks are performed to verify whether the specified guard entry exists (guard name), whether it may be used, and whether the access profile involved permits the user to access the file in the desired access mode.

Even co-owners of a guard-protected file have only those access rights defined in the guard entry.

i If GUARDS protection is entered for a file in the file catalog, but no access profile has been defined in the guard catalog for the specified guard name (e.g. if the access profile has been deleted), the file cannot be accessed. Files protected by GUARDS can only be accessed if the GUARDS subsystem is loaded

Deactivating GUARDS protection

Existing GUARDS protection can be removed only by means of an explicit specification (using the operand GUARDS=NONE in the CATAL macro or MODIFY-FILE(-GROUP)-ATTRIBUTES command).

6.2.4 Assigning a file password

A user can protect his/her files by passwords. Thereby it is possible to define a different password for each type of access. Passwords are assigned when creating or modifying the catalog entry (e.g. by means of the commands CREATE-FILE and MODIFY-FILE-ATTRIBUTES, operands READ-/WRITE-/EXEC-PASSWORD, or by means of the CATAL macro, operands RDPASS, WRPASS and EXPASS).

There are read, write and execute passwords. Knowledge of the write password also enables users to read and execute the file. Knowledge of the read password also enables users to execute the file. DMS permits access to a file only if the appropriate password is specified in the requesting job (by means of the ADD-PASSWORD command).

If write access is not permitted (due to the definition of a write password or access mode being restricted to read or execute access), DMS prevents the file from being overwritten, extended or erased. The file can be used only as an input file. The password governing write access must be added to the job's password table by means of the ADD-PASSWORD command (see "Access to password-protected files") before the catalog entry is modified.

Passwords are not shown in plaintext in trace listings or similar printed outputs!

Note

When a file protected with a read and/or execute password is converted into an encrypted file, the file loses its and/or execute password protection (see also section“File encryption”).

6.2.5 Specifying a retention period

The user can specify a retention period for a file; during this period, DMS prevents the file from being overwritten, extended or erased. The file can thus be used only as an input file during the retention period. A retention period can be specified or subsequently modified by means of the RETPD or EXDATE operand of the macros FILE and CATAL, or the EXPIRATION-DATE operand of the commands ADD-FILE-LINK and MODIFY-FILE-ATTRIBUTES.

6.2.6 Defining a deletion date

The FREE-FOR-DELETION attribute introduced for files on SF and SM pubsets (see the commands CREATE-FILE and MODIFY-FILE-ATTRIBUTES as well as the CATAL macro, DELDATE operand) enables the user to define a period after which his/her files may be deleted irrespective of access control.

Files which have reached this deletion date are detected using the selection criteria of SHOW-FILE-ATTRIBUTES and DELETE-FILE or FSTAT and ERASE. The user or systems support can create simple cleanup procedures for the deletion of any files that have reached their deletion date.
Cleanups can be performed either decentrally by the individual users, or centrally by systems support. Systems support can use centralized cleanups, for instance, in the event of memory bottlenecks to free storage space by deleting released files.

6.2.7 Examples of assigning protection attributes

Macros

A user with user ID GENERAL uses the CATAL macro to define protection attributes for a number of files in an Assembler program:

```
CATAL  FIL1,...,STATE=*UPDATE,SHARE=*NO,ACCESS=*READ ——————————————————  (1)
CATAL  FIL2,...,STATE=*UPDATE,SHARE=*YES,ACCESS=*READ —————————————————  (2)
CATAL  FIL3,...,STATE=*UPDATE,SHARE=*YES,WRPASS=C'007' ————————————————  (3)
CATAL  FIL4,...,STATE=*UPDATE,SHARE=*YES,RDPASS=C'0815',DELDATE='+150'   (4)
CATAL  FIL5,...,STATE=*UPDATE,SHARE=*YES,RDPASS=C'1111',EXPASS=C'2222',
       RETPD=20 ———————————————————————————————————————————————————————  (5)
CATAL  FIL6,...,STATE=*NEW,PROTECT=(*FROM-FILE,FIL5),DESTROY=*YES,
       BASACL=*STD ————————————————————————————————————————————————————  (6)
```

See below under “Commands” for an explanation of these program lines.

As an alternative, the FILE macro can be used to define or modify the protection attributes. The protection attributes are also evaluated by the selection criteria of FSTAT and ERASE.

Commands

A user with user ID GENERAL defines protection attributes for a number of files in a procedure or in a dialog:

```
/MODIFY-FILE-ATTRIBUTES FIL1,...,USER-ACCESS=*OWNER-ONLY,ACCESS=*READ —  (1)
/MODIFY-FILE-ATTRIBUTES FIL2,... USER-ACCESS=*ALL-USERS,ACCESS=*READ ——  (2)
/MODIFY-FILE-ATTRIBUTES FIL3,...,USER-ACCESS=*ALL-USERS,
         WRITE-PASSWORD=C'007'—————————————————————————————————————————  (3)
/MODIFY-FILE-ATTRIBUTES FIL4,...,USER-ACCESS=*ALL-USERS,
         READ-PASSWORD=C'0815',FREE-FOR-DELETION=+150——————————————————  (4)
/MODIFY-FILE-ATTRIBUTES FIL5,...,USER-ACCESS=*ALL-USERS,
         READ-PASSWORD=C'1111',EXECUTE-PASSWORD=C'2222',
         EXPIRATION-DATE=+20———————————————————————————————————————————  (5)
CREATE-FILE FIL6,...,PROTECTION-ATTR=*FROM-FILE(FILE-NAME=FIL5),
         DESTROY-BY-DELETE=*YES,BASIC-ACL=*STD—————————————————————————  (6)
```

As an alternative, the protection attributes can be defined using the commands CREATE-FILE and ADD-FILE-LINK and are also evaluated by the selection criteria of SHOW-FILE-ATTRIBUTES and DELETE-FILE.

(1) Only users working under the user ID GENERAL, the user ID of a co-owner or under TSOS (for restrictions see "Restrictions on TSOS co-ownership") can access the file FIL1. Note, however, that write access is not permitted.

(2) The file FIL2 is shareable, i.e. access by jobs running under other user IDs is permitted. Write access is, however, prevented by ACCESS=*READ.

(3) The file FIL3 is also shareable.
Write access is possible only if the requesting job has specified the password C'007'.

(4) The file FIL4 is again shareable. If it contains an executable program or a procedure, this can be executed by any user ID. Read or write access to FIL4 is possible only by specifying the password C'0815'. DELDATE and FREE-FOR-DELETION serve to define a period (150 days as from this day) after which the file may be deleted irrespective of access control.

(5) The file FIL5 is shareable, but no access is possible without a password.Entering the execute password C'2222' permits the file to be called via commands such as START-PROGRAM or CALL-PROCEDURE (for a program or procedure file, respectively). If the read password C'1111' is specified, the user can execute the program or procedure or can read, modify or delete the file. Once the retention period of 20 days has expired, the user may modify or delete the file if he/she specifies the read password.

(6) The new file FIL6 is to be assigned the same protection attributes as file FIL5. However, the file is to be overwritten with binary zeros in the event of releasing storage space, irrespective of the value specified for FIL5. Access control via BACL is activated. The following values are set for *STD when a new catalog entry is created: the file owner is automatically granted all access rights (read, write, execute), group members and “others” are assigned no access rights.

6.3 Defining co-ownership (co-owners)

Using SECOS, object owners can define the objects for which they wish to designate coowners together with the access conditions that these co-owners must fulfil when making administrative access attempts. The object owner is the user ID under which the object was created. Objects may take the form of files, job variables or libraries.

A co-owner is a user ID which is different from that of the object owner but which possesses the same rights as the object owner with regard to a specific object.

In general, the following applies to co-owners: all read, write and execute accesses to files are controlled by rules deriving from traditional file protection mechanisms:

- If a file is protected via SHARE/ACCESS or BASIC-ACL then a co-owner has the same read, write and execute rights as the file's owner.
- If a file is protected by GUARDS then access control is performed by evaluating the access conditions which are defined in STDAC guards.
- If a file is encrypted with a crypto password, access to the content can only take place after the crypto password has been entered.

Using the following commands:

- ADD-/MODIFY-/REMOVE-/SHOW-COOWNER-PROTECTION-RULE and
- ADD-/MODIFY-/REMOVE-/SHOW-ACCESS-CONDITIONS

Co-owners can be defined, displayed and removed.

If co-owners create files under a different ID and then protect these with an STDAC guard, then before accessing the file they must ensure that their ID possesses the necessary access rights. At the same time, file owners should be aware that co-owners can prevent them from accessing data.

However, both file owners and co-owners can use the MODIFY-FILE-ATTRIBUTES command at any time to recover unrestricted access rights.

6.3.1 Restrictions on TSOS co-ownership

By default, the TSOS user ID possesses an unrestricted co-administration right for files and job variables throughout the system. However, the use of SECOS makes it possible for all users to restrict this right provided that certain system-specific preconditions have been fulfilled by systems support.

The restriction to TSOS file co-ownership applies to certain commands and macros and, in some cases, to only certain of the involved operands. These commands and macros are listed in the table below:
.

Commands	Macros
MODIFY-FILE-ATTRIBUTES	CATAL (STATE=*UPDATE)
MODIFY-FILE-GENERATION-SUPPORT	CATAL (STATE=*UPDATE)
MODIFY-FILE-GROUP-ATTRIBUTES	CATAL (STATE=*UPDATE)
DELETE-FILE	ERASE
COPY-FILE	COPFILE

To provide effective protection against TSOS access, the owner of the object must make **two** user-specific protection settings:

1. First of all the owner must withdraw the **co-administration right** for his/her objects from the user TSOS.
2. Then the owner must withdraw the **access right** for his/her objects from the user TSOS. To do this, it is necessary to use GUARDS access protection since this is the only way to prevent TSOS access.

Both steps must be performed since withdrawing the co-administration right in step 1 simply prevents the modification of protection attributes. It does not prevent file accesses, for example attempts to read the file. To do this, it is necessary to perform step 2.

For more detailed information on the required steps and the system-specific settings to be made by systems support, refer to the “SECOS” manual [8 (Related publications)].

Another way to hide the file content from TSOS users is file encryption with a crypto password (see "File encryption"
).

6.3.2 Example for the definition of co-ownership

USER1 wants to give USER2 the right to create and administer files under his/her own user ID (USER1) provided that the file name contains the string “TEST”.

USER1 defines a condition guard COND1 which gives USER2 access that is unrestricted in terms of time constraints.

```
/create-guard cond1,user-inf='Zugriffsbedingungen fuer Coowner'
/add-access-conditions guard-name=cond1, -

/                      subjects=*user(user-identification=user2)
```

Then USER1 defines a rule container COO1 which contains a co-ownership rule. This specifies that the access conditions for the co-owners of the files whose name matches the pattern “*TEST*” are defined in the condition guard COND1.

```
/create-guard coo1,user-inf='Coowner-Regelbehaelter'

/add-coowner-protection-rule rule-container-guard=coo1, -

/            protection-rule=rule1, -

/            protect-object=*parameters(name=*test*,condition-guard=cond1)
```

For monitoring purposes, USER1 outputs information about all the guards and the rule container COO1. Precondition: no guards may have been present under the user ID USER1 at the start of this example session.

```
/show-guard-attributes
      Guard name        Scope   Type      Creation Date       LastMod Date
 ---------------------------------------------------------------------------
 :DEL1:$USER1.COND1     USR  STDAC    2011-04-19/10:35:47 2011-04-20/11:36:33
                        Zugriffsbedingungen fuer Coowner
 :DEL1:$USER1.COO1      USR  COOWNERP 2011-04-19/10:37:26 2011-04-20/11:38:53
                        Coowner-Regelbehaelter
 ---------------------------------------------------------------------------
 Guards selected: 2                                           End of display
/show-coowner-protection-rule coo1
 ---------------------------------------------------------------------------
 RULE CONTAINER :DEL1:$USER1.COO1                         COOWNER PROTECTION
 ---------------------------------------------------------------------------
 RULE1          OBJECT     = *TEST*
                CONDITIONS  = $USER1.COND1
                TSOS-ACCESS = SYSTEM-STD
 ---------------------------------------------------------------------------
 RULE CONTAINER SELECTED: 1                                   END OF DISPLAY
```

Since the name of the rule container does not comply with the naming conventions for rule containers it is simply used to prepare the default protection rule. USER2 as yet has no coownership rights for files under the user ID USER1, as is indicated by the call of the SHOW-COOWNER-ADMISSION-RULE command under the ID USER2.

```
/show-coowner-admission-rule $user1.*
 COO3316 NO COOWNER PROTECTION ACTIVE
```

To activate co-ownership protection, USER1 renames the inactive rule container COO1.

```
/mod-guard-attr guard-name=coo1,new-name=sys.ucf
```

USER1 displays the contents of the now active rule container.

```
/show-coowner-protection-rule
 ---------------------------------------------------------------------------
 RULE CONTAINER :DEL1:$USER1.SYS.UCF                ACTIVE COOWNER PROTECTION
 ---------------------------------------------------------------------------
 RULE1          OBJECT      = *TEST*
                CONDITIONS  = $USER1.COND1
                TSOS-ACCESS = SYSTEM-STD
---------------------------------------------------------------------------
 RULE CONTAINER SELECTED: 1                                    END OF DISPLAY
```

USER2 checks which rules make him or her a co-owner of files belonging to the user ID USER1.

```
/show-coowner-admission-rule $user1.*
 ---------------------------------------------------------------------------
 COOWNER RULES FOR FILE  :DEL1:$USER1.*
 ---------------------------------------------------------------------------
 RULE1          OBJECT     = *TEST*
                CONDITIONS = $USER1.COND1
 ---------------------------------------------------------------------------
 RULES SELECTED: 1                                             END OF DISPLAY
```

USER2 can now create the file TESTTEST under $USER1.

```
/create-file $user1.testtest
/show-file-att $user1.testtest
 0000003 :DEL1:$USER1.TESTTEST
 :DEL1: PUBLIC:      1 FILE  RES=       3 FRE=         3 REL=        3 PAGES
```

6.4 Importing protection attributes from another file or via default protection

The assignment of protection attributes for files should often follow predefined patterns. To support this feature, the system lets you import such patterns from existing files or assign specially configured default values that are controlled by default settings or the file name.

With the default protection function of SECOS (subsystem GUARDDEF), you can set such pubset-global or user-global default values for protection attributes. These default values are stored in attribute guards.

Default protection can be preset for the following protection attributes:

Protection attribute	Meaning
ACCESS	Standard access control (access type)
USER-ACCESS	Standard access control (access by other users)
BASIC-ACL	Basic access control list
GUARDS	Access control via GUARDS
READ-PASSWORD	Read password
WRITE-PASSWORD	Write password
EXEC-PASSWORD	Execute password
DESTROY-BY-DELETE	Binary deletion
SPACE-RELEASE-LOCK	Memory space lock
FREE-FOR-DELETION	Release date for deletion
EXPIRATION-DATE	Retention period

The values that are to apply for a file name through the use of default protection are assigned with the commands ADD-/MODIFY-DEFAULT-PROTECTION-RULE and ADD-/MODIFY-DEFAULT-PROTECTION-ATTR (see the “SECOS” manual [8 (Related publications)]).

Assigning protection attributes

The following DMS interfaces provide functions for default protection:

- CATAL macro
- CREATE-FILE command
- CREATE-FILE-GROUP command
- MODIFY-FILE-ATTRIBUTES command
- MODIFY-FILE-GROUP-ATTRIBUTES command

Assigning protection attributes via default protection

Protection attributes in accordance with predefined, name-sensitive default values (default protection) are transferred either implicitly by specifying nothing (default) or explicitly. The following specifications are required for the explicit transfer of protection attributes:

- in the commands, with the operand PROTECTION=*PAR(PROTECTION-ATTR=*BY-DEF-PROT-OR-STD)
- in the CATAL macro with the operand PROTECT=*BY_DEF_PROT_OR_STD

and by referring to this value in the individual operands (e.g. ACCESS=*BY-PROTECTION-ATTR).

Assigning protection attributes from existing files

Protection attributes are imported from existing files as follows:

- in the commands, with the operand PROTECTION=*PAR(PROTECTION-ATTR=*FROM-FILE(...))
- in the CATAL macro, with the operand PROTECT=(*FROM-FILE,<filename>)

Notes

- When a default date is imported, 0.00 hours local time is set; there is no conversion to UTC time.
- For temporary files, the only possible default protection attributes that can be preset using default protection are DESTROY-BY-DELETE and SPACE-RELEASE-LOCK.

Protection attributes when cataloging new files

Protection attribute	**PROTECTION-ATTR=**			
	***FROM-FILE**	***STD[1)]**	***BY-DEF-PROT-OR-STD**	
			Default protection not active[1)]	**Default protection active**
ACCESS	Value transferred from reference file	WRITE	WRITE	Value supplied by default protection
USER-ACCESS		OWNER-ONLY	OWNER-ONLY	
BASIC-ACL		NONE	NONE	
DESTROY-BY-DELETE		NO	NO	
GUARDS		NONE	NONE	
SPACE-RELEASE-LOCK		NO	NO	
READ-PASSWORD	NONE	NONE	NONE	

WRITE-PASSWORD	NONE	NONE	NONE	
EXEC-PASSWORD	NONE	NONE	NONE	
FREE-FOR-DELETION	NONE	NONE	NONE	NONE
AUDIT	NONE	NONE	NONE	NONE

1) System default values are entered.

No expiration date (EXPIRATION-DATE) can be defined for the first entry. In the case of files, it is implicitly preset to *NONE, and in the case of file generation groups to *TODAY.

Protection attributes when changing file attributes

Protection attribute	**PROTECTION-ATTR=**				
	***UNCH**	***FROM-FILE**	***STD1)**	***BY-DEF-PROT-OR-STD**	
				Default protection not active1)	**Default protection active**
ACCESS	UNCHANGED	Value transferred from reference file	WRITE	WRITE	Value supplied by default protection
USER-ACCESS	UNCHANGED		OWNER-ONLY	OWNER-ONLY	
BASIC-ACL	UNCHANGED		NONE	NONE	
DESTROY-BY-DELETE	UNCHANGED		NO	NO	
GUARDS	UNCHANGED		NONE	NONE	
SPACE-RELEASE-LOCK	UNCHANGED		NO	NO	
EXPIRATION-DATE 2)	UNCHANGED		TODAY	TODAY	
READ-PASSWORD	UNCHANGED	UNCHANGED	UNCHANGED	NONE	
WRITE-PASSWORD	UNCHANGED	UNCHANGED	UNCHANGED	NONE	
EXEC-PASSWORD	UNCHANGED	UNCHANGED	UNCHANGED	NONE	
FREE-FOR-DELETION	UNCHANGED	UNCHANGED	UNCHANGED	NONE	
AUDIT	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED

1) System default values are entered.

2) The expiration date is only entered for permanent files with creation dates or for file generation groups. If the reference file has no expiration date, *TODAY is entered.

Notes on default protection

Default protection and file types

Default values that do not match a file type are ignored. This affects:

- SPACE-RELEASE-LOCK, GUARDS, BASIC-ACL and free-for-deletion date for tape files
- SPACE-RELEASE-LOCK, GUARDS and free-for-deletion date for files and file generation groups on private disks
- GUARDS, BASIC-ACL, expiration date and free-for-deletion date for temporary files
- ACCESS, USER-ACCESS and passwords for temporary files on pubsets
- EXEC rights, EXEC passwords and USER-ACCESS=*SPECIAL for file generation groups.

In the case of tape files with a creation date, the specification PROTECTION-ATTR=*BY-DEF-PROT-OR-STD is rejected since the protection attributes of these files cannot be modified.

Renaming files

When changing file names, keep the following in mind:

- The default protection for the new file name is checked only if the default values are reset at the same time.
- When a file is renamed from permanent to temporary or vice versa and the protection attributes are simultaneously reset, the default values are determined on the basis of the new file name and file type.
- When a file is renamed as a file generation, the protection attributes cannot be simultaneously reset.

Protection function hierarchy

The values entered in each case are determined in the following order of priority:

1. Explicit specification in the command or macro
2. Value supplied by default protection or via a reference file
3. System default value

If you explicitly specify a protection attribute, no default value for another protection attribute is entered if this value would invalidate the explicit specification. In this case, the system default is entered instead of the default value.

Examples

- If you specify ACCESS or USER-ACCESS explicitly, BASIC-ACL and GUARDS are not set.
- If a value unequal to *NONE is specified for BASIC-ACL, GUARDS is not set.
- The value for the FREE-FOR-DELETION date of a file is skipped if a value unequal to *NONE has been explicitly specified for ACCESS, BASIC-ACL, the passwords or the expiration date.

Passwords

Default passwords are always stored in encrypted format in the attribute guard, even if N has been specified for the system parameter ENCRYPT.

If a default password is entered, all the passwords for the file are subsequently encrypted, again regardless of the value of the system parameter ENCRYPT.

In the case of new catalog entries, default values for passwords with PROTECTION-ATTR= *STD or PROTECTION-ATTR=*FROM-FILE() are not entered.

RFA (Remote File Access)

When RFA is used, the remote system's default values apply at all times.

Restrictions for default protection

Default protection is not used in the following circumstances:

- when a file is imported
- when a single file generation is specified
- for the GUARDS catalog
- when a reference file is specified (PROTECTION-ATTR=*FROM-FILE(...))
- when PROTECTION-ATTR=*STD is specified

Restrictions for new catalog entries

No default value for the free-for-deletion date is entered.

When entered for the first time (i.e. when the file is opened and when a file generation group is entered for the first time), the expiration date is not set to the defined default value, but to the system default value.

Restrictions/special features when resetting to default values

Unlike when resetting to system default values, the free-for-deletion date and the passwords are also changed.

If the default value for the expiration date has already passed, the current date is entered instead.

6.5 Passing on attributes when copying a file

When a file is copied by means of the COPFILE macro or COPY-FILE command, protection attributes can be passed on depending on the value of the PROTECTION or PROTECT operand.

In addition to the existing ways of passing on protection attributes with PROTECTION or PROTECT=*SAME, it is now also possible to enter protection attributes with PROTECTION or PROTECT=*STD when used together with the “default protection” function. The last change date of the source file can be accepted with CHANGE-DATE or CHANGE=*SAME.

Generally, the protection attributes of the source file are not used. This is possible only if copying within the caller's user ID or copying a foreign file to one of the caller's files. In all other cases, the shareability and access authorizations are set to the default values (namely SHARE=NO and ACCESS=WRITE with COPFILE or USER-ACCESS=*OWNER-ONLY and ACCESS=*WRITE with COPY-FILE).

The passing of protection attributes is contingent upon the following factors:

- the direction of the copy operation, namely
  - within the same user ID
  - from some other user ID
- how the source file is protected
  - by standard access control
  - by a BACL (basic access control list)
  - by GUARDS
- the type of the target file
  - a file on a public volume (pubset)
  - a file on a private disk
  - a tape file
  - a file generation
  - a temporary file

The backup attributes LARGE, BACKUP, MIGRATE, NUM-OF-BACKUP-VERS, OPNBACK and MANCLAS (see the CATAL macro) and the backup attributes DESTROY, RETPD, ENCRYPT, RDPASS, WRPASS, EXPASS and DELDATE are copied subject to the conditions mentioned above. The following applies to write attributes:

Copying within the same user ID

Source file protected by	Target file on pubset	Target file on Private disks	Target file on Tape
Standard access control	(1)	(1)	(1)
Basic Access Control List, (BACL)	(1)	(1)	(2)
GUARDS	(1)	(3)	(2)

Table 22: Copying within the same user ID and CATID

Copying from a foreign user ID to one's own user ID

Source file protected by	Target file on pubset	Target file on Private disks	Target file on Tape
Standard access control	(1)	(1)	(1)
Basic Access Control List, (BACL)	(3)	(3)	(2)
GUARDS	(3)	(3)	(2)

Table 23: Copying from a foreign user ID or CATID

(1) The attributes of the source file are copied. If there is already a catalog entry for the target file, the attributes in this entry are replaced; any existing BACL or GUARDS entry is deleted or overwritten.

(2) The default values for tape files (SHARE=YES or USER-ACCESS=ALL-USERS, ACCESS=WRITE) are inserted. Tape files do not have any BACL or GUARDS entries.

(3) The default values for disk files are inserted. If there is already a catalog entry for the target file, the standard access control attributes in this entry are replaced; any existing BACL or GUARDS entry is deleted.

Copying from the user's own user ID to another user ID

Copying in this manner is possible only if there is already a file cataloged under the other user ID. Only the data is copied into the existing file. Specifying the operand PROTECT=SAME in the macro or the operand PROTECTION=SAME in the command has no effect in this case.

Special cases

- Systems support

  The user ID with the TSOS privilege is the co-owner of all files and may also create files under other user IDs (for restrictions see "Restrictions on TSOS co-ownership"). Copying under such a user ID is thus always carried out as described for “Copying within the same user ID”, even if the user IDs of the source and target files are different. This does not apply to other co-owner IDs (see "Defining co-ownership (co-owners)").

- file generations.

  The protection attributes defined for a generation group apply to each generation in this group. The individual generations do not have their own separate protection attributes.

  If a generation is copied, the protection attributes of the associated group are used (since these apply to each individual generation).

  If a generation is copied into a group, it receives the attributes of this target group (i.e. no attributes of the source file are transferred).

- Temporary files

  Temporary files are task-local files. For this reason, no protection attributes of the source file are transferred when a temporary file is copied.

Copying with PROTECTION=*STD or PROTECT=*STD

If the target file is not present and default protection is active then the target file is assigned the protection attribute values supplied by default protection. If default protection is not active then – as previously– the system default values are assigned. If the target file exists then the existing values remain unchanged.

6.6 Effects of the protection attributes during file processing

6.6.1 Deleting files, and displaying and modifying file attributes

Only the file owner (see "Defining protection attributes" for definition) can delete a file (ERASE macro, DELETE-FILE command) or modify its attributes (CATAL macro, MODIFY-FILE-ATTRIBUTES command).
If co-ownership rights have been assigned for a user ID (see "Defining co-ownership (co-owners)"), then the co-owners have the same rights with regard to the deletion, modification and display of file attributes as the file owner.

The user ID with the TSOS privilege has the same access rights as the file owner (subject to restrictions, see "Restrictions on TSOS co-ownership").
It can display and modify the attributes of any file, even without entering an existing password, provided that its co-ownership has not been restricted. For tape files, this is true only insofar as it is technically possible (e.g. the labels cannot be modified without the use of a special copy program).

The user ID with the TSOS privilege can also find out which password has been assigned to a file. If, however, password encryption is used, only the encrypted value is shown; this cannot be used to gain access to files.

Write access authorization is required in order to delete a file (ACCESS=WRITE or the appropriate attributes with BACL or GUARDS).
If IGNORE=ACCESS is used in the ERASE macro or IGNORE=*ACCESS in the DELETE-FILE command, the absence of write access can be ignored and the file can be deleted without first changing the protection attributes.

6.6.2 Access to password-protected files

When a job wishes to access a file protected by passwords, the password required to permit access must be specified using the ADD-PASSWORD command. The passwords entered in this manner are entered in the password table of the job and do not need to be repeated for each subsequent access to the file.

Passwords are subject to the following hierarchy:

- write password
- read password
- Execute password

The table on "Access to password-protected files" shows which passwords must be entered before the various types of access.

As long as the password required for catalog or file access is stored in the job's password table, it does not need to be entered again before each access to this file or to any other file protected by this password. If the user wishes to restore full password protection, he/she can delete the password from the password table or delete the entire password table via the REMOVE-PASSWORD command.

If a password is not explicitly removed from the password table or the entire password table explicitly deleted by means of the REMOVE-PASSWORD command, then this is done implicitly at the end of the job, since the password table is job-specific.

The table below shows the possible combinations of password protection:

Password protection	**Password entered**	**Execute**	**Read**	**Write**
EXEC-PASSWORD	None	--	--	--
	Execute password	X	X	X
READ-PASSWORD	None	X*)	--	--
	Read password	X	X	X
WRITE-PASSWORD	None	X	X	--
	Write password	X	X	X
EXEC-PASSWORD READ-PASSWORD WRITE-PASSWORD	None	--	--	--
	Execute password	X*)	--	--
	Read password	X	X	--
	Write password	X	X	X
EXEC-PASSWORD READ-PASSWORD	None	--	--	--
	Execute password	X*)	--	--
	Read password	X	X	X

EXEC-PASSWORD WRITE-PASSWORD	None	--	--	--
	Execute password	X	X	--
	Write password	X	X	X
READ-PASSWORD WRITE-PASSWORD	None	X*)	--	--
	Read password	X	X	--
	Write password	X	X	X

Table 24: Passwords to be entered for various types of access

X	Access is granted
--	Access is not granted
*)	The program code is protected against access by dumps

Deleting password-protected files

A file protected by a password can be deleted only if write access is enabled by entering the appropriate password (exception: the FREE-FOR-DELETION date has been reached, see "Defining a deletion date"). The password can be added to the password table of the job by means of the ADD-PASSWORD command or it can be specified in the ERASE macro or DELETE-FILE command.

Both the PASSWD operand of the ERASE macro and the PASSWORDS-TO-IGNORE operand of the DELETE-FILE command permit input of a password which is valid only for this macro or command and is not entered in the password table. In this way, full access protection is maintained for all other files protected by the same password which are not affected by the deletion operation.

6.6.3 Access to a file for which a retention period is defined

A retention period can be defined or modified with the CATAL macro or MODIFY-FILE-ATTRIBUTES command (entry in the file catalog) or with the FILE macro or ADD-FILE-LINK command (entry in the TFT).

No write access to the file is permitted during the specified retention period. Only the owner of a file can modify its retention period. A file with a retention period can be deleted without resetting the retention period by specifying IGNORE=EXDATE in the ERASE macro or IGNORE-PROTECTION=*EXPIRATION-DATE in the DELETE-FILE command.
If the FREE-FOR-DELETION date has been reached, the file can be deleted even if its retention period has not yet expired.

6.6.4 Protection of multifile tapes against implicit deletion

If write access to a tape file is permitted, all subsequent files on this tape are implicitly deleted, regardless of their protection attributes. Their catalog entries remain unchanged, but the data can no longer be accessed.

If the FILE macro with the operand SECLEV=(...,OPR) or the ADD-FILE-LINK command with the operand OVERWRITE-PROTECTION=YES is executed, the system checks, during a write access, which protection attributes have been specified for the new file. If this new file is not the first file on the tape, its protection attributes are compared with those of the immediately preceding file on the tape.

An attempt to write to the tape is rejected if

- ACCESS=READ is specified for the current file and ACCESS=WRITE is specified for the immediately preceding file or
- the retention period specified for the current file is greater than that for the immediately preceding file.

As a result of this, the protection specified for the first (or immediately preceding) file on the tape determines whether or not the following files are protected against overwriting.

6.6.5 Transferring protection attributes to the tape label

The tape labels are generated when a tape file is created; they cannot be modified after this. The LABEL operand of the FILE macro and the LABEL-TYPE operand of the ADD-FILE-LINK command define which file attributes are to be transferred to the tape label. Specifying LABEL=(STD,3) in the macro or LABEL-TYPE=*STD(DIN-REV-NUM=3) in the command provides the highest possible degree of protection. The setting LABEL=NO (macro) and LABEL-TYPE=*NO (command) should be avoided in all cases.

6.6.6 Checking the protection attributes of tape files

When a tape file is created, the attributes are stored in the file catalog. If the catalog entry is deleted, only the entries in the tape labels are available.

During read and write accesses to tape files, the entries in the file catalog are compared with the entries in the tape labels. Depending on the authorization of a user ID and the user specifications for access to the tape file (command ADD-FILE-LINK, operand LABEL-TYPE or BYPASS-LABEL-CHECK), the prevailing security measures can be bypassed.

6.7 Data protection by “data destruction” (DESTROY option)

Both in the case of disks and tapes, the user can specify that files that are no longer required can be overwritten with binary zeros and thus “destroyed”. These files are no longer readable in BS2000, even with diagnostic programs and special privileges.

i Irrespective of whether this option is used, the following **always** applies: If a volume (electronic, magnetic or optical storage medium, like hard disk, storage component of a disk storage system, tape etc.) on which sensitive data are or were stored, is exchanged, the volume itself must be destroyed after the data have been deleted in BS2000. This is the only way to guarantee that no data can be reconstructed from this volume.

Using the CATAL macro or the CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands it is possible to specify in the catalog entry that:

- For disk files:
  Storage space assigned to the file is explicitly deleted when it is released.
- For tape files:
  On tape change or following the closing of the output file, any residual data on the tape is deleted.

Disk files

In the case of disk files, it is possible to specify “data destruction” when storage space is released.

This option can either be entered in the file catalog or explicitly specified when files are deleted.

in catalog entry:	CATAL ...,DESTROY=*YES /CREATE-FILE ...,DESTROY-BY-DELETE=*YES /MODIFY-FILE-ATTRIBUTES ...,DESTROY-BY-DELETE=*YES
on deletion:	ERASE ...,DESTROY /DELETE-FILE ...,OPTION=*DESTROY-ALL

Tape files

In the case of tape files, the FILE macro (operand DESTOC=YES) or ADD-FILE-LINK command (operand DESTROY-OLD-CONTENTS=*YES) can be used to specify that any subsequent (old) data on the tape should be overwritten (i.e. physically deleted) when the output file is closed (CLOSE) or the tape is changed.

Automatic “data destruction” on reconstruction

DMS performs automatic “data destruction” when recovering files that were destroyed, for example, due to a system crash: if the file that is to be recovered is located on a private disk or is not cataloged under the job's user ID then the auxiliary files created by DMS in order to perform the recovery are deleted using the “data destruction” mechanism (see "Recovery of files under foreign user IDs").

6.8 File encryption

Outsourcing, server consolidation and storage virtualization have resulted in a call for the contents of files to be protected both on backups and also from people with the TSOS privilege. This requirement is implemented by file encryption integrated in the file system.

6.8.1 Concept of file encryption

Encryption takes place using one of the standard symmetrical methods AES or DES (the encryption method is defined with a system option in the system parameter FILECRYP) and a crypto password that must be entered. To permit the unencrypted content of an encrypted file to be accessed, this crypto password must be entered on a task-local basis.

File encryption is defined in a protection attribute in the file attributes, but in technical terms also affects the file format.

The system parameter FILECRYP can be modified at any time, for instance from AES to DES. The system parameter affects (only) the selection of the method in the event of conversion to an encrypted file (ENCRYPT-FILE). Files already encrypted maintain their encryption attributes (also the method used) in the catalog entry and remain unaffected by a modification of the system parameter.

Encrypted files are maintained on the pubset disks, encrypted on a page basis. Here all pages of a file are encrypted in the same way. The key for encrypting/decrypting for read or write access to the file is derived from the check of the crypto password when an encrypted file is opened.

Encryption/decryption is implemented using the Crypt subsystem.

Different files can have identical or different crypto passwords. As a result, identical or different keys are used for encryption.
When a file is converted into an encrypted file (ENCRYPT-FILE) the crypto password can also be taken over from a reference file that is already encrypted. This supports the assignment of identical crypto passwords for associated files. The system parameter FREFCRYP enables the introduction of new crypto passwords to be restricted to files of a particular user ID. Files outside this user ID can then only be assigned a crypto password taken over from a reference file.
The catalog entry for an encrypted file contains the encryption indicator, the encryption method and a check pattern for checking the crypto password.
The crypto password and the key are not stored in the system.

The use of DAB (Disk Access Buffer) reduces the inputs/outputs to disk by means of buffering. When encrypted files which use DAB buffering are accessed, this obviates the need for inputs/output to disk and also encryption /decryption.

Homogeneous transfer and integration into existing applications

In the event of **homogeneous transfer** of an encrypted file, the encrypted content is transferred 1:1 to a target file which has the same encryption attributes as the source file.

This homogeneous transfer is used for:

- homogeneous COPY-FILE
- saving and restoring (SAVE/RESTORE) with HSMS/ARCHIVE
- migrating and recalling (MIGRATION/RECALL) with HSMS
- moving files within an SM pubset to another volume set
- file transfer (homogeneous only; for information on unhomogeneous file transfer see Restrictions on "Preconditions and restrictions for file encryption")

Thus in the event of homogeneous transfer no decryption takes place, and no key and no crypto password are required. The target file is created in encrypted form, which requires no more time than for the corresponding operation for an unencrypted file.

Systems support can, on the one hand, operate as usual with encrypted files as no crypto password is required for the aforementioned transfers, but then, on the other hand, they will not have access to the decrypted content of encrypted files.

To permit the simple introduction of encrypted files into existing applications, securityrelevant files are converted once into encrypted files. In this case a crypto password must be defined. As with the file passwords, a crypto password can be specified outside the application before the application is then started in the same task. The application then automatically reads and writes the encrypted content of the file in the event of file accesses. On the disks the content of the file remains encrypted. In this way applications do not need to be modified when encrypted files are used.

6.8.2 Preconditions and restrictions for file encryption

BS2000 supports the use of encrypted files via the CRYPT subsystem (see the “CRYPT” manual [24 (Related publications)]). CRYPT can also be used for other purposes in the same system (e.g. for encryption in the event of file transfer or IPSec).

If an encrypted file is located on a shared pubset, the preconditions for operation with encrypted files must be met on all systems from which the file is to be accessed with encryption.

Existing application which are to work with encrypted files generally do not need to be modified. Modifications to applications are necessary only if new, encrypted files are to be created by the application.

Restricted access

Pubsets with encrypted files can also be imported when the system does not fulfill the requirements for file encryption (e.g. CRYPT subsystem is not available).

In this case it is not possible to access these files with decryption of the contents. Nor can the file attributes be modified. The encrypted files can only be displayed (SHOW-FILE-ATTRIBUTES) and deleted (DELETE-FILE). Homogeneous transfer of these files (see "Concept of file encryption") is possible.

Scope and use of encrypted files

Only disk files on pubsets can be encrypted.

Encrypted files can be used for all access types (exceptions: EAM and BTAM), all file types and all pubset types.

The crypto password must be specified for restoring ISAM files (VERIFY) and for conversion using PAMCONV.

Compatibility with other protection mechanisms

When a file protected with a read and/or execute password is converted into an encrypted file, it loses its read and /or execute password protection.

File encryption can be combined with the write password (WRITE-PASSWORD) and with all other protection mechanisms. The familiar features of these protection mechanisms are complemented by the crypto password.

Restrictions and special aspects

- The link to the CRYPT subsystem means that access to encrypted files is possible only as of “System Ready”.
- Job variables, tape files and files on Net-Storage or private disks cannot be encrypted.
- Files of the TSOS user ID on the home pubset cannot be encrypted.
- An encrypted file must be decrypted before it can be printed (see also “Printing out anencrypted file” (Application scenarios)).
- The SAVE-PLAM-INFO option is not executed for encrypted PLAM libraries in the event of an HSMS backup.
- Conversion from K format to an NK format is not possible when restoring or importing a K file from a save volume to an NK disk if the file is encrypted.
- Files with a valid last-byte pointer are not encrypted.

6.8.3 User interfaces

The following DMS commands and macros are available for file encryption:

Command	Meaning
ADD-CRYPTO-PASSWORD	Stores the crypto password for decrypting encrypted files in the task's password table
DECRYPT-FILE	Converts an encrypted file into an unencrypted file.
ENCRYPT-FILE	Converts an unencrypted file into an encrypted file.
REMOVE-CRYPTO-PASSWORD	Removes the crypto password from the password table of the ongoing task.
Macro	**Meaning**
DECFILE	Converts an encrypted file into an unencrypted file.
ENCFILE	Converts an unencrypted file into an encrypted file.

Encryption

An unencrypted file is converted into an encrypted file using the ENCFILE macro or the ENCRYPT-FILE command. If a reference file that has already been encrypted is specified instead of the crypto password, the file is assigned the same encryption attributes as the reference file, in particular the same crypto password. The file need only be cataloged for encryption.

When an unencrypted file is converted into an encrypted file, the read and/or execute password (READ-/EXEC-PASSWORD) is deleted.

The encryption method (AES or DES) defined in the system parameter is used for conversion to an encrypted file and is then linked to the file.

Encryption is possible for an owner's files or for co-owned files (as is the authorization to create a file) and can only take place locally, not via RFA.

! **Caution**

If the crypto password is no longer known, the file content cannot be decrypted! The crypto password should therefore be stored in a safe place (see also "Administering encryption" (Application scenarios)).

Access to encrypted files

The ADD-CRYPTO-PASSWORD command stores the crypto password in the task's crypto password table. Only with the correct crypto password is it possible to open a file to access the unencrypted content.

When an encrypted file is simply moved, no crypto password is required.

The crypto passwords are stored in the crypto password table in permanently encrypted form only, regardless of the system parameter for encrypting file passwords.

If a procedure contains a crypto password (e.g. in an ADD-CRYPTO-PASSWORD
command) and should thus be made unreadable for third parties – in particular for the executing party – it is advisable to use SDF-P compilation with automatic permanent encryption (see “Processing encrypted files in procedures” (Application scenarios)). For details on the use of SDF-P see the “SDF-P” manual [17 (Related publications)].

The REMOVE-CRYPTO-PASSWORD command is used to remove a crypto password from the crypto password table of the ongoing task. It is not possible to decrypt files which are encrypted with this crypto password.

The ADD-/REMOVE-CRYPTO-PASSWORD commands are automatically forwarded to all RFA partner processes.

Decryption

The DECFILE macro or DECRYPT-FILE command is used to convert an encrypted file back into a decrypted file. The crypto password must be specified before the function is called.

Decryption is possible for an owner’s files or for co-owned files (as is the authorization to create a file) and can only take place locally, not via RFA.

Further dependencies

CONCATENATE-DISK-FILES
The crypto password must be specified to conatenate encrypted SAM files (with ADD-CRYPTO-PASSWORD).

COPY-FILE
The encryption attributes are - as far as possible - taken over from the source file to the target file. The content of the file is copied 1:1. It is not necessary to specify the crypto password.
If the encryption attributes cannot be taken over, the crypto password must be specified (ADD-CRYPTO-PASSWORD). This is the case, for example, if the target file cannot be encrypted or if the target is a file generation with the uniform file protection attributes of the file generation group.

DELETE-FILE
A file selection which is dependent on the file encryption is offered, in other words according to the values of the file attribute ENCRYPTION.

MODIFY-FILE-ATTRIBUTES
If en encrypted file has a catalog entry but as yet no storage space, it may not be allocated storage space or a tape type on private disk.
Encryption attributes cannot be modified. Modifications to the read/execute password are ignored in the case of encrypted files.

REPAIR-DISK-FILES
In order to repair encrypted ISAM files you must specify the associated crypto password (ADD-CRYPTO-PASSWORD). The file copy is assigned - as far as possible - the same encryption attributes.

SHOW-FILE
The content of an encrypted file is displayed in encrypted form. You must specify the crypto password here (ADD-CRYPTO-PASSWORD).

SHOW-FILE-ATTRIBUTES
The file attribute ENCRYPTION in the SECURITY section indicates whether and with which method (AES or DES) a file is encrypted. A file selection is offered in accordance with these values.

6.8.4 Application scenarios

The EXTERNAL_ACCOUNTS file is rated as “security-relevant” and is thus to be maintained in encrypted form. It is converted once only into an encrypted file:

```
/ENCRYPT-FILE FILE-NAME=EXTERNAL_ACCOUNTS, -

               CRYPTO-PASSWORD='ELEPHANT',CONFIRM-PASSWORD='ELEPHANT'
```

You can then check whether and with which encryption method the file was encrypted:

```
/SHOW-FILE-ATTRIBUTES EXTERNAL_ACCOUNTS,SECURITY=*YES
0000000066 :X:$U234.EXTERNAL_ACCOUNTS
------------------------------- SECURITY     -----------------------------
   READ-PASS  = NONE        WRITE-PASS = NONE        EXEC-PASS  = NONE
   USER-ACC   = OWNER-ONLY  ACCESS     = WRITE       ACL        = NO
   AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = 2007-02-08
   DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME =   00:00:00
   SP-REL-LOCK= NO          ENCRYPTION = AES
 :X:    PUBLIC:      1 FILE  RES=        66 FRE=       50 REL=       18 PAGES
```

The following application scenarios show the different requirements for applications or users which result from working with encrypted files.
Some of the applications must be adapted, with others it is sufficient to specify the crypto password to work with encrypted files.

The following application scenarios are presented:

- Application writes to an encrypted file
- Application reads an encrypted file and generates an encrypted output file
- Processing encrypted files in procedures
- Forwarding an encrypted file to subtasks
- Printing out an encrypted file
- Administering encryption

Application writes to an encrypted file

The UPDATE.EXT_ACCO application enters the current account balances in the EXTERNAL_ACCOUNTS file. The application can remain unchanged even if the EXTERNAL_ACCOUNTS file is now encrypted. Before the application is called only the crypto password need be specified:

```
/ADD-CRYPTO-PASSWORD 'ELEPHANT'
/START-EXEC-PROGRAM UPDATE.EXT_ACCO
/REMOVE-CRYPTO-PASSWORD 'ELEPHANT'
```

Application reads an encrypted file and generates an encrypted output file

The KONTSORT application sorts the accounts according to the current account balances and generates an output file for this purpose. If the input file is encrypted (as for example the EXTERNAL_ACCOUNTS file), the output file should also be encrypted in the same way. KONTSORT must be modified for this purpose.

After the output file has been created and before it is opened, the conversion must be entered in an encrypted file:

```
FILE OUTPUT_FILE
ENCFILE PATHNAM='OUTPUT_FILE',REFFILE='INPUT_FILE'
OPEN OUTPUT_FILE
```

If the input file is not encrypted, execution of the ENCFILE macro is rejected and the output file is not encrypted. If the input file is encrypted, the encrypted output file is assigned the same crypto password as the input file. This must be specified when the KONTSORT application is called (as in “Application writes to an encrypted file”) so that KONTSORT can open the input file.

Processing encrypted files in procedures

The UPDATE.EXT_ACCO program (see “Application writes to an encrypted file”) must be called in the S procedure UPDATE.PROC which is run each day by the systems support. The ADD-CRYPTO-PASSWORD='ELEPHANT' statement has been entered in the S procedure, but the systems support should not know the crypto password.

Procedure: The S procedure UPDATE.PROC is converted with COMPILE-PROCEDURE (SDF-P is required for this). The result is a compiled procedure file in which the crypto password is no longer readable. Only this file is made accessible to the systems support. Systems support can execute it, but not read it.

Forwarding an encrypted file to subtasks

The TAX.EXT_ACCO program analyzes each account in the EXTERNAL_ACCOUNTS file and creates tax certificates for them. The program operates asynchronously in multiple subtasks and opens the encrypted file EXTERNAL_ACCOUNTS there. Specifying the crypto password via the task-specific crypto password table is not suitable for this purpose because the authorization is not required by the calling task, but in the subtasks. The program must thus be modified.

A program parameter is introduced via which the crypto password can be specified. The program call then looks as follows:

```
/START-EXEC-PROGRAM TAX.EXT_ACCO
```

```
*Crypto-Password: 'elephant'
```

The program must then forward the specified crypto password to the subtasks which open the EXTERNAL_ACCOUNTS file.
In the subtask the program must then specify the crypto passsword via the P1-FCB (for a description of the FCB macro see the “DMS Macros” manual [1 (Related publications)]) before opening the EXTERNAL_ACCOUNTS file.

Printing out an encrypted file

When printing out an encrypted file (manually or from a program) security considerations must be borne in mind – particularly in the case of a centralized printout.
You can satisfy these security considerations by, for example, restricting the printouts to manual printouts only on a local printer in the same room.

An encrypted file cannot be printed out directly. The user must temporarily generate an unencrypted copy of the file to be printed out and print out this copy. After the copy has been printed out its contents must be overwritten with binary zeros.

```
/ADD-CRYPTO-PASSWORD 'ELEPHANT'
/COPY-FILE EXTERNAL_ACCOUNTS,TEMP-PRINT
/DECRYPT-FILE TEMP-PRINT
/REMOVE-CRYPTO-PASSWORD 'ELEPHANT'
/PRINT-DOCUMENT TEMP-PRINT,DELETE-AFTER-PRINT=*DESTROY
```

Note

If the TEMP-PRINT copy is created as a private disk file before COPY-FILE, the file content is decrypted with COPY-FILE (because a private disk file may not be encrypted), which means that DECRYPT-FILE is not needed.

Administering encryption

The auditing department must monitor the use of encrypted files, check on the assignment of crypto passwords, and store them in a safe place to prevent them being lost.

1. The auditing department sets up its own ID SAMPLE on the pubsets which are to contain encrypted files.

   In addition it sets the system parameter FREFCRYP to the value “SAMPLE” in those systems where encryption is to be used.

   Now encrypted files can only be set up outside the ID SAMPLE with reference to files which are already encrypted and no longer using freely selectable crypto passwords.
2. The department for foreign accounts applies to the auditing department for permission to use encrypted files.
3. The auditing department authorizes this and sets up a shareable file with the name $SAMPLE.REFERENZ. EXTERNAL_ACCOUNTS. The file content is irrelevant.
4. The representative for the foreign accounts converts this file into an encrypted file and specifies a self-defined crypto password.
5. A hardcopy of this ENCRYPT-FILE call (with the defined crypto password) is sealed and placed in the safe of the auditing department.
6. The department for foreign accounts can now encrypt files on its ID as required by referring to this reference file.

   It is not possible for the department to refer to other reference files if it does not know the associated crypto password.
7. An in-house convention could be, for example, that employees who know the associated crypto password are listed in the reference file.

6.8.5 Performance, load balancing, failure situations

Encryption/decryption slows down file accesses as if the disk access as a whole were slower. The use of DAB is beneficial because the absence of disk accesses also means that encryption/deccryption is not required.

7 Data security

7.1 Protection during parallel access

Besides protecting files by means of catalog attributes, DMS offers mechanisms which prevent mutual interference between jobs that wish to process a file at the same time.

- Multiple jobs may read a file simultaneously (the protection attributes of the file are, of course, still observed).
- For the access methods ISAM, UPAM, FASTPAM and DIV (see the corresponding descriptions), DMS offers the “shared-update processing” function. This function permits several jobs to read and write a file concurrently. In order to avoid mutual interference, parts of the file can be locked by a specific job.

In addition to these internal mechanisms, the user himself can protect his data against multiple access by reserving volumes or files by means of the SECURE-RESOURCE-ALLOCATION command. This is particularly useful for file generation groups, where it is important to maintain data consistency between the generations (see section “Pubset backup with Snapsets”).

7.2 File recovery

If processing of a file is not terminated normally (e.g. because the job is aborted as the result of a system crash or because the job is suspended due to memory saturation and not reactivated before the system is closed down), the contents of the file may become inaccessible for future processing.

Possible consequences of abnormal termination of file processing:

- a file lock is not cleared
- the file contents and the catalog entry do not match (e.g. last-page pointer)
- the index and data sections of an ISAM file are inconsistent.

The user can reconstruct such files with the aid of the VERIF macro or the REPAIR-DISK-FILE command.

If a program attempts to access a destroyed or locked file, an error routine containing a VERIF macro can be activated via the EXLST exit OPENC (see the EXLST macro).

The user can use FSTAT (STATE=NOCLOS operand) or the SHOW-FILE-ATTRIBUTES command (STATUS=*PAR(CLOSED-OUTPUT=*NO) operand) to check whether any files are still open.

7.2.1 Clearing a file lock

“Clearing a file lock” means that the entry in the file lock table is deleted. This can be done using the VERIFY function (VERIF macro or REMOVE-FILE-ALLOCATION-LOCKS command). A file can be unlocked with this macro or command only if the job processing the file is permanently inactive and the file was locked by means of the SECURE-RESOURCE-ALLOCATION command or if the file is a tape file.

Files on a shared pubset may be locked after a system crash.

Disk files that were not locked by the SECURE-RESOURCE-ALLOCATION command can be unlocked only by systems support, under the user ID TSOS. To do this, systems support must first ensure that no one is working with the file.

Disk files which have already been unlocked are treated by the VERIF macro and REPAIR-DISK-FILE command in different ways, depending on the access method with which they were created.

- SAM files: the last-page pointer in the catalog entry is updated, which means that the file can be opened again even in EXTEND mode. However, any records which were located in the I/O buffer when the job was aborted and which had not been transferred to the file may be lost (see section “Recovery limits”).
- ISAM files: all records that can be recovered are written into a new ISAM file (the recovery file); data blocks which cannot be interpreted are written into a PAM file (see section “Recovery limits”).

  The user may specify which file is to be used as the recovery file. If this file is on a private disk, there must be sufficient storage space available. If the user specifies a recovery file in the VERIF macro or REPAIR-DISK-FILE command, the damaged file is not overwritten or erased.

  If the user does not specify a recovery file, the ISAM file is recovered in a work file on public volumes. The subsequent processing of the original file and the work file depends on the type of volumes on which the damaged file was stored.

  - If the damaged file was on public volumes, it is erased (but, for performance reasons, without data destruction) and the work file is renamed with the name of the original file.
  - If the damaged file was on private volumes, it is overwritten with the contents of the work file. Then the work file is erased (here, for data protection reasons, with data destruction).
- PAM files: DMS executes a privileged close operation and then updates the last-page pointer.

  The last-page pointer is set to the last occupied page for files for which BLOCK-CONTROL-INFO=NO has been specified. Files with BLOCK-CONTROL-INFO=WITHIN-DATA-BLOCK are read backwards, and the last-page pointer is set to the last page that belongs logically to the file. This may be very time-consuming in the case of large files.

  The last-byte pointer is set to zero.

7.2.2 Recovery limits

The fact that records of SAM and ISAM files are buffered in main memory may result in the loss of updates which were made immediately before file processing was aborted. For ISAM files, this problem can be countered with the WROUT function (see "WROUT function"). However, users should bear in mind that this involves increased time outlay.

If duplicate records (i.e. records with the same key and the same data) are encountered during the recovery of ISAM files, only one record is transferred to the recovery file. This restriction is necessary in order to permit the recovery of ISAM files where block splitting was in progress when file processing was aborted.

If the damaged ISAM file contains several records with the same key(s) but different data, the order in which these records are written into the recovery file may differ from that in the original file.

If the ISAM file to be recovered was encrypted with a crypto password, its encryption attributes are transferred to the recovered file.

If a data block in an ISAM file cannot be recovered (e.g. because internal pointers have been destroyed), this block is written into a separate PAM file. This PAM file is then available to the user, who can attempt to rescue the contents of the blocks which could not be recovered. The file name is S.filename.REPAIR, where “filename” is the file name of the original (damaged) file.

During the recovery of ISAM files, no space is reserved in the data blocks for subsequent extensions.

The time required for the recovery of ISAM files is considerable. In the case of files on private disks, it can become even greater if the user does not provide a recovery file (which already has sufficient storage space allocated).

ISAM files whose index and data sections are stored on different private disks can be recovered only if BUF-LEN=STD/(STD,1) has been defined in the catalog entry.

7.2.3 Recovery of files under foreign user IDs

A user can reconstruct a file cataloged under another user ID only if this file is shareable. Unless specified otherwise, DMS creates a work file under the requesting user's ID. After the damaged file has been recovered, the contents of this work file are copied back into the original file and the work file is erased with data destruction. In some cases, a third file is needed for recovery. This is also created under the user ID of the requesting job. The work files are created on the same volume as the damaged file.The user requesting recovery must therfore be authorized to access the volume containing the damaged file.

7.3 Checkpoint and restart processing

In programs, checkpoints can be set by means of the WRCPT macro; when a checkpoint is encountered, the program status, the system environment, etc. are written into a checkpoint file. For large tape files, automatic writing of checkpoints can be specified using the FILE macro (operand CHKPT) or ADD-FILE-LINK command (operand CHECKPOINT-WRITE).

If a program is aborted, it can be restarted from one of these checkpoints with the aid of the RESTART-PROGRAM command.

7.4 WROUT function

For ISAM files, the so-called “WROUT function” (also called “write immediate”) can be used to control how often updated blocks are written to disk. If WROUT is active, each updated block is written to disk immediately, thus maintaining consistency between the data on the disk and the data in virtual memory. However, due to the increased number of I/O operations when WROUT is active, system performance drops considerably.

If the WROUT function is not active, I/O operations are executed only when the contents of the associated buffer area need to be replaced. This delay means, for example, that the number of write operations can be reduced if several records in the same block are updated.

The WROUT function can be activated explicitly or implicitly:

- explicitly with WROUT=YES in the FILE or FCB macro for both K-ISAM files and NK-ISAM files.
- explicitly with the operand WRITE-IMMEDIATE=*YES of the ADD-FILE-LINK command for both K-ISAM files and NK-ISAM files.
- explicitly with WROUT=YES when task-specific ISAM pools are created by means of the CREPOOL macro; this then applies to all NK-ISAM files which are processed in this pool.
- explicitly with WRITE-IMMEDIATE=YES when task-specific ISAM pools are created by means of the CREATE-ISAM-POOL command; this then applies to all NK-ISAM files which are processed in this pool.
- implicitly when a host-specific ISAM pool is created using the CREPOOL macro or the CREATE-ISAM-POOL command. This applies to all NK-ISAM files which are processed in this pool.
  ISAM pools which are not task-specific are created with the attribute WRITE-IMMEDIATE=YES by default. ISAM pools, in which updates are written to disk after a delay are, however, also permitted and can be created by explicitly specifying an appropriate parameter.
- implicitly with SHARUPD=YES in the FILE or FCB macro when a file is opened for shared-update processing (for both K-ISAM and NK-ISAM). When K-ISAM files with SHARUPD=YES are processed, updated blocks are always written to disk. In the case of NK-ISAM files, writing to disk is set as the default processing mode. It is, however, also possible to operate without using disks.
- implicitly with SHARED-UPDATE=*YES in the ADD-FILE-LINK command when a file is opened for shared-update processing (for both K-ISAM and NK-ISAM). When K-ISAM files with SHARED-UPDATE=*YES are processed, updated blocks are always written to disk. In the case of NK-ISAM files, writing to disk is set as the default processing mode. It is, however, also possible to operate without using disks.

If the WROUT function is activated implicitly, it cannot be deactivated by the user (with the exception of NK-ISAM files), i.e. WROUT=NO in the FILE/FCB macro or WRITE-IMMEDIATE=*NO in the ADD-FILE-LINK command will have no effect.

7.5 Read-after-write check

A read-after-write check serves to detect recording errors when data is written to a disk. Because of the additional disk revolutions involved, this check significantly impairs system performance.

The user can activate a read-after-write check only for the current processing run, using either the WRCHK operand of the FILE/FCB macro or the WRITE-CHECK=*YES operand of the ADD-FILE-LINK command. This operand must be entered each time a read-afterwrite check is required, as it is not stored in the catalog entry.

7.6 File attributes for file backup

The archiving programs ARCHIVE and HSMS, used in regular save operations, create backup copies of the current files.

The user can define a “backup level” for each of his files with the aid of the operand of the same name in the CATAL macro or CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands.
Backup level “A”, for example, is the highest level and means that the file is saved with each save operation.
Backup level “E”, in contrast, means that the file is saved when the maximum backup class "E" is set in HSMS /ARCHIVE.
Whether or not the selected backup level (A/.../E) affects the frequency with which the file is saved depends on the backup procedures used in the computer center.

If the computer center executes partial save runs, the user can also specify the scope of the partial save for his/her files: Using the CATAL macro (LARGE operand) or the CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands (SAVED-PAGES operand), he/she specifies whether the complete file is to be saved in a partial save operation or – advisable for large files – a partial backup is sufficient.

The user can prevent HSMS from displacing a file to the background level by setting the file attribute MIGRATE to INHIBITED or FORBIDDEN. The HSMS administrator can bypass the INHIBITED setting, but not the FORBIDDEN setting.

The user can specify a file to be part of the version backup by setting a value > 0 for the NUM-OF-BACKUP-VERS file attribute. If the value is 0, the file is not part of the version backup. The value specifies the maximum number of file versions to be saved in the version backup archive. Temporary files, files on private disk, files on tape and file generations are not supported by the version backup, i.e. for these files the value can only be 0.
The setting is specified in the CATAL macro (NUM_OF_BACKUP_VERS operand) or in the CREATE-FILE and MODIFY-FILE-ATTRIBUTES (NUM-OF-BACKUP-VERS operand) commands. If no value is specified, the value from the NUMBACK system parameter is taken over for all files that support version backup.

More detailed information on file backups can be found in the manuals “ARCHIVE” [9 (Related publications)] and „HSMS“ [10 (Related publications)]; macros and commands for setting file attributes for file backup are described in the manuals “DMS Macros” [1 (Related publications)] and “Commands” [3 (Related publications)].

7.7 Pubset backup with Snapsets

A Snapset is the backup copy of an SF or SM pubsets. A Snapset serves as the pubset backup for restoring lost data (e.g. data deleted inadvertently).

Snapsets are created by systems support or the HSMS administrator during ongoing pubset operation and are deleted again later. The oldest Snapset can be implicitly deleted during the generation process.
The Snapsets can be used as a logical backup of all files and job variables of a pubset. The complete pubset can also be reset to the status at the time the Snapset was created.

Snapsets are placed in service when they are generated or when the pubset is imported and can then be accessed in read mode. They permit individual files and job variables to be restored to the status of the Snapset involved.

Snapsets can also be used in shared pubset mode.

Snapsets of a Pubset are not availiable temporary, if the subsystem SHC-OSD was not active at the moment of the pubset imports (particularly the snapsets of the home pubsets are not put automatically into operation when starting the system). As soon as SHC-OSD is active and the command SHOW-SNAPSET-CONFIGURATION is called, these snapsets are supplementaryput into operation .

Files on the Snapset

The CHECK-SNAPSET-CONFIGURATION command is used to check or activate the Snapset configuration of an imported pubset.

The LIST-FILE-FROM-SNAPSET command or LFFSNAP macro enables users to obtain information about files which have been backed up on a Snapset. They can obtain information about all files which they can access (as with SHOW-FILE-ATTRIBUTES, which provides information from the current file catalog).

Files backed up in this way can be restored from the Snapset using the RESTORE-FILE-FROM-SNAPSET command or RFFSNAP macro. During restoration, individual files are copied from the Snapset to the active pubset. This operation is comparable to an HSMS restore from a backup archive.

A particular backup version (the latest Snapset backup is preset) can be specified. Alternately, the latest version of the files can be restored on the basis of all existing Snapsets.

All file attributes of a restored file are taken over unchanged from the original file (including the creation and change dates and the protection attributes). Only the allocation can differ from the original file, also in the case of files with physical allocation. Files on SM pubsets are restored on the “most suitable” volume set. This need not be the same as the original volume set.

Individual file generations can only be restored with the entire file generation group. Files on private disk are not taken into account. In the case of migrated files and tape files, only the catalog entries are restored (without checking the availability of the associated tapes). Migrated files and tape files cannot be renamed when they are restored.

A nonprivileged user can restore a file from a foreign user ID only if they have read permission for the original file and are the owner of the target file.

Overwriting through restoration must be explicitly permitted for existing files (REPLACE operand). For files which are protected against unauthorized overwriting by means of a password, the required password must be entered in the caller’s password table (see the ADD-PASSWORD).

Files can also be restored under a new name (NEW-FILENAME operand). They are renamed by specifying either a different user ID or a file name prefix.

Files which were opened for writing when the Snapset was created can optionally be restored (RESTORE-OPEN-FILES operand). A file restored in this way has the same status as when the system crashed. Verification may be required for an ISAM file. When a Snapshot is created, the system files of SRPM, Guards and GCF, which are permanently open, are placed in a consistent status which permits them to be restored later.

If required, the caller can have a log of restore processing output to SYSOUT or SYSLST (OUTPUT operand). The log can cover either all files or only those files which, for particular reasons, could not be restored (REPORTING operand).

Job variables on the Snapset

The same mechanism is used for job variables as for files (see the section above).

The only differences are in the asasociated commands and/or operands:

- The LIST-JV-FROM-SNAPSET operand or LJFSNAP macro provides information about job variables which have been backed up on a Snapset (as with SHOW-JV-ATTRIBUTES, which provides information about job variables from the current file catalog).
- The RESTORE-JV-FROM-SNAPSET command or RJFSNAP macro restores job variables to the status of a particular Snapset or to the latest status on the basis of all existing Snapsets. Renaming takes place here using the NEW-JV-NAME operand. The other options for overwriting (REPLACE operand) and controlling log output (OUTPUT and REPORTING operands) are the same as for restoring files.

Displaying Snapsets

Users can obtain information on existing Snapsets (= existing backup versions of the pubset) using the SHOW-SNAPSET-CONFIGURATION command.

Output takes place to SYSOUT, or optionally also to SYSLST.
The creation date, the Snapset ID, the ID of the CCOPY session via which the Snapset can be accessed, the name of the assigned snap pool and the ID of the SRDF target disk storage system (among other things) can be output for each Snapset if snap copies were also created for the target units (see also the “SHC-OSD” manual [26 (Related publications)]).

Restoring library members

While a PLAM library is restored as a complete file using the RESTORE-FILE-FROM-SNAPSET command or the RFFSNAP macro, the PLAM library in LMS can be opened directly on the Snapset. This permits read access to the individual library members and consequently restoration on a member-by-member basis.

Incorporating files and job variables from Snapsets in the backup archive

The HSMS statement BACKUP-FILES enables files and job variables which have been backed up on a Snapset to be incorporated in a backup archive. As a result, files and job variables can practically be transferred “offline” to active pubset operation for long-term backup. For details, please see the “HSMS” manual [10 (Related publications)
].

8 Volume and device management

Which devices are available for user jobs?

User jobs can use all peripheral devices and system resources. System resources are:

- Consoles
- disk devices for pubsets
- devices which the operator has assigned to the system (e.g. printers).

As a rule, user jobs need to reserve only tape and disk devices. Other devices, such as printers, are served by the system and can be used via the SPOOL interfaces (using the PRINT-DOCUMENT command, PRNTDOC and WRLST macro).

8.1 Requesting volumes and devices

User jobs can request devices and volumes by means of the FILE macro and the commands CREATE-FILE, IMPORT-FILE, MODIFY-FILE-ATTIBUTES, ADD-FILE-LINK and SECURE-RESOURCE-ALLOCATION. In the case of tape files, volumes may also be requested implicitly during OPEN processing or automatic tape swapping.

By means of the CREATE-FILE and IMPORT-FILE commands, the user requests devices and volumes which are required for a specific file.

Using the SECURE-RESOURCE-ALLOCATION command, other resources such as files on public volumes can be reserved. If the SECURE-RESOURCE-ALLOCATION command is used to reserve a file which extends over several private volumes, DMS reserves all of the volumes and the necessary devices.

In a batch job, the SECURE-RESOURCE-ALLOCATION command causes the job to wait if one of the requested volumes is not free when the request is issued. The maximum wait time can be explicitly specified in the WAIT operand of the SECURE-RESOURCE-ALLOCATION command (for both batch and interactive mode).

Devices of a specific type can also be requested (in a CREATE-FILE, IMPORT-FILE or SECURE-RESOURCE-ALLOCATION command); the device manager then selects any free device of the specified type. Reservation by device type of disk devices in the SECURE-RESOURCE-ALLOCATION command can be used only for special applications (e.g. VOLIN).

A device can also be reserved via a mnemonic device name in the SECURE-RESOURCE-ALLOCATION command. NDM (Nucleus Device Management) reserves the device with the specified name if it is free. Once again, this is permitted only for special applications in the case of disk devices.

If all requested devices are free, they are reserved for the requesting job. If the user has also requested volumes, NDM sends a MOUNT message to the console, i.e. it requests the operator to mount the volumes. For files on private disks, all disks which contain parts of the file must be mounted.

If one or more of the requested devices is not free, no devices are reserved. In interactive mode, the job does not wait. In batch mode on the other hand, the job is entered in a queue (if the reservation was made using the SECURE-RESOURCE-ALLOCATION command) and is continued if all requested devices become free during the predefined wait period. If the devices were requested by means of the CREATE-FILE command, or if not all devices are available, a spin-off is initiated and DMS branches to the next SET-JOB-STEP or LOGOFF command.

8.2 Releasing devices and volumes

Volumes and devices can be released:

- when the job is terminated or aborted
- by means of RELEASE (RELTFT macro): the devices and volumes allocated to a file are released
- REMOVE-FILE-LINK command
- by means of the SECURE-RESOURCE-ALLOCATION command: when this command is entered, all previously reserved volumes and devices are released.

If a job is deactivated (WAIT-EVENT command) or terminated (LOGOFF command) or if a job is aborted, the reserved devices are released implicitly.

RELEASE

The devices and volumes allocated to a file are released and the entry in the TFT is deleted. If a HOLD lock exists, the RELEASE command is executed only after input of the DROP command.

If tapes or devices assigned to the file are also implicitly reserved for other files or tapes, respectively, they are released only when all reservations have been released.

The macro call RELTFT ...,KEEP releases the file and any related volumes, but retains the reserved device(s).

REMOVE-FILE-LINK

The devices and volumes allocated to a file are released. REMOVE-FILE-LINK deletes any existing entry in the TFT. The tapes and tape devices allocated to the file are being released. If a LOCK-FILE-LINK lock exists for the file, the REMOVE-FILE-LINK command is executed only after /UNLOCK-FILE-LINK.

If tapes or devices assigned to the file are also implicitly reserved for other files or tapes, respectively, they are released only when all reservations have been released.

The REMOVE-FILE-LINK command with RELEASE-DEVICE=NO releases the file and any related volumes, but retains the reserved device(s).

SECURE-RESOURCE-ALLOCATION

If no operands are specified in the SECURE-RESOURCE-ALLOCATION command (abbreviated to SEC-RES), all devices and volumes reserved for the job are released. If further devices or volumes are requested in the command, all currently reserved devices are released before the new ones are reserved, unless reservations exist for open files.

This means that all previous reservations are reset if a SEC-RES command is rejected. In the following situations, however, they will be retained:

- the rejection is due to a syntax error
- the rejection is due to a tape file being open
- the rejection is due to a TFT entry for the file being locked by a LOCK-FILE-LINK.

The SECURE-RESOURCE-ALLOCATION command will be rejected if a tape file is open or if a TFT entry for the file is locked with LOCK-FILE-LINK.

8.3 Operator intervention

Once a job has successfully requested devices and volumes, intervention by the operator becomes necessary: the volumes have to be mounted on the appropriate devices.

This activity is controlled by NDM (Nucleus Device Management), which issues messages asking the operator to mount the volumes on specific devices (MOUNT messages). Each of these messages contains the VSN of the volume and the mnemonic name of the device which has already been reserved for the requesting job.

The operator can mount the volume on the specified device or on another device of the same type, or he/she may reject the MOUNT request. The user receives an appropriate message on the terminal if the command MODIFY-JOB-OPTIONS LOGGING=
*PAR(LIST=*YES) was issued for the job.

8.4 Private disk management

The user sees only the reservation for the volume. The system manages the device needed to mount the volume. Devices cannot be reserved.

All reservation functions which result from DMS calls are interpreted as shareable reservation requests.

The SECURE-RESOURCE-ALLOCATION command permits shared or exclusive reservation of a private volume.

The control and supervision types for volumes can be set or modified with the aid of operator commands.

Device management offers automatic online supervision of the volumes used.

Volume supervision supports and monitors, amongst other things, the mounting of volumes and access to the volumes during file processing.

8.5 Tapes and tape devices

A prerequisite for any access to a tape file is that the associated volumes (magnetic tapes or magnetic tape cartridges (MTC)) and devices (tape or MTC drives) are available. Tapes and tape devices can be requested explicitly using the CREATE-FILE, ADD-FILE-LINK and SECURE-RESOURCE-ALLOCATION commands, the FILE macro or implicitly at OPEN time or, if the next tape is to be mounted on a different device, when tapes are swapped.

A tape request via the SECURE-RESOURCE-ALLOCATION or ADD-FILE-LINK command or by means of the FILE macro causes a PREMOUNT message to be sent to the console unless the tapes were reserved offline. A tape request at OPEN time causes a MOUNT message to be sent to the console. MOUNT=0 means that no PREMOUNT message is output on the console. The device is reserved offline. DMS issues the MOUNT message only when the device is actually required. The device is allocated to the requesting job.

By default, DMS does not issue MOUNT messages if the requested tapes are already mounted. In this case an implicit device allocation is made. For the processing of MF/MV sets, BS2000 issues PREMOUNT messages so that the operator can mount further tapes of the set in good time.

The operator can accept the MOUNT message for a specific device or he can suggest the use of another device. When he confirms that the volume has been mounted, the requested device is allocated to the job.

If devices are reserved implicitly (e.g. by specifying a device type), the volume can be moved to another device if a device error occurs on the first device. If the device was reserved explicitly, this is not possible.

Device management should be informed whenever a volume is to be moved to another device. The system then suggests which device to use. After the volume has been moved, device management carries out the necessary positioning of the tape so that it can be processed further.

8.5.1 Device reservation and allocation

Tape devices can be reserved or allocated by means of a macro or command.

Macros

The user can control when the devices are actually assigned:

- In the case of the FILE macro, the number of devices specified in the MOUNT operand are reserved (default value: one device is reserved). If MOUNT=0 is specified, no device is reserved. If necessary, this must be done at OPEN time
- Devices are allocated during OPEN processing if no device was previously allocated with FILE

Commands

Tape devices can be reserved by means of the ADD-FILE-LINK command (with the operands VOLUME-LIST=*CATALOG and VOLUME-LIST=*TAPE-SET), by a SECURE-RESOURCE-ALLOCATION command, or as part of the OPEN processing for tape files. The user can control when the devices are actually assigned:

- In the case of the ADD-FILE-LINK command, the number of devices specified in the NUMBER-OF-PREMOUNTS operand is allocated.
- In the SECURE-RESOURCE-ALLOCATION command, the user can also specify how many devices are to be reserved for the job: if implicitly requesting devices by means of the FILE operand, the user can enter MOUNT=n to specify how many devices are needed. If MOUNT=0 is specified, any necessary devices are reserved only when the file is opened.
- Devices are allocated during OPEN processing if no ADD-FILE-LINK or SECURE-RESOURCE-ALLOCATION command was issued previously, or if a device with an inappropriate volume type was allocated by a ADD-FILE-LINK or SECURE-RESOURCE-ALLOCATION command (in this case, a further device is allocated).

In the case of the MTC devices only identical specifications for the device type are compatible in the SECURE-RESOURCE-ALLOCATION and ADD-FILE-LINK commands.

A UNIT reservation has the disadvantage that the tape cannot be moved to a free tape device if a permanent hardware error occurs on the reserved device. Tape processing is terminated abnormally (SHOW-TAPE-STATUS shows ACTION=CANCELLED). It is therefore better to reserve devices either explicitly by specifying the device type or implicitly by reserving the required tape volumes.

If the requested volume is not mounted by the operator on the device reserved via UNIT, it must be moved to this device. If there was no UNIT reservation before the tape was requested, the tape can be mounted on any device which supports the requested recording density.

8.5.2 Ambiguous volume serial numbers

If several tapes with the same VSN are mounted concurrently in a computer center, the operator is asked via a user request to allocate the correct tape.

Device management guarantees that only one volume with a given VSN can be written at any one time. However, “SPECIAL” applications (INIT, VOLIN,...) may be running at the same time and may be using volumes with the same VSN. Device management has no way of protecting the user against mistakes by the operator; this means that a false tape with the specified VSN may be assigned by the operator after an INOP and that tape processing may be continued on a second volume.

Ambiguous VSNs may, for example, occur in a computer center which processes tapes belonging to different clients.

8.5.3 Tape swapping for multivolume files and the PREMOUNT message

The next tape of a multivolume file can be premounted on any physically compatible tape device unless a tape device was reserved via the command SECURE-RESOURCE-ALLOCATION UNIT=mn. In this case, the next tape must be processed on the same device as the preceding tape.

If DMS detects, as the result of a FILE macro, ADD-FILE-LINK command, or from the catalog entry for the file, that the file is stored on several tapes, it first processes the labels of the current tape and then informs the operator that the next tape in the sequence can be mounted.

If a tape change is necessary because the file is continued on the next tape or because the file was not found on the current tape of an MF/MV set, the current tape is rewound and unloaded. If the next tape is already mounted on another device, it is processed by DMS without operator intervention. This is also true for the first tape of a file if this tape was premounted.

If the next required tape is not already mounted, the operator receives a MOUNT message. The system suggests that the tape be mounted on the same device as the preceding tape. If this next tape is mounted on another device and this device is assigned, device management accepts this. In this case, the currently reserved device is released.

8.5.4 Switching tape drives in the case of a device error

If a tape was requested via its VSN, a device was requested via its device type, or both were requested implicitly by means of the file to be processed, the operator can move the tape to another device of the same type in the event of a device error.

If a device was reserved explicitly via its mnemonic device name (UNIT operand in the SECURE-RESOURCE-ALLOCATION command), the volume mounted on this device cannot be moved to another device.

In the case of a hardware fault, this means that tape processing is terminated abnormally if the device was reserved explicitly (with UNIT). The operator rejects the PREMOUNT message and the tape is implicitly canceled. This CANCEL state can be cleared only by releasing the tape reservation.

8.6 Output of information on resources

Device management permits the user to obtain, by means of the NKDINF macro and the commands listed below, information about the resources currently assigned to his job, namely devices, volume characteristics, reservations, etc.

Using the operands of the NKDINF macro, the user can restrict the information output to certain device families, etc. The RECORD operand also offers a DSECT for the output area for each of the other operands in this macro.

The table below indicates the various NDM commands which can be used to request specific information.

Command	Information on
SHOW-DEVICE-CONFIGURATION	System configuration
SHOW-DEVICE-DEPOT	Assignment of tape devices to depots
SHOW-DEVICE-STATUS	Device allocation and monitoring
SHOW-DISK-DEFAULTS	Default values for disk parameters
SHOW-DISK-STATUS	Disk allocations and parameters
SHOW-MOUNT-PARAMETER	MOUNT specifications
SHOW-RESOURCE-ALLOCATION	Resource allocations and outstanding operator actions
SHOW-RESOURCE-REQUESTS	State of device queue and collector task
SHOW-TAPE-STATUS	Tape allocation and monitoring

Table 25: Commands for displaying information on resources

The the "Commands" manual [3 (Related publications)] lists the meanings of the output columns produced by these SHOW commands.

9 Files on magnetic tape or MTC

9.1 Definitions

In this section, the term “tape” is used for both magnetic tapes and magnetic tape cartridges (MTC). In cases where tapes and cartridges have different characteristics, special reference is made to this fact.

9.1.1 File, file section and file set

A file section is that part of a file which is recorded on one tape.

If a complete file fits on one tape, it consists of only one file section. If a file consists of several sections, the first section of this file must be the last or only section on a tape, each intermediate section must be the only section on a further tape, and the last file section must be the first or only section on a tape. All file sections can be identified by means of their sequence numbers, which start with 1 (DIN 66029).

A file set consists of one or more files with a common file set identifier. The files of a file set can be identified by means of their sequence numbers, which start with 1. They may be recorded on one or more tapes (DIN 66029). The file set identifier is the volume serial number (VSN) of the first tape in the file set.

Furthermore, all files of a file set must have the same label and code characteristics and file protection attributes.

9.1.2 Volume set, volume series and tape owner

A volume set (or tape set) consists of the tapes on which a file set is stored. It contains precisely one file set (DIN 66029). A “single-tape file” can thus be regarded as a special “file set on one tape”.

A volume series is the set of tapes formed by the volume list in the catalog entry for a file. It is not necessarily identical with the tape set. The volume series or the volume list may also contain volumes which have not (yet) been used for the file. However, the file must begin on the first tape of the volume series. The tape sequence is the same as the sequence of file sections.

If the file is part of a file set, other files may also be stored on the first tape of the volume series; however, the file must begin on this tape.

The tape owner is the user under whose user ID the first file was written on the tape. When this first file is created, this user ID is written into the appropriate field of the VOL1 label as the “owner identifier”. All tapes of a volume set should have the same owner identifier in their VOL1 labels.

Shareability of tapes and tape files

The “shareability” of a tape depends on whether or not the first file on the tape is shareable. If a tape file is created by means of the FILE macro or CREATE-FILE command, it is implicitly cataloged as shareable (output field USER-ACC(ESS)=ALL-USERS). The tape is then also shareable. Tapes and tape files can be locked against access from other user IDs if the file is

- first cataloged using the CATAL macro (which implies USER-ACC=OWNER-ONLY) and the FILE macro is then issued with the operands DEVICE and VOLUME;
- cataloged using a FILE macro (operands DEVICE and VOLUME) and then made nonshareable by means of a CATAL macro (operands STATE=UPDATE and SHARE=NO). In this case the CATAL macro must be effective before the file is opened for the first time.
- cataloged using the CREATE-FILE command (entries USER-ACCESS=*OWNER-ONLY and SUPPORT=*NONE) before the MODIFY-FILE-ATTRIBUTES command with the VOLUME and DEVICE-TYPE operands is issued.
- cataloged using the CREATE-FILE command and then declared non-shareable with the MODIFY-FILE-ATTRIBUTES command via the USER-ACCESS=*OWNER-ONLY operand. The MODIFY-FILE-ATTRIBUTES command must take effect before the file is opened for the first time.

9.1.3 Scratch tape

A tape is regarded as a scratch tape if it has never been written (i.e. has been initialized but contains no files) or if the retention period for the first file on the tape has elapsed and write access is permitted.

On completion of processing, the tape is archived and the file on the tape has all the defined protection attributes.

9.2 Labels

9.2.1 DIN standards

The recording of data on magnetic tapes and cartridges is regulated by DIN standards. This allows the interchange of data between different systems.

DIN standards supported by BS2000:

- **DIN 66029, exchange levels 1-3**
  Labels and file arrangements on magnetic tapes for dataexchange
- **DIN 66229, exchange level**
  Labels and file arrangements on magnetic tape cartridges fordata exchange; track labels are not generated

The above standards deal only with the logical arrangement of files and labels on tapes, not with the physical recording methods and characteristics of tapes. The latter are defined in other standards. The standards DIN 66229 and DIN 66029 deal only with that part of the tape between the BOT (Beginning-Of-Tape) label and the last double tape mark which follows an end-of-file or end-of-volume label group.

The user may use the standard labels and/or his own labels for his tapes and tape files. (S)he can also specify that no labels are to be used.

The exchange levels of DIN 66029 describe the following restrictions:

- exchange level 1: a volume set contains a single file with fixed-length records
- exchange level 2: a volume set contains only files with fixed-length records
- exchange level 3: a volume set contains files with fixed-length or variable-length records

The DIN standards are completely satisfied by the access method SAM. The access methods BTAM and UPAM are block-oriented methods and do not therefore support record-by-record processing of files; furthermore, UPAM does not support tape swapping.

i The following descriptions refer to tapes and tape files with standard labels. Processing with nonstandard labels or without labels is described separately.

9.2.2 Label types (standard labels)

Position of the label	Label type	Label identifier	Permissible labels
Beginning of tape	Volume header label	VOL	VOL1
	User volume header label	UVL	UVL1 ... UVL9
Start of file or start of file section	File header label	HDR	HDR1 ... HDR3
	User file header label	UHL	UHL0 ... UHL255
End of file section (except for the last file section)	End-of-volume label	EOV	EOV1 ... EOV3
	User end-of-volume label	UTL	UTL0 ... UTL255
End of file or end of the last file section	End-of-file label	EOF	EOF1 ... EOF3
	User-trailer label	UTL	UTL0 ... UTL255

Table 26: Label types (standard labels)

9.2.2.1 System labels

By means of the FILE or FCB macro (LABEL operand) or via the ADD-FILE-LINK command (LABEL-TYPE operand), the user specifies that a tape file is to be created with standard labels, where the value specified is the exchange level in accordance with DIN 66029. figure 7 (Single-volume file: one file on one tape) shows the structure of a tape file with standard labels.

Volume header label (VOL1)

The first block on a tape is the volume header label VOL1; this may be used only after the BOT marker. The VOL1 label contains the volume serial number (VSN), the owner identifier and flags indicating whether or not the tape is shareable and the DIN standard with which the tape and file labels comply.

DIN 66029 permits up to 9 volume header labels, but BS2000 writes only VOL1. When reading tapes from non-BS2000 systems with multiple volume header labels, DMS ignores all such labels except VOL1.

For information on label processing see chapter “Label processing when opening tape files”.

End-of-volume labels (EOV)

For files which extend over several tapes, these labels identify the end of a tape (volume) and thus the end of a file section. The last data block of a file section is followed first by a tape mark, then by the end-of-volume labels, and finally by a double tape mark.

DIN 66029 permits up to 9 end-of-volume labels (EOV1, ..., EOV9), but BS2000 supports only a maximum of three (EOV1, EOV2, EOV3); only these three labels are written and the labels EOV4...EOV9 are ignored when a tape is read.

For information on label processing see chapter “EOV processing”.

File header labels (HDR)

Each file is preceded by file header labels. These identify the file and contain the file attributes, i.e. their function is similar to that of the entry in the F1 label of a private disk.

DIN permits up to 9 file header labels (HDR1, ..., HDR9), but BS2000 supports only a maximum of three (HDR1, HDR2, HDR3). Only these three labels are written and the labels HDR4...HDR9 are ignored when a tape is read.

The file header labels are always followed by a tape mark. If a file extends over several tapes (multivolume file), each file section begins with file header labels, which follow the volume header label.

For information on label processing see chapter “Label processing when opening tape files”.

End-of-file labels (EOF)

The last data block is always followed by a tape mark and then the end-of-file labels which, like the file header labels, identify the file and are used to monitor access to the file.

DIN 66029 permits up to 9 end-of-file labels (EOF1, ..., EOF9), but BS2000 supports only the labels EOF1-EOF3; only these labels are written and the labels EOF4...EOF9 are ignored when a tape is read.

The end-of-file labels are followed by a tape mark if there are more files on the tape. If this is the last file on the tape, the end-of-file labels are followed by a double tape mark.

For information on label processing see chapter “CLOSE processing”.

9.2.2.2 User labels

User labels must be written and evaluated by the user program. User program routines for reading and writing user labels are started via the exits OPENV, LABGN, LABEOV and LABEND of the EXLST macro.

In all cases, the user label groups follow the corresponding system label groups. Just like the system labels, they can be identified by the label names in the first byte of the label: UVL, UHL and UTL. However, there are only three types of user labels: volume labels (UVL), file header labels (UHL) and general trailer labels (UTL).

User volume labels (UVL)

The user may write up to 9 user volume labels (UVL1,...,UVL9). These are placed between the VOL1 label and the HDR labels. The program routine for reading or writing UVL labels is activated via the EXLST exit OPENV.

User file header labels (UHL)

The user may write up to 256 user file header labels (UHL0, ..., UHL255), which must follow the last HDR label. The program routine for reading or writing UHL labels is activated via the EXLST exit LABGN.

User trailer labels (UTL)

User end-of-volume and file trailer labels have the same format. The user end-of-volume labels follow the EOV labels and the user file trailer labels follow the EOF labels. The user may write up to 256 end-of-volume and 256 file trailer labels (UTL0, ..., UTL255). The appropriate program routines are activated automatically via the EXLST exits LABEND (for end-of-file) and LABEOV (for end-of-volume).

9.2.2.3 Sequence of labels and label groups

System and user labels within a tape or a file must always appear in a specific order. Tape marks are not permitted within label groups. Each label group must be located completely on the tape containing the first label of this group.

Each label group begins with the label with the number 1 (HDR1, EOF1, EOV1, UVL1), except for UHL (UHL0) and UTL (UTL0). This first label is followed by all other labels of the same group; these must be arranged in ascending order of the label numbers, with no gaps in the sequence.

The number of end-of-file and end-of-volume labels (EOF/EOV) must be equal to the number of file header labels (HDR).

If a tape or a file (or a volume or file set) contains both system labels and user labels, each user label group follows the appropriate system label group.

In the case of user labels, the number of user trailer labels (UTL) is not determined by the number of user file header labels (UHL).

9.2.3 Nonstandard labels and tape files without labels

BS2000 also permits the processing of tapes and tape files which do not contain standard labels. This is also controlled by the FILE and FCB macros (LABEL operand) and by the ADD-FILE-LINK command (LABEL-TYPE operand). The specifications LABEL=NO and LABEL-TYPE=*NO mean that the file to be processed has no labels, while LABEL=NSTD and LABEL-TYPE=*NON-STD mean that the file has labels, but these are not standard labels. Such nonstandard labels must be written and evaluated by special EXLST routines in the user program; the user may define labels of any format and type.

9.3 Arrangement of files on magnetic tapes and tape cartridges

The “classic” example of a tape file is where one tape contains one file. However, BS2000 also supports processing of multivolume files and multifile tapes (= file sets) and of MF/MV (= multifile/multivolume) sets, where a file set is stored on a tape set. The following description deals only with files with system (standard) labels; user labels are not described.

The arrangement of files on both magnetic tapes and tape cartridges is governed by the same rules.

With respect to the arrangement of files on a tape, the file management functions of BS2000 prepare for and monitor the creation, access to and extension of files. By means of the FILE, FCB and CATAL macros, and the commands ADD-FILE-LINK, CREATE-FILE and MODIFY-FILE-ATTRIBUTES, the user can control the file management functions to meet his/her needs.

DMS supports the user in the management of

- one file on one tape,
- one file on several tapes,
- several files on one tape,
- several files on several tapes,
- different versions of a file, and
- file generations.

9.3.1 Single-volume file: one file on one tape

A single-volume file is a file which is stored on a single tape (volume).

Structure of a single-volume file

The volume header label VOL1 is followed by the file header labels, which are separated from the first data block by a tape mark. The data blocks may have the standard format (PAM page: 2048 bytes; with or without a PAM key) or a nonstandard format (no PAM key, variable block size; see section “Block formats for tape files”). The end-of-file labels are also separated from the data blocks by a tape mark. The end of the tape is always indicated by a double tape mark.

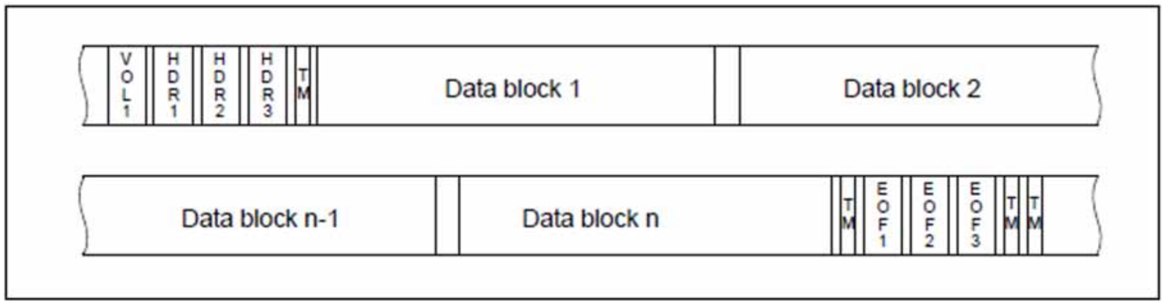


Figure 7: One file on one tape

Creating a single-volume file

In the FILE macro, the DEVICE and VOLUME operands define the device type for the volume and the volume itself. In the CREATE-FILE command this is done by means of the DEVICE-TYPE and VOLUME operands. An entry is made in the task file table by means of the LINK operand in the FILE macro or the LINK-NAME operand in the ADD-FILE-LINK command, as appropriate. The user can request a specific volume (VOLUME=vsn) or allow the system operator to allocate a volume (VOLUME=PRIVATE or VOLUME=*ANY). If the user makes no further entries for LABEL (or LABEL-TYPE) etc., a standard tape file is created, currently in accordance with DIN 66029, exchange level 3. In other words the file receives, by default, three HDR labels (HDR1-3) and three EOF labels (EOF1-3).

If the user wishes to use his/her own labels, he/she must define the appropriate label routines in the EXLST macro. The system branches automatically to these routines after it has written the preceding system labels.

With respect to all other file attributes which are defined by means of a macro or in the program (access method, record format, etc.), the rules for tape files are the same as for disk files, except that ISAM processing of tape files is not possible.

9.3.2 Multivolume file: one file on several tapes

A multivolume file is a file which extends over several tapes (volumes). The tapes form a so-called “volume set”. Multivolume files are stored on volume sets. In the case of files on tape cartridges, a volume set consists of several cartridges.

Structure of a multivolume file

For multivolume files with standard labels, each tape begins with the VOL1 label, followed by the HDR labels and a tape mark. Then comes the first section of the file, terminated by a further tape mark. Instead of the EOF labels, EOV labels indicate the end of the tape and are written automatically by the system when a tape swap is performed.

On the second tape, the field “File section number” in the HDR1 label contains the value “0002”, since the HDR labels precede the second file section. If further tapes are used for the file, EOV and HDR labels with the current file section numbers are written when the tapes are swapped.

The last file section is then terminated with EOF labels.

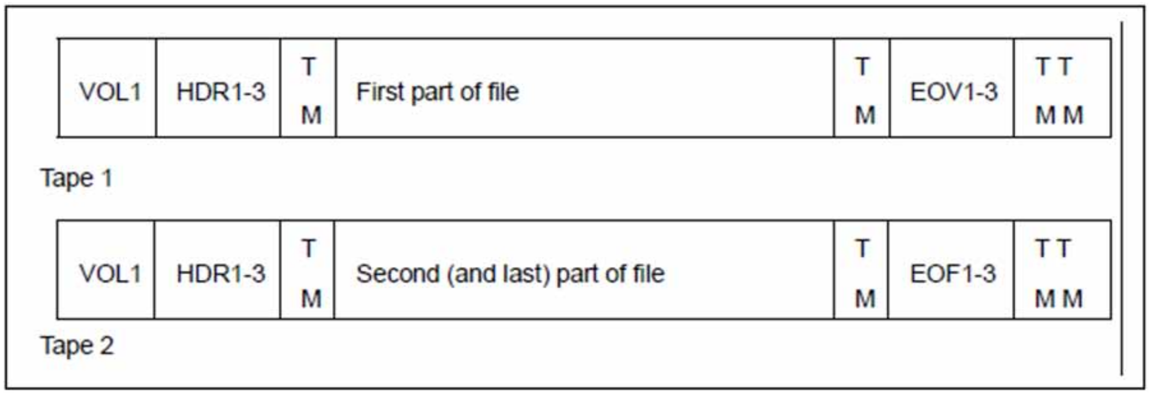


Figure 8: Multivolume file

Creating a multivolume file

The user can specify in the FILE macro or CREATE-FILE command that he wants to create a multivolume file. He uses the DEVICE or DEVICE-TYPE operand, as appropriate, to specify whether the file is to be written on a tape or a tape cartridge. The VOLUME operand can be used to specify how many volumes are likely to be required for the file, together with the VSNs of these volumes.

The list of specified VSNs is transferred to the catalog entry, where it forms the volume list.

The user can specify in the ADD-FILE-LINK command (PROCESS-VOLUME operand) a list of VSNs for input files which have already been cataloged. This means that the volume labels entered in the catalog are ignored during processing, and only the volumes specified in the command are used. The catalog entry itself is not, however, modified.

The TAPE-SET-NAME operand of the ADD-FILE-LINK command anchors a volume list created in the “tape set table” (TST) with the CREATE-TAPE-SET command in the TFT entry. This TST is then updated during processing and, once processing has been concluded, evaluated to form the volume list in the catalog entry.

If the volume list specifies several volumes, the user can use the MOUNT operand of the FILE macro or the PREMOUNT-LIST operand of the CREATE-FILE command to specify how many volumes are to be “immediately” available, i.e. how many devices he wants to use. If he asks for two devices, he can use these two devices alternately during file processing, thus reducing the time involved in tape swapping considerably.

If the user's program does not contain routines for writing user labels, the system handles the writing of all labels and controls tape swaps. The LABEL operand of the FILE or FCB macro and the LABEL-TYPE operand of the ADD-FILE-LINK command determine whether standard labels are written. A detailed description of tape swapping can be found in chapter “EOV processing”.

9.3.3 File set: several files on one tape

If several files are written on one tape, they form a file set. The special case “file set on volume set” (= MF/MV set) is described in section “MF/MV set: file set on volume set” and section “Processing of MF/MV sets”).

Structure of a file set

A tape with standard labels starts with the volume header label VOL1, followed by the file header labels of the first file and a tape mark. Then come the data blocks of this first file, followed by a tape mark and the EOF labels.

As there are more files on the tape, the EOF labels are followed only by a single tape mark, which precedes the HDR labels of the next file, etc. The EOF labels of the last file are then followed by the double tape mark. The position of the file in the file set is recorded in the field “File sequence number” (FSEQ) in the HDR1 and EOF1 labels of each file.

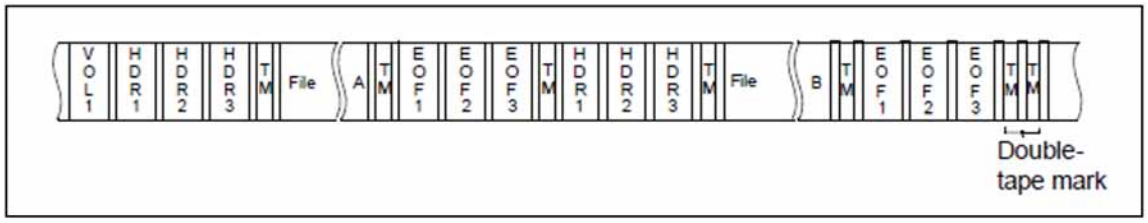


Figure 9: File set

Creating a file set

By means of the operands DEVICE and VOLUME of the FILE macro and the DEVICE-TYPE and VOLUME operands of the CREATE-FILE command, the user specifies the volume on which he wants to store his file. If a tape is used for several files, each file is usually written from the beginning of the tape, which means that any file already on the tape is overwritten.

In order to create a file set, the files must be written one after the other on the tape. This is controlled by the FSEQ operand in the FILE or FCB macro and by the FILE-SEQUENCE operand of the ADD-FILE-LINK command. FSEQ and FILE-SEQUENCE, as appropriate, allocate a position in the file set to each file; the files must be numbered in ascending order without gaps in the sequence. This also makes it possible to write files with the same name on one tape, as they can be identified unambiguously by the operand.

If the user does not know the actual sequence number of a file, he can have the system search for an input file with FSEQ=UNK (= unknown) in the macro or FILE-SEQUENCE=UNKNOWN in the command or can write an output file at the end of the set by specifying FSEQ=NEW or FILE-SEQUENCE=NEW, as appropriate.

9.3.4 MF/MV set: file set on volume set

A multifile/multivolume set (= MF/MV set) is the result of storing a file set on a volume set. The term “MF/MV set” refers only to the logical relationship between the file set and the volume set. DMS does not maintain a volume/file catalog.

Structure of an MF/MV set

The first file of the file set begins on the first tape of the volume set.

The file set identifier is kept in the HDR1 label of each file. It is the volume serial number (VSN) of the first tape which contains the first or only section of the first file in the file set - this means: the file set identifier of all files in the file set is the VSN of the first tape in the associated volume set.

Example: MF/MV set

The following files are recorded in the file catalog:

```
File A        (FSEQ=1)
               Volume series: 000001, 000002
File B        (FSEQ=2)
               Volume series: 000001, 000002, 000003, 000004
File C        (FSEQ=3)
               Volume series: 000004, 000005
File D        (FSEQ=4)
               Volume series: 000005, 000006, 000007, 000008
```

When these files were created, the following volume sets were specified in the related CREATE-FILE and ADD-FILE-LINK commands:

```
File A :  000001, 000002
File B :  000001, 000002, 000003, 000004
File C :  000003, 000004, 000005, 000006
File D :  000004, 000005, 000006, 000007, 000008
```

The file set consists of files (A,B,C,D). The associated volume set contains the tapes: (000001, 000002, 000003, 000004, 000005, 000006, 000007, 000008).

Explanation to the figure

(n,n) = (file sequence number, file section)

- File C begins on volume 000004 with an empty file section (see figure 10 (Empty file sections)).

- File D ends on volume 000008 with an empty file section.

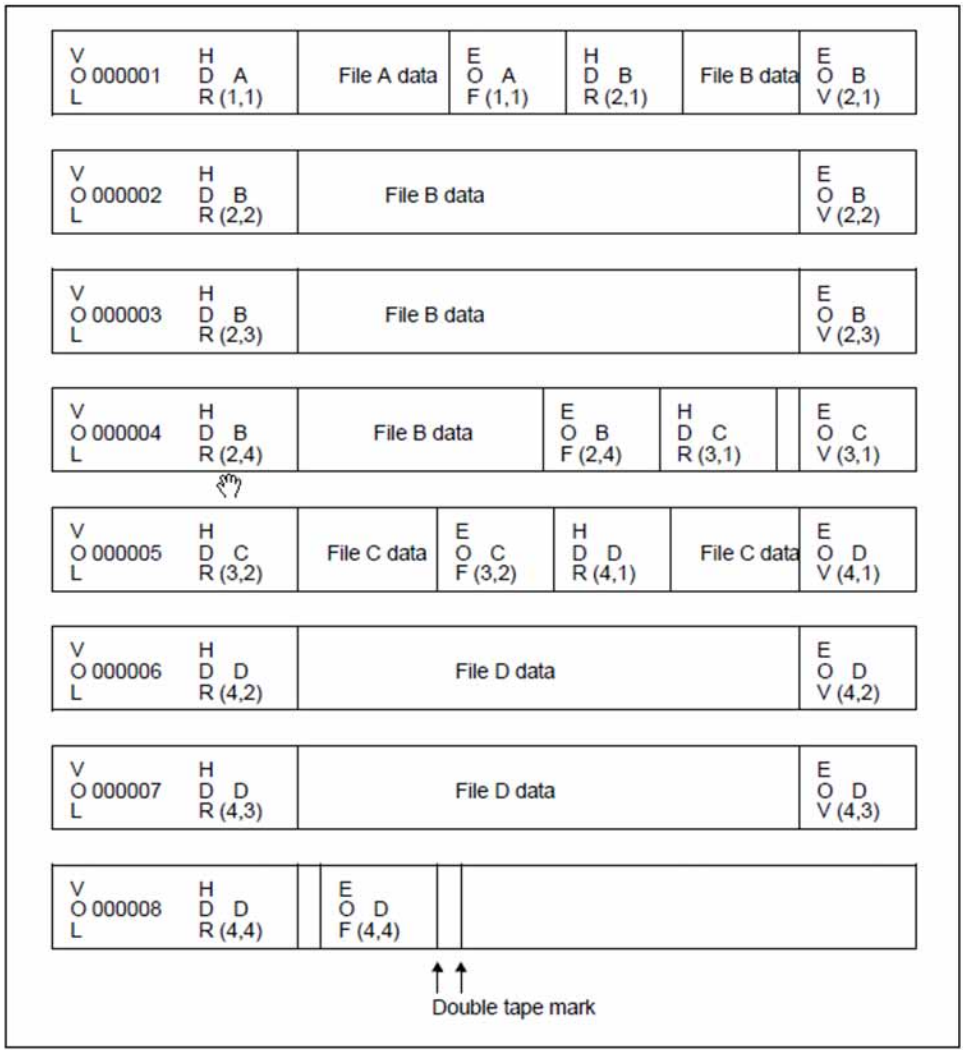


Figure 10: MF/MV set

Creating an MF/MV set

An MF/MV set is created just like a file set, except that it extends over several volumes. To achieve this, the operands DEVICE, VOLUME and FSEQ must be specified in the FILE macro and one FILE macro must be called for each file in the file set. In the CREATE-FILE command, the DEVICE-TYPE and VOLUME operands must be supplied with values and in the ADD-FILE-LINK command the FILE-SEQUENCE operand. Both files must be entered for every file in the set.

The DEVICE operand of the macro and the DEVICE-TYPE operand of the command specify which type of device (and thus which type of volume) the user wants to use. In the VOLUME operand, the user lists the VSNs of all of the tapes on which he wants to store the files. The FSEQ operand of the macro and the FILE-SEQUENCE operand of the command specify the position of each file in the file set. Starting with FSEQ/FILE-SEQUENCE=1, there must be no gaps in the sequence of numbers and the files must be created in the same order. If the user specifies FSEQ=NEW in the FILE macro or FILE-
SEQUENCE=NEW in the ADD-FILE-LINK command, BS2000 creates the files at the end of the file set and assigns the next number to each of these. This also makes it unnecessary for the user to make this operand a variable in a procedure or a program.

The following information is transferred to the catalog entry of the file:

- the volume series, i.e. the volume list from the VOLUME operand, starting with the volume on which the file begins;
- the FSEQ or FILE-SEQUENCE number which the user specified in FILE/FCB macro or ADD-FILE-LINK command, as appropriate, or which was allocated by the system (if FSEQ=NEW or FILE-SEQUENCE=NEW was specified).

The system writes the standard labels automatically (depending on the LABEL operand in the FILE/FCB macro or ADD-FILE-LINK command) and also automatically executes tape swaps.

If the user specifies a volume set in the VOLUME operand of the FILE macro or CREATE-FILE command, these volumes must be free.

The label level of the first and all following volumes of the volume set is defined via the LABEL operand of the FILE /FCB macro or ADD-FILE-LINK command, as appropriate, when the first file of the file set is created. If a file is to be added to the file set, it must be created with labels of the same label level, as recorded in the volume header label (VOL1). If the file extends over several volumes, the following volumes are created with the same label level.

9.3.5 Empty file sections

Under certain circumstances, empty file sections may occur at the start or end of a tape in an MF/MV set, e.g. if end-of-tape is detected while the labels are being written. DMS then initiates a tape swap and finishes writing the labels on the next tape.

Empty file section at the start of a tape

If the system detects the end of the tape when it is writing the last data block of a file or the end-of-file labels, it terminates the tape with end-of-volume labels and a double tape mark. After the next tape has been mounted, the system writes the volume header label, followed by the file header labels of the file which was not completed, a double tape mark and, finally, the end-of-file labels.

If this file is the last file of a file set, a double tape mark is then written to terminate the set. If further files follow in the set, only a single tape mark is written, followed by the HDR labels of the next file.

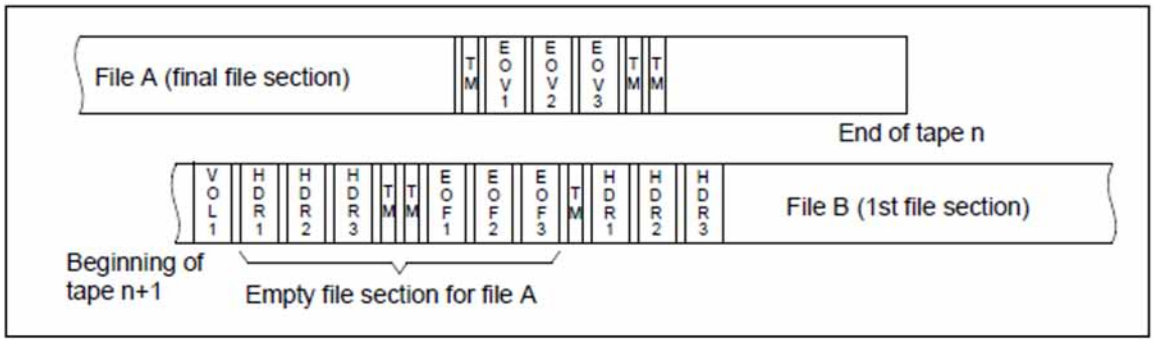


Figure 11: Empty file section at the start of a tape

Empty file section at the end of a tape

If the system detects the end-of-tape mark while writing the file header labels for a file, it writes the end-of-volume labels, enclosed between two double tape marks, immediately after these file header labels.

The next tape starts with the volume and file header labels, the file section number in the HDR1 label being incremented by 1. This is followed by a tape mark and then the first section of the file.

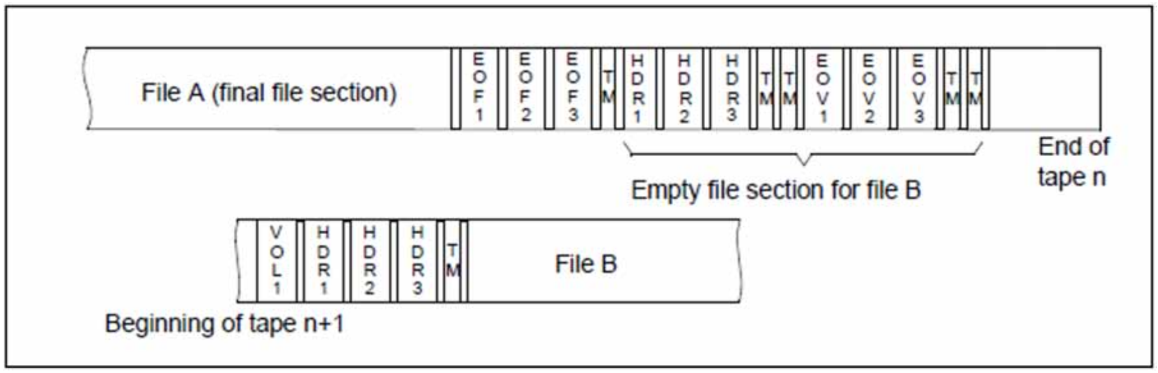


Figure 12: Empty file section at the end of a tape

9.4 Processing of file sets

Overwriting a file set

A tape can be overwritten only if its retention period has elapsed and write access is permitted.

If the user wishes to overwrite a file, DMS positions the tape in accordance with the file sequence number. File protection attributes such as passwords, etc. are evaluated only for cataloged files.

i If a file within a file set is overwritten or extended, access to the following files in the file set is no longer possible! The rest of the tape is implicitly erased.

If the file specified in the FILE/FCB macro or ADD-FILE-LINK command is cataloged, the file sequence number in the FSEQ or FILE-SEQUENCE operand must match that in the catalog entry. If the file is not yet cataloged, DMS positions the tape on the basis of the FSEQ/FILE-SEQUENCE specification and the file can then be overwritten without checking the file name.

If the file is not cataloged and the FSEQ/FILE-SEQUENCE specification is 1 greater than the highest existing file sequence number in the file set, the file set is extended by one file.

Extending a file set

In order to extend a file set by one file, the user must specify either FSEQ=NEW in the FILE/FCB macro, or FILE-SEQUENCE=NEW in the ADD-FILE-LINK command, or a number which is 1 greater than the last file sequence number in the file set. The file must not be cataloged before this is done.

If FSEQ=NEW/FILE-SEQUENCE=*NEW is specified, DMS positions the tape to the end of the file set. If this lies within the volume set specified in the FILE macro or ADD-FILE-LINK command, the new file is written to the tape with a file sequence number which is 1 greater than that of the current last file in the set.

If SECLEV=OPR is specified in the FILE/FCB macro, the security level of the new file must be less than or equal to that of the preceding file. The same applies if the operand OVERWRITE-PROTECTION=*YES is specified in the ADD-FILE-LINK command.

Adding several files to a file set (program interface)

If the user is certain that all of the files will fit on one tape, he/she can use the following sequence of commands and program statements:

```
         FILE   AAA,VOLUME=000001,FSEQ=1,DEVICE=TAPE,LINK=OUTP1
         OPEN   OUT1,OUTPUT
         .
         .
         CLOSE  OUT1,LEAVE
         FILE   BBB,VOLUME=000001,FSEQ=NEW,DEVICE=TAPE,LINK=OUTP2
         OPEN   OUT2,OUTPUT
         .
         .
         CLOSE  OUT2,LEAVE
         FILE   CCC,VOLUME=000001,FSEQ=NEW,DEVICE=TAPE,LINK=OUTP3
         OPEN   OUT3,OUTPUT
         .
         .
         CLOSE  OUT3,LEAVE
         .
         .
OUT1     FCB   LINK=OUTP1...
OUT2     FCB   LINK=OUTP2...
OUT3     FCB   LINK=OUTP3...
```

If the size of a file in the file set (e.g. the first file) cannot be predicted, the user could extend the volume sets in the FILE macros for files AAA, BBB and CCC as follows:

`VOLUME=(000001,000002)`

If file AAA extends onto the second tape, DMS requests the first tape again when it creates files BBB and CCC in order to determine the end of the file set. If file AAA is so large that file BBB does not fit on the second tape, DMS requests a further free tape. After this, however, OPEN processing for file CCC will result in an error because the file set ends on a tape which was not specified in the FILE macro and is thus not part of the volume set.

However, the user can use the TSET operand of the FILE macro to temporarily link the file and volume sets. In the TSET operand, the user enters the name of a “tape set”, i.e. a temporary volume list which is maintained in the TST (tape set table). A TST entry is a table in which the VSNs of the tapes requested for a job are stored – linked via a TSET name.

In the following example, the file set AAA, BBB and CCC is again created.

```
START
        LDBASE 10
        USING *,10
        .
         FILE   AAA,LINK=OUTP1,VOLUME=000001,DEVICE=TAPE,FSEQ=1,TSET=SET1
         OPEN   OUT1,OUTPUT
         .
         .
         CLOSE  OUT1,LEAVE
         FILE
BBB,LINK=OUTP2,DEVICE=TAPE,FSEQ=NEW,TSET=SET1,VOLUME=000002
         OPEN   OUT2,OUTPUT
         .
         .
         CLOSE  OUT2,LEAVE
         FILE   CCC,LINK=OUTP2,DEVICE=TAPE,FSEQ=NEW,TSET=SET1
         OPEN   OUT2,OUTPUT
         .
         .
         CLOSE  OUT2,LEAVE
         .
         .
OUT1     FCB    LINK=OUTP1...
OUT2     FCB    LINK=OUTP2...
OUT3     FCB    LINK=OUTP3...
```

All files refer to the same TST entry and the files and the TST entry are linked via the TFT (task file table), which is created by means of a LINK operand for each file. DMS automatically organizes the structure of the file set on the common volume set.

The TST entry is set up if the LINK operand is specified in the FILE macro and, at the same time, a new TST name is specified in the TSET operand. If the TSET specification is repeated in further FILE macros for other files, no new TST entry is created. Instead, the corresponding TFT entry simply receives a pointer to the TST entry and the file counter in the TST entry is incremented.

When tapes are swapped, DMS updates a pointer in the TST entry which points to the VSN of the tape containing the (current) end of the file set. It is thus unnecessary to search through the whole volume set for the end of the file set when a new file is added, and this reduces the number of tape swaps.

If the user is working with a TSET, FSEQ=1 must be specified when the first file of a file set is created.
For further files of the file set, it is permissible to list in the VOLUME operand the VSNs of tapes which are unknown in the TST entry (see the examples below).
Several TSTs can be created in one job.

Example 1

Requesting a specific tape as the first tape of a volume set:

```
FILE AAA,LINK=OUTP,DEVICE=TAPE,VOLUME=000001,FSEQ=1,TSET=T
```

Example 2

Requesting any free tape:

```
FILE AAA,LINK=OUTP,DEVICE=TAPE,FSEQ=1,TSET=T[,VOLUME=PRIVATE]
```

Example 3

A file set is created as follows:

```
FILE A,LINK=A,DEVICE=TAPE,VOLUME=(VOL1,VOL2),FSEQ=1,TSET=SET1
```

The file set is now to be extended:

```
FILE C,LINK=C,DEVICE=TAPE,FSEQ=NEW,TSET=SET1,VOL=VOL3
```

Adding several files to a file set (command interface)

If certain that all of the files in a set will fit on one tape, the user can execute the following sequence of commands and program statements:

Command	Program
CREATE-FILE FILE-NAME=A,SUPPORT=TAPE(VOLUME=000001, DEVICE-TYPE=xyz)	Create output file A
ADD-FILE-LINK LINK-NAME=A,FILE-NAME=A, SUPPORT=*TAPE(FILE-SEQUENCE=1)	Open output file A : Close file A
CREATE-FILE FILE-NAME=B,SUPPORT=TAPE(VOLUME=000001,DEVICE-TYPE=xyz)	Create output file B
/ADD-FILE-LINK LINK-NAME=B,FILE-NAME=B, SUPPORT=*TAPE(FILE-SEQUENCEC=*NEW)	Open output file B : Close file B
CREATE-FILE FILE-NAME=C,SUPPORT=TAPE(VOLUME=000001, DEVICE-TYPE=xyz)	Create output file C
ADD-FILE-LINK LINK-NAME=C,FILE-NAME=C, SUPPORT=*TAPE(FILE-SEQUENCE=*NEW)	Open output file C : Close file C

Table 27: Extending a file set (Example 1)

If the size of a file in the file set cannot be predicted, the user could extend the volume sets in the CREATE-FILE command for files A, B and C as follows:

```
VOLUME=(000001,000002)
```

If file A extends onto the second tape, DMS requests the first tape again when it creates files B and C, in order to determine the end of the file set. If file A is so large that file B does not fit on the second tape, DMS requests a further scratch tape. After this, however, OPEN processing for file C will result in an error, because the file set ends on a tape which was not specified in the CREATE-FILE command and is thus not part of the volume set.

However, the user can link the file and volume sets temporarily by using the TAPE-SET-NAME operand of the ADD-FILE-LINK command. The user enters the name of a “tape set”, i.e. a temporary volume list which is maintained in the TST (tape set table), via the TAPE-SET-NAME operand. The TST is a table in which the VSNs of the tapes requested for a job are stored – linked via a TSET name.

In the following example, the file set A, B and C is again created.

Command	**Program**
CREATE-TAPE-SET TAPE-SET-NAME=SET1, VOLUME=000001	Set up tape set SET1
CREATE-FILE FILE-NAME=A,SUPPORT=TAPE(DEVICE-TYPE=xyz)	Create output file A

ADD-FILE-LINK LINK-NAME=A,FILE-NAME=A,
SUPPORT=*TAPE(VOLUME-LIST=*TAPE-SET(
TAPE-SET-NAME=SET1),FILE-SEQUENCE=1)
	Open output file A
	:
	Close file A

CREATE-FILE FILE-NAME=B,SUPPORT=TAPE(DEVICE-TYPE=xyz)	Create output file B

ADD-FILE-LINK LINK-NAME=B,FILE-NAME=B,
SUPPORT=*TAPE(VOLUME-LIST=*TAPE-SET(
TAPE-SET-NAME=SET1),FILE-SEQUENCE=*NEW)
	Create output file B
	:
	Close file B

CREATE-FILE FILE-NAME=C,SUPPORT=TAPE(DEVICE-TYPE=xyz)	Create output file C

ADD-FILE-LINK LINK-NAME=C,FILE-NAME=A,
SUPPORT=*TAPE(VOLUME-LIST=*TAPE-SET(
TAPE-SET-NAME=SET1),FILE-SEQUENCE=*NEW)
	Open output file C
	:
	Close file C

Table 28: Extending a file set (Example 2)

When a file is closed, the tape must be positioned to the end of the file.

All the files refer to the same TST entry and the files and the TST entry are linked via the TFT (Task File Table), which is created by means of a LINK-NAME operand for each file. DMS automatically organizes the structure of the file set on the common volume set.

The TST entry is constructed with the CREATE-TAPE-SET/EXTEND-TAPE-SET commands. If the TAPE-SET-NAME specification is repeated in ADD-FILE-LINK commands, no new TST entry is created. Instead, the corresponding TFT entry receives a pointer to the TST entry and the file counter in the TST entry is incremented.

When tapes are swapped, DMS updates a pointer in the TST entry which points to the VSN of the tape which contains the (current) end of the file set. It is thus unnecessary to search through the whole volume set for the end of the file set when a new file is added, and this reduces the number of tape swaps.

i If the user is working with a TSET, FILE-SEQUENCE=1 must be specified when the first file of a file set is created. For further files of the file set, it is permissible to list the VSNs of the tapes which are unknown in the TST entry in the VOLUME operand (see the example below). Several TSTs can be created in one job.

Example: creating a file set

By means of the following commands, the user requests a specific tape as the first tape of the volume set:

```
/CREATE-TAPE-SET TAPE-SET-NAME=T,VOLUME=000001
/CREATE-FILE FILE-NAME=A,SUPPORT=TAPE(DEVICE-TYPE=xyz)
/ADD-FILE-LINK LINK-NAME=A,FILE-NAME=A,SUPPORT=*TAPE(VOLUME-LIST=
 *TAPE-SET(TAPE-SET-NAME=T),FILE-SEQUENCE=1
```

The user requests any scratch tape with the following command:

```
/CREATE-FILE FILE-NAME=A,SUPPORT=TAPE(VOLUME=*ANY,DEVICE-TYPE=xyz)
```

The file set can be extended by the commands:

```
/CREATE-FILE FILE-NAME=B,SUPPORT=TAPE(DEVICE-TYPE=xyz)
/ADD-FILE-LINK LINK-NAME=B,FILE-NAME=B,SUPPORT=*TAPE(VOLUME-LIST=
 *TAPE-SET(TAPE-SET-NAME=T),FILE-SEQUENCE=*NEW
```

Example: extending a file set

A file set was created as follows:

```
/CREATE-TAPE-SET TAPE-SET-NAME=SET1, VOLUME=(000001,000002)
/CREATE-FILE FILE-NAME=A,SUPPORT=TAPE(DEVICE-TYPE=xyz)
/ADD-FILE-LINK LINK-NAME=A,FILE-NAME=A,SUPPORT=*TAPE(VOLUME-LIST=TAPE-SET
 (TAPE-SET-NAME=SET1),FILE-SEQUENCE=1)
```

The file set is now to be extended:

```
/CREATE-TAPE-SET TAPE-SET-NAME=SET2, VOLUME=000003
/CREATE-FILE FILE-NAME=C,SUPPORT=TAPE(DEVICE-TYPE=xyz)
/ADD-FILE-LINK LINK-NAME=C,FILE-NAME=C,SUPPORT=*TAPE(VOLUME-LIST=TAPE-SET
 (TAPE-SET-NAME=SET2),FILE-SEQUENCE=*NEW)
```

9.5 Processing of MF/MV sets

File access in an MF/MV set

If the user wishes to access a cataloged file within an MF/MV set, (s)he only needs to specify the file name in the FILE/FCB macro or the ADD-FILE-LINK command. DMS then takes the associated VSNs and the file sequence number from the catalog entry. When the file is opened, DMS positions the first tape in the volume list to the start of the file, checks the file header records and, if permitted, then allows access to the file. Label checking and tape positioning are described in detail in chapter “Label processing when opening tape files”.

If the user wishes to access an existing, but uncataloged, file (= foreign file), he/she can use the macros to specify the following:

- the file name, the operand STATE=FOREIGN, the file sequence number (FSEQ=) and the volume list (in the VOLUME operand) in the FILE macro.
- If he does not know the file sequence number, he can specify the file name and the volume list, together with the operand FSEQ=UNK. DMS then searches for the file in the specified volume set, using the file name as the search argument. A prerequisite for this is, however, that the foreign file was created with standard labels.

If using commands, the user must first import the file with the IMPORT-FILE command. He/she can then specify the following:

- the file name and the file sequence number (FILE-SEQUENCE= operand) in the ADD-FILE-LINK command.
- If he does not know the file sequence number, he can specify the file name together with the operand “FILE-SEQUENCE=UNKNOWN”. DMS then searches for the file in the specified volume set, using the file name as the search argument. A prerequisite for this is, however, that the foreign file was created with standard labels.

If the FILE macro does not contain a VSEQ specification or the ADD-FILE-LINK command does not include a specification for the VOL-SEQUENCE-NUMBER operand, DMS assumes that the user wants the first section of the file.

Access in REVERSE mode

For access in REVERSE mode, DMS requests the volume which contains the last file section and positions the tape to the end of this section. DMS uses the number of volumes recorded in the catalog entry in order to determine which volume in the volume set contains the end of the file. If the file is not cataloged (foreign file), DMS assumes that the last volume in the volume list contains the end of the file.

If the user specifies only one volume in the FILE macro (VSEQ operand) or ADD-FILE-LINK command (VOL-SEQUENCE-NUMBER operand), DMS requests this volume since the file section number and the tape sequence number correspond. It then positions the tape to the end of the file section.

Access to any desired file section

If the user wishes to process a file which extends over several tapes, he/she can use the FILE macro (VSEQ operand) or ADD-FILE-LINK command (VOL-SEQUENCE-NUMBER operand) to position the file to any desired file section. The user program can then process either this file section or all file sections from this file section to the end of the file.

If the program is to process only one file section, the user must specify the VSEQ operand as follows:

```
VSEQ=(L=(n))
```

If the user program is to process all file sections from the specified section to the end of the file, the VSEQ operand must be specified as follows:

```
VSEQ=n
```

where “n” is the number of the volume which contains the first file section to be processed.

Random access to file sections

File sections of a file can be processed in any desired order. To do so a list of file section numbers in the VSEQ or VOL-SEQUENCE-NUMBER operand must be specified. Each of the entries in this list refers to a volume of the volume set which contains the file.

Example 1 (program interface)

File A extends over the tape volumes 00000A, 00000B, 00000C, 00000D, 00000E. If the user specifies the VSEQ operand:

```
VSEQ=(L=(4,1,3))
```

and if the user program completely processes each specified file section, DMS requests that volumes 00000D, 00000A and 00000C be mounted in this order.

Random access to file sections is particularly important if sections of a multivolume file have been destroyed or are unreadable and the user wants to process those sections of his file which can still be read.

Example 2 (program interface)

If, in the above example, the second section of file A had been destroyed, the user could specify the following to process the file sections which are still accessible:

```
VSEQ=(L=(1,3,4,5)
```

Example 3 Processing a file section by section (command interface)

File A extends over the tape volumes 00000A, 00000B, 00000C, 00000D, 00000E. If the user issues the following ADD-FILE-LINK command

```
/ADD-FILE-LINK FILE-NAME=A,
     SUPPORT=*TAPE(VOLUME-LIST=*CATALOG(VOL-SEQUENCE-NUMBER=(4,1,3))
```

and if the user program completely processes each specified file section, DMS requests that the volumes 00000D, 00000A and 00000C be mounted in this order.

Example 4 Access to file sections (command interface)

If, in the above example, the second section of file A had been destroyed, the user could issue the following command to enable the remaining file sections to be processed:

```
/ADD-FILE-LINK FILE-NAME=A,
     SUPPORT=*TAPE(VOLUME-LIST=*CATALOG(VOL-SEQUENCE-NUMBER=(1,3,4,5))
```

Extending files in an MF/MV set

The rules for extending files in an MF/MV set are the same as for extending files in a file set:

i If files in the MF/MV set or file sections of such files are extended, the following files are implicitly overwritten and the file set is destroyed.

The catalog entries of the overwritten files are not automatically erased, which means that they point to non-existent files. Any attempt to access a file which has been implicitly overwritten will lead to an OPEN error. It is the responsibility of the user to delete such catalog entries.

This implicit overwriting may destroy the nth file section of a multivolume file. However, by specifying VSEQ=n+1 in the FILE macro or START-POSITION=n+1 in the ADD-FILE-LINK command, the user can still access the following section of this file.

All files of a file set which are stored before the overwritten or extended file on a volume, and all files on the other volumes of a volume set, remain accessible.

The overwritten/extended file becomes the last file of the file set.

The original volume set no longer exists. The new volume set consists of the volumes of the old volume set up to and including the volume on which the file was overwritten or extended. Any attempt to position from this new volume set to a volume in the “remainder” of the original volume set will lead to unpredictable results.

Implicit overwriting of a file is prevented if the preceding file has a higher security level (retention period, access type). If the SECLEV operand is specified with the option OPR in the FILE/FCB macro or the OVERWRITE-PROTECTION operand is specified in the ADD-FILE-LINK command, a file cannot be created with a higher security level than that of its predecessor in the file set.

Positioning MF/MV sets

A prerequisite for access to a file in an MF/MV set is that the volume set is first positioned to the desired file. This positioning is carried out automatically for cataloged files with standard labels or with no labels when the file is to be opened. In the case of uncataloged files or files with nonstandard labels, it is the user's responsibility to ensure that positioning is correct.

Positioning is based on the file sequence number, which is specified by FSEQ or FILE-SEQUENCE, as appropriate. This sequence number is allocated to each file of an MF/MV set when the file is created. If a file is continued on the next volume, it retains the same file sequence number. The next higher sequence number is allocated to the file which follows it on the next volume. The file sequence number of a file is recorded in the HDR1 label and in the catalog entry.

If a user wants to access a cataloged file, DMS automatically fetches the file sequence number and the VSNs of the volumes containing the file from the file catalog entry.

For access to an uncataloged file, or when creating a new file, the user must specify the FSEQ operand in the FILE /FCB macro or the FILE-SEQUENCE operand in the ADD-FILE-LINK command. Otherwise, DMS assumes that the user wishes to access the first file of a file set.

When a file on a tape is opened, the current file sequence number is taken from the FILE or FCB macro, from the ADD-FILE-LINK command, or from the catalog entry.

The tape is then positioned until

- the sequence number of the file being read matches the requested sequence number, or
- if FSEQ=NEW/FILE-SEQUENCE=NEW is specified, the end of the MF/MV set is found, namely a double tape mark following an EOF label group; the tape is then moved one tape mark backwards, or

- if FSEQ=UNK/FILE-SEQUENCE=UNKNOWN is specified, the file name is found.

If necessary, DMS automatically has the tapes swapped and carries out label checking (see chapter “Label processing when opening tape files”).

If a file is to be opened in REVERSE mode, the search is executed as described above, but the tape is in each case positioned to the end of the file.

The user must provide special EXLST routines for positioning tapes which use nonstandard labels.

9.6 Processing of non-EBCDIC tapes

Various codes can be used for writing and reading magnetic tapes. The data and the labels of tape files are processed in the specified code. The code is a file attribute. Labels are also checked in the specified code.

The following operands in the FILE/FCB macro or the ADD-FILE-LINK command specify the code for SAM/BTAM:

CODE = EBCDIC	No code conversion
CODE = ISO7	ISO 7-bit code (128 text characters)
CODE = OWN	All other codes

In EBCDIC and ISO7 code, the national and international character sets are encoded with the same bit combinations (double assignment).

If the CODE operand is not specified in the FILE/ FCB macro or in the ADD-FILE-LINK command, the specification is taken from the catalog entry for the file or from the tape label.

As a rule, the codes EBCDIC and ISO7 are recognized. If CODE=OWN is entered, labels are recognized only if a conversion table is provided in the user program. Separate conversion tables (256 bytes each) must be provided for writing and reading. The addresses of the conversion tables for reading and writing are specified in the FCB with the operands TRTADR (TRanslate Table ADdress Read) and TRTADW (TRanslate Table ADdress Write).

The user can also specify that tape files in any code are to be read directly (without conversion, as for EBCDIC).

Once the code has been specified, the user does not need to worry about the details of code conversion. He can assume that the data traffic with the tape file will be executed just as for an EBCDIC file.

9.7 Use of MTC systems

BS2000 supports magnetic tape cartridge (MTC) systems with different data formats (number of tracks) and recording methods (with and without compression). The types supported in this version are described in the Release Notice for BS2000/OSD-BC.

The volume type determines the data format (HSI) and thus implicitly selects the MTC device. With device type 3590E, the volume type can additionally be used to determine the recording method. The volume type table is contained in the "System Installation" manual [15 (Related publications)].

As one would expect, files generated on any of these MTC devices cannot simply be read and further processed on one of the other device types without further ado.For this to be possible, systems support must comply with the guidelines and recommendations outlined below.

Basic prerequisites from the point of view of the system

“Compression” as a volume attribute

In the same way as the data format, “compression” is regarded as a volume attribute, i.e. the files on a tape cartridge are recorded in either compressed or non-compressed form. However, in this context users should bear in mind that all labels are always written without compression, including those belonging to files which contain compressed data.

Compression and data format as customer options

The system does not automatically set a data format and a recording method. It is systems support's responsibility to set the desired values for the system (by means of an appropriate parameter specification in the FILE macro with parameter DEV or CREATE-FILE or IMPORT-FILE command with parameter DEVICE-TYPE).

Internal system considerations mean that the only parameter available for these entries in the FILE macro and CREATE-FILE command is the existing DEV and DEVICE-TYPE parameter, respectively. As a result this therefore defines both the device to be used and the recording method. The full responsibility for the use of the DEV parameter rests with the user. It may be necessary to modify this parameter in existing procedures.

There is no generic term for “MTC” (without defining further options).

Consequently, the user should exercise due caution when specifying the DEV or DEVICE-TYPE parameter, to ensure that his files are processed exactly as required.

Possible entries for the DEV parameter

A number of cases must be distinguished:

Creating a new file as the first file in a multifile configuration

The DEV or DEVICE-TYPE value specified by the user determines the device to be used as well as the desired recording method.

Processing (reading) an existing file

Two cases must be distinguished when reading a file:

1. There is a catalog entry. In this case, the volume type must be stored in the catalog entry. The system selects a suitable device and the file is read in the correct mode.

2. There is no catalog entry, i.e. the file is treated as a foreign file. In this case, it is the user's responsibility to ensure that the correct device is selected in the DEV or DEVICE-TYPE parameter. In other words, the user has to know precisely how the data on his magnetic tape cartridges was recorded.

In the case of files with standard labels, DMS checks the compatibility of specifications in the FILE macro or IMPORT-FILE command, or of catalog entries with the values stored in the labels of existing files. Whether or not a file (and thus the entire volume) is compressed is indicated in bytes 34 and 35, which are reserved for BS2000, in the HDR2 (and EOF2) labels.

In the event of a discrepancy between the values entered by the user or stored in the catalog and the information in the label, the system rejects the job.

Extending files and creating continuation files

When extending a partly existing file by means of OPEN EXTEND or INOUT, a check as to whether the entry from the file label matches the desired processing mode ensures that any new data written to the file is likewise compressed.

If a continuation file is created on a multifile tape, the first file on the tape determines the only possible processing mode. The entry in the label of the first file serves to ensure that compression is treated as a volume attribute.

If there are any incompatibilities, the system rejects the job.

Note

All checks performed by DMS can only refer to tapes created with standard labels and supplied with the entry “BS2000” in the HDR2 label. As the compression indicator in HDR2 is not contained in a standard label field, unambiguous identification of the compression status is not possible for tapes not produced by Fujitsu Technology Solutions (or for tapes with nonstandard labels or without labels). When working with tapes without standard labels, it is the user's task to ensure that the correct compression mode is used.

10 File generation groups (FGGs)

Chronologically related files with the same file attributes can be managed in BS2000 in socalled file generation groups (FGGs). The files belonging to an FGG are called “file generations” or simply “generations”.

File generation groups are intended for the storage of large amounts of data for relatively long periods, e.g. the data resulting from regularly executed jobs, and the book-keeping data stored in tape pools. They also offer a convenient method of saving data based on the grandfather-father-son principle.

Executable programs or procedures cannot be stored in file generation groups. If they are stored as file generations, they must be transferred to “normal” files before they can be executed.

The group name is subject to the same rules as a file name in BS2000 (see section “File names”), with the difference that the maximum length of the name is restricted to 34 characters, since a generation number is added to the file name to identify the generations. File generations can thus be addressed via the group name and the appropriate generation number.

```
Group name:       [:catid:][$userid.]filename

Generation name:  [:catid:][$userid.]filename(generationnumber)
```

10.1 Generation numbers

The generation numbers reflect the chronological relationship between the generations, since they are the “consecutive numbers” allocated as the generations are created.

The catalog entry for a file generation always contains its “absolute generation number”. This generation number is incremented by 1 as each new generation is created. The upper limit for this number is 9999, and the next generation is then cataloged with the generation number 0001. An asterisk “*” before the generation number indicates that this is an absolute generation number; leading zeros may be omitted.

```
Absolute generation number: (*number); 0001 <= number <= 9999
```

In order to avoid the need to update the absolute generation number in programs or procedures which process or create file generations, a base for the use of relative generation numbers is defined in the catalog entry for the file generation group. The user may specify any existing generation as the base for relative indexing and may change this base at any time by means of the BASE operand of the CATAL macro or the BASE-NUMBER operand of the CREATE-FILE-GROUP and MODIFY-FILE-GROUP-ATTRIBUTES commands. The absolute generation number which is the current reference point is stored in the field “BASE-NUM” of the group entry in the catalog. DMS recognizes relative generation numbers by the fact that they start with the sign “+” or “-” instead of the asterisk “*” used for an absolute generation number.

```
Relative generation number: (+number) or (-number); 0 <= number <= 99
```

Example: allocating relative and absolute generation numbers

A file generation group GROUP is to consist of the cataloged generations 20 to 23; the catalog entry contains the information BASE=21. This results in the following relationships between relative and absolute generation numbers:

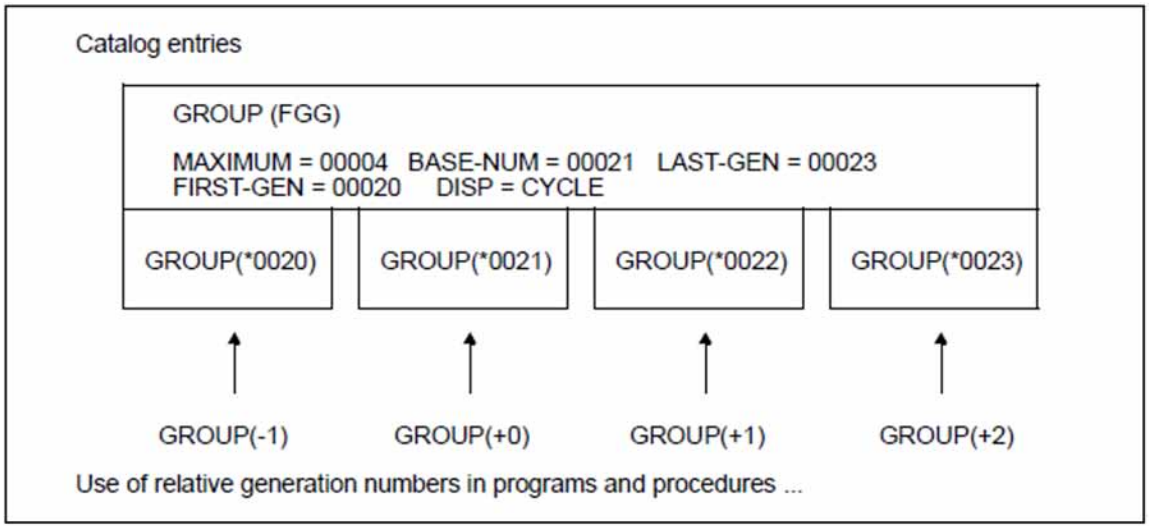

Catalog entries
GROUP (FGG)
MAXIMUM = 00004 BASE-NUM = 00021 LAST-GEN = 00023
FIRST-GEN = 00020 DISP = CYCLE
GROUP(*0020)
GROUP(*0021)
GROUP(*0022)
GROUP(*0023)
GROUP(-1)
GROUP(+0)
GROUP(+1)
GROUP(+2)
Use of relative generation numbers in programs and procedures ...

Figure 13: Absolute and relative generation numbers

10.2 Creating a file generation group

When file generation groups or file generations are cataloged – either during creation or when they are imported – care must be taken that the FGG is always completely cataloged in the related pubset. That means there must be no gaps in the sequence of cataloged generations. File generations may also be cataloged simultaneously on several pubsets if, for example, they are imported into several pubsets at the same time- but they must again be completely cataloged in each pubset.

File generations and file generation groups can also be backed up on a Snapset. Individual file generations can only be restored from a Snapset with an entire file generation group.

10.2.1 Creating a group entry

File generation groups are managed with the aid of their group entries. This means that this group entry must be created before file generations can be set up.

The group entry for a file generation group can be created only by means of the CATAL macro or CREATE-FILE-GROUP command. For attributes which are not specified by operands of the macro or command during creation of the group, DMS inserts default values in the group entry. The user can modify the attributes defined in the group entry at any time – at least for disk files.

The DISP operand of the CATAL macro and the OVERFLOW-OPTION operand of the CREATE-FILE-GROUP and MODIFY-FILE-GROUP-ATTRIBUTES commands define what DMS is to do with the “superfluous” old generations when the maximum number of simultaneously cataloged generations is exceeded. The user can, for example, specify that the volumes containing these superfluous generations are to be used for storing new generations. If these volumes are private disks, the new generation is created before the old generation is erased. If storage space is limited, this may mean that the new generation cannot be created, since there is (at the moment) not enough space available.

By default, the catalog entry for the oldest existing generation is deleted.

The group entry created by means of the macro or command defines the save and file protection attributes for all generations of this file generation group. The protection attributes BACL and GUARDS protect both the FGG and each individual file generation – with the exception of tape generations. A caller without write access right cannot create any new file generation in an FGG that is write-protected by these attributes.

The encryption attributes for file encryption are defined uniformly for all generations of a group via the group entry. Here the tape generations are always excluded from file encryption. Only the entire group, not an individual generation, can be converted into encrypted files using ENCRYPT-FILE (see "Preconditions and restrictions for file encryption"). New generations (with the exception of tape generations) are assigned the encryption attributes of the group.

Attribute	**Operand in the CATAL macro**	**Operand in the CREATE-FILE-GROUP and MODIFY-FILE-GROUP-ATTRIBUTES commands**
Shareability	SHARE=NO/YES /SPECIAL BASACL, GUARDS	USER-ACCESS=*OWNER-ONLY/ *ALL-USERS/ *SPECIAL BASIC-ACL, GUARDS
Write access permission	ACCESS=WRITE /READ	ACCESS=*WRITE/*READ
Passwords	RDPASS, WRPASS	READ-PASSWORD, WRITE-PASSWORD
Frequency and scope of automatic file backup	BACKUP, LARGE	BACKUP-CLASS, SAVED-PAGES
Automatic data destruction when storage space is released	DESTROY=NO /YES	DESTROY-BY-DELETE

Monitoring of DMS accesses	AUDIT	AUDIT

Table 29: Attributes for file generations

Further fields in the group entry refer to the file generation group as an whole:

- the maximum number of simultaneously cataloged generations (MAXIMUM)
- the base value for relative generation numbers (BASE-NUM)
- the absolute generation numbers of the newest and oldest cataloged generations (LAST-GEN, FIRST-GEN)

The remaining fields in the group entry refer only to accesses to this catalog entry (ACC-COUNT, CRE-DATE, EXPIR-DATE, CHANG-DATE, VERSION).

With respect to file protection attributes, the catalog entries for the file generations are identical with the group entry. The other attributes (HIGH-US-PA, CRE-DATE, etc.) may, however, differ from one generation to another.

Example: group entry and catalog entries for file generations

```
0000000000 :2OS6:$ULR.MAX.GROUP.2 (FGG)
   ------------------------------- HISTORY      -----------------------------
   CRE-DATE   = 2014-10-12  ACC-DATE   = NONE        CHANG-DATE = NONE
   CRE-TIME   =   14:08:14  ACC-TIME   = NONE        CHANG-TIME = NONE
   ACC-COUNT  = 0           S-ALLO-NUM = 0
   ------------------------------- SECURITY     -----------------------------
   READ-PASS  = YES         WRITE-PASS = NONE        EXEC-PASS  = NONE
   USER-ACC   = ALL-USERS   ACCESS     = WRITE       ACL        = NO
   AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = 2014-10-12
   DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME =   00:00:00
   SP-REL-LOCK= NO          ENCRYPTION = *NONE
   ------------------------------- BACKUP       -----------------------------
   BACK-CLASS = A           SAVED-PAG  = COMPL-FILE  VERSION    = 0
   MIGRATE    = ALLOWED
   #BACK-VERS = 0
 ------------------------------- GENERATION-INFO -------------------------
   MAXIMUM    = 3           BASE-NUM   = 11          OVERFL-OPT = CYCL-REPL
   FIRST-GEN  = 11          LAST-GEN   = 13
 0000000024 :2OS6:$ULR.MAX.GROUP.2(*0011)
   ------------------------------- HISTORY      -----------------------------
   CRE-DATE   = 2014-10-12  ACC-DATE   = 2014-10-12  CHANG-DATE = 2014-10-12
   CRE-TIME   =   14:11:44  ACC-TIME   =   14:11:44  CHANG-TIME =   14:11:44
   ACC-COUNT  = 1           S-ALLO-NUM = 0
   ------------------------------- SECURITY     -----------------------------
   READ-PASS  = YES         WRITE-PASS = NONE        EXEC-PASS  = NONE
   USER-ACC   = ALL-USERS   ACCESS     = WRITE       ACL        = NO
   AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = 2014-10-12
   DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME =   00:00:00
   SP-REL-LOCK= NO          ENCRYPTION = *NONE
   ------------------------------- BACKUP       -----------------------------
   BACK-CLASS = A           SAVED-PAG  = COMPL-FILE  VERSION    = 1
   MIGRATE    = ALLOWED
   #BACK-VERS = 0
   ------------------------------- ORGANIZATION -----------------------------
   FILE-STRUC = PAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
   IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
   AVAIL      = *STD
   WORK-FILE  = *NO         F-PREFORM  = *K          S0-MIGR    = *ALLOWED
   ------------------------------- ALLOCATION   -----------------------------
   SUPPORT    = PUB         S-ALLOC    = 34          HIGH-US-PA = 4
   EXTENTS     VOLUME     DEVICE-TYPE      EXTENTS     VOLUME     DEVICE-TYPE
       2       6VS1.0       D3435
   NUM-OF-EXT = 2
0000000024 :2OS6:$ULR.MAX.GROUP.2(*0012)
   ------------------------------- HISTORY      ----------------------------
   CRE-DATE   = 2014-10-12  ACC-DATE   = 2014-10-12  CHANG-DATE = 2014-10-12
   CRE-TIME   =   14:16:00  ACC-TIME   =   14:16:00  CHANG-TIME =   14:16:00
   ACC-COUNT  = 1           S-ALLO-NUM = 0
   ------------------------------- SECURITY     ----------------------------
   READ-PASS  = YES         WRITE-PASS = NONE        EXEC-PASS  = NONE
   USER-ACC   = ALL-USERS   ACCESS     = WRITE       ACL        = NO
   AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = 2014-10-12
   DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME =   00:00:00
   SP-REL-LOCK= NO          ENCRYPTION = *NONE
   ------------------------------- BACKUP       ----------------------------
   BACK-CLASS = A           SAVED-PAG  = COMPL-FILE  VERSION    = 1
   MIGRATE    = ALLOWED
   #BACK-VERS = 0
   ------------------------------- ORGANIZATION ----------------------------
   FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
   IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
   REC-FORM   = (V,N)       REC-SIZE   = 0
   AVAIL      = *STD
   WORK-FILE  = *NO         F-PREFORM  = *K          S0-MIGR    = *ALLOWED
   ------------------------------- ALLOCATION   ----------------------------
   SUPPORT    = PUB         S-ALLOC    = 9           HIGH-US-PA = 1
   EXTENTS     VOLUME     DEVICE-TYPE      EXTENTS     VOLUME     DEVICE-TYPE
      2       6VS1.0       D3435
   NUM-OF-EXT = 2
 0000000024 :2OS6:$ULR.MAX.GROUP.2(*0013)
   ------------------------------- HISTORY      ----------------------------
   CRE-DATE   = 2014-10-12  ACC-DATE   = 2014-10-12  CHANG-DATE = 2014-10-12
   CRE-TIME   =   14:16:04  ACC-TIME   =   14:16:05  CHANG-TIME =   14:16:05
   ACC-COUNT  = 1           S-ALLO-NUM = 0
   ------------------------------- SECURITY     ----------------------------
   READ-PASS  = YES         WRITE-PASS = NONE        EXEC-PASS  = NONE
   USER-ACC   = ALL-USERS   ACCESS     = WRITE       ACL        = NO
   AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = 2014-10-12
   DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME =   00:00:00
   SP-REL-LOCK= NO          ENCRYPTION = *NONE
   ------------------------------- BACKUP       ----------------------------
   BACK-CLASS = A           SAVED-PAG  = COMPL-FILE  VERSION    = 1
   MIGRATE    = ALLOWED
   #BACK-VERS = 0
   ------------------------------- ORGANIZATION ----------------------------
   FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
   IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
   REC-FORM   = (V,N)       REC-SIZE   = 0
   AVAIL      = *STD
   WORK-FILE  = *NO         F-PREFORM  = *K          S0-MIGR    = *ALLOWED
------------------------------- ALLOCATION   ----------------------------
   SUPPORT    = PUB         S-ALLOC    = 9           HIGH-US-PA = 1
   EXTENTS     VOLUME     DEVICE-TYPE      EXTENTS     VOLUME     DEVICE-TYPE
       2       6VS1.1       D3435
   NUM-OF-EXT = 2
:2OS6: PUBLIC:      4 FILES RES=       72 FRE=       66 REL=       60 PAGES
```

As defined in the group entry, all generations of this FGG are shareable and protected by a read password. However, they differ with regard to the attributes which are not defined in the group entry: ACC-COUNT, CRE-DATE, CHANG-DATE, EXPIR-DATE.

10.2.2 Creating file generations

File generations are cataloged just like “normal” files with the CATAL macro or the CREATE-FILE-GROUP command. File processing – in this case: creation of the file – is likewise the same as “normal” file processing. The only difference is the format of the file name, which must contain an absolute or relative generation number. When cataloging or creating file generations, the user must ensure that there are no gaps in the sequence of absolute generation numbers, regardless of whether (s)he uses absolute or relative generation numbers in his/her programs or procedures. The group entry is updated automatically by DMS.

The following attributes can be assigned (and modified) separately for individual file generations of an FGG: STOCLAS, IOPERF, IOUSAGE, DISKWR, S0MIGR, AVAIL, USRINFO, ADMINFO (with macros) and STORAGE-CLASS, PERFORMANCE, USAGE, DISK-WRITE, S0-MIGRATE, AVAILABILITY, USER-INFORMATION, FILE-PREFORMAT, ADM-INFORMATION (with command).

10.3 Deleting file generations or file generation groups

Just as for the creation of generations, deleting a generation must leave no gaps in the sequence of absolute generation numbers. Thus, starting from the generation to be deleted (the POS operand in the ERASE macro; DELETE operand in the DELETE-FILE-GENERATION command), only all older or all newer generations can be deleted.

If the current base generation for relative generation numbers is deleted, the generation specified in the ERASE macro or DELETE-FILE-GENERATION command becomes the new base generation.
The fields in the group entry are updated accordingly (BASE-NUM, FIRST-GEN, LAST-GEN).

File generation groups can be deleted by specifying the group name in the ERASE macro or DELETE-FILE-GROUP command. The group entry and all related file generations which are currently cataloged are erased.

In the ERASE macro (VERSION=1 / 2 / 3), the user may also specify the selection operand TYPE=FGG or wildcards in the file name in order to erase several file generation groups at the same time.

The user may also specify wildcards in the file name in the DELETE-FILE-GROUP command in order to erase several file generation groups at the same time.

Example: deleting generations

Initial situation: GROUP1 consists of generations 6 to 9, BASE-NUM = 7

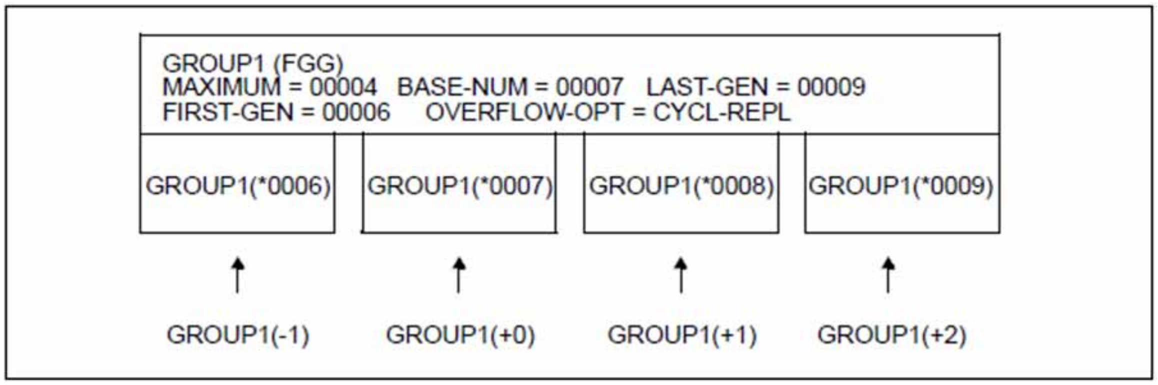


Figure 14: Deleting file generations (1)

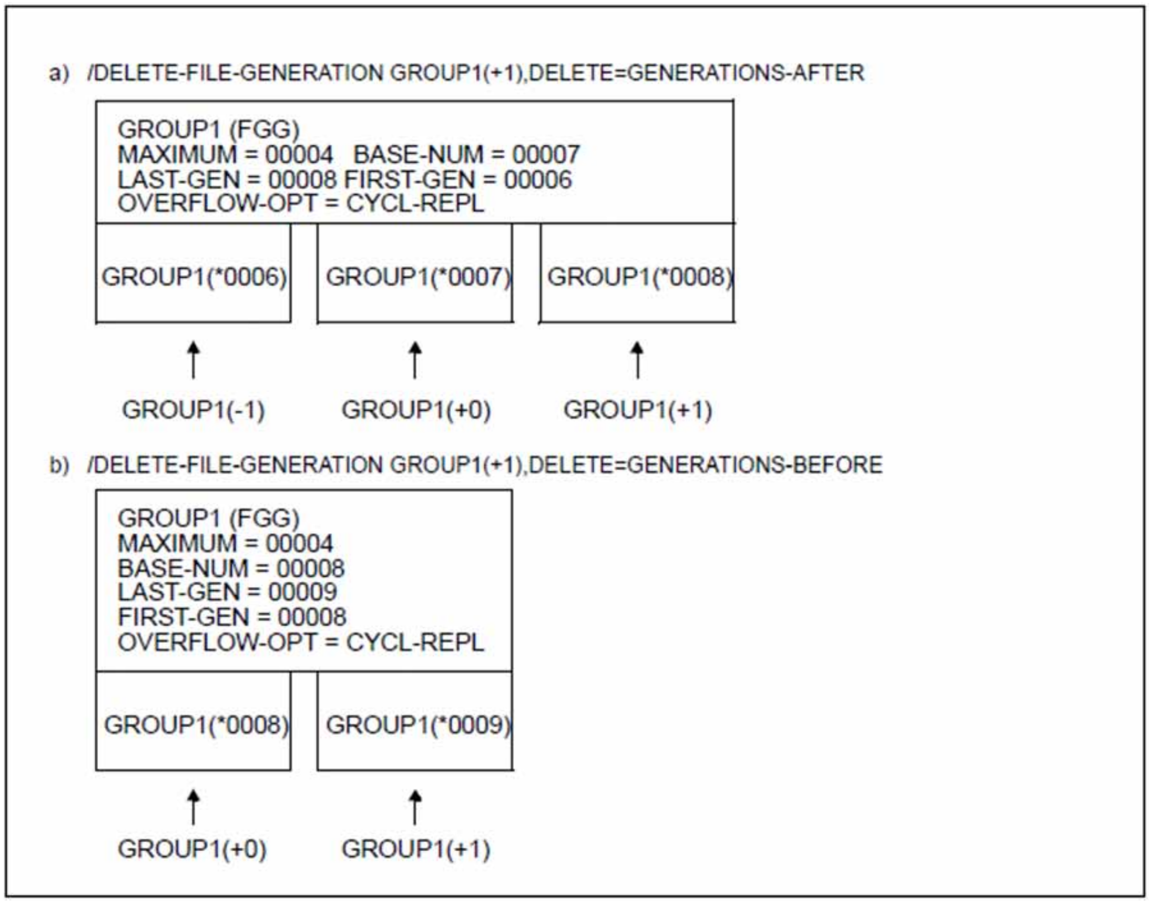


Figure 15: Deleting file generations (2)

10.4 Volumes for file generation groups

File generation groups can be stored on public or private volumes (disks or tapes). When selecting the type of volume, the user should consider the purpose for which the FGG is to be used: is it, for example, to be used for long-term storage of book-keeping data or for managing the backup copies of files created as part of regular processing (such as updating master data) or created by regular save operations in a computer center?

i If an FGG is created on private disks, all generations of this FGG must also be on private disks. In contrast to this, the mixed use of pubsets and tapes is permitted.

10.4.1 Pubsets

If an FGG is created on an SF pubset, the user – just as for other files – has no control over where, i.e. on which disks of the pubset, the generations are stored.

When creating an FGG on an SM pubset, the user has to define the FGG either as a group of permanent generations or as a work file generation group (WORK-FILE attribute). This is done either implicitly by assigning a default storage class or explicitly by specifying the. Once assigned, the attribute cannot be modified. If the associated generations are assigned a storage class (e.g. at initial allocation with the FILE macro) or the storage class is changed, the WORK-FILE attribute in the storage class must match the attribute of the group.

10.4.2 Private disks

The use of private disks for the generations of an FGG precludes the simultaneous use of other volumes for the generations of this FGG.

The group entry and the file generation entries are written into the F1 label of the private disk. By means of the CATAL macro (operand DISP=REUSE) or CREATE-FILE-GROUP and MODIFY-FILE-GROUP-ATTRIBUTES commands (operand OVERFLOW-OPTION= REUSE-VOLUME) the user can specify, in the catalog entry, that the volume containing the oldest generation (which is to be deleted) is to be used for storing a new generation. If the old generation being deleted extends over several disks, the new generation is cataloged only on the first disk of the series.

The user must remember the following with regard to the above-mentioned operands: the space occupied by the generation to be deleted is released only *after* the new generation has been created. If the private disk is full when the user attempts to create a new generation by means of the FILE macro or CREATE-FILE-GENERATION command, DMS cannot carry out the primary allocation for this new generation and the macro or command will be rejected due to the lack of storage space.

In contrast to FGGs on pubsets, FGGs or generations on private disks can be exported (by means of the ERASE macro, operands CATALOG/DELETE-OR-EXPORT/VOLUME or the EXPORT-FILE command) and imported into the system again at any time.

10.4.3 Magnetic tapes and magnetic tape cartridges

If file generations are stored on tapes, the user must remember that the group entry exists only in the file catalog of the pubset on which the FGG and its generations were cataloged. It is not transferred to tape. The file generations are identified as such in the HDR1 label of the tape by the fact that they have generation numbers.

Just like other tape files, file generations may also extend over several tapes, or there may be several generations on a single tape. Volumes which become free can, however, be reused (DISP=REUSE or OVERFLOW-OPTION=REUSE-VOLUME) only if the file generations are not stored as file sets. Instead, each must be stored as a single file on a tape or in a volume set.

Just like FGGs and generations on private disks, file generations on tape can be exported and imported again at any time.

10.5 Exporting file generation groups and generations

Only FGGs or file generations on private volumes can be exported. Exporting means that the related entry in the file catalog is deleted, but the storage space is not released, with the result that the data is retained.

Exporting is performed by means of the EXPORT-FILE command or the ERASE macro, using the operands CATALOG or DELETE-OR-EXPORT and VOLUME operands. The VOLUME operand may be used only for FGGs on private disks: the catalog entries of all generations stored on the specified volume are then deleted.

When exporting file generations, care must be taken – just as when deleting generations – that no gaps are left in the sequence of cataloged generations.

Example: exporting FGGs from private disks

A file generation group is stored as follows on two private disks:

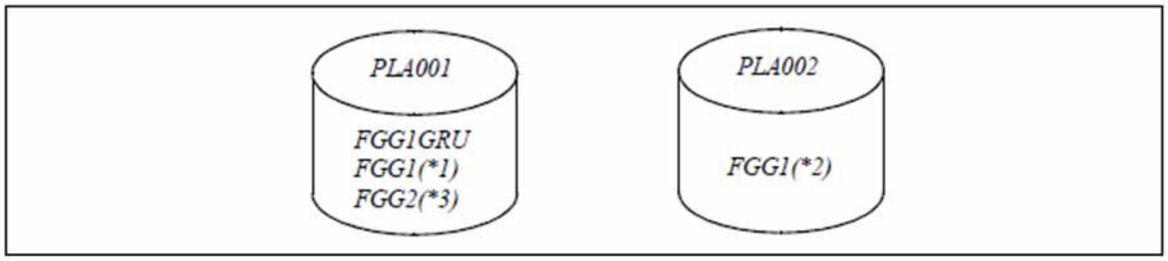


Figure 16: Exporting a file generation group

`GRU` = group entry

The command /EXPORT-FILE,VOLUME=PLA002 deletes the entry for generation FGG1(*2) from the file catalog, while generations *1 and *3 remain cataloged; the file generation group has a gap in its catalog entries, although the file generation group is still complete on the disks.

10.6 Importing file generation groups and generations

File generation groups or file generations on private disks, tapes or tape cartridges can be imported into a system at any time.

FGGs or generations on private disks can be imported either individually by means of the CATAL and FILE macros (with STATE=FOREIGN in each case) or in groups by means of the IMPORT macro or individually and in groups with the IMPORT-FILE command.

i The CATAL macro can only import FGGs, not generations; the FILE macro can only import generations, not FGGs.

The IMPORT macro and the IMPORT-FILE command generate the catalog entries from the entries in the F1 label of the private disk. File generations on tapes can be imported only by means of the FILE macro, not via the IMPORT macro. If the IMPORT-FILE command is used, file generations can be processed on tapes and private disks.

When importing an FGG or parts of an FGG, the user must remember that the group entry must be created first, before the first file generation can be cataloged. For the use of the IMPORT macro or IMPORT-FILE command for FGGs on private disks, this means that the private disk which contains the group entry must be requested first, or a group entry must be created by means of the CATAL macro or CREATE-FILE-GROUP command.

The following applies to importing by means of the CATAL and FILE macros: the group entry must be created first using CATAL and the operands STATE=FOREIGN, DEVICE and VOLUME. After this, the generations must be cataloged individually by means of FILE, specifying the operands STATE=FOREIGN, DEVICE and VOLUME.

The importation of FGGs and/or file generations is likewise subject to the rule that there must be no gaps in the sequence of generation numbers.

When importing from private disks which contain file generation groups, the user should remember that an IMPORT macro and IMPORT-FILE command catalog only those generations whose group entry is either on the specified disk or already in the system catalog. For a file generation group which extends over several disks and is not yet cataloged, this has the following consequences: if a disk which does not contain the group entry is imported first, followed by the disk with the group entry, the catalog entries for the file generations on the first disk will be missing. This can be remedied by issuing a new IMPORT macro or IMPORT-FILE command for the appropriate volumes or by issuing a FILE macro (operand STATE=FOREIGN) or IMPORT-FILE command for each of the generations which have not been cataloged.

Importing tape generations also causes the following attributes to be copied from the FGG index to the generations' catalog entry: AUDIT, BACKUP, DESTROY, LARGE, MIGRATE, NUM-OF-BACKUP-VERS and SHARE as well as the password encryption indicator.

Example: importing FGGs from private disks

Two file generation groups are divided as follows over three disks:

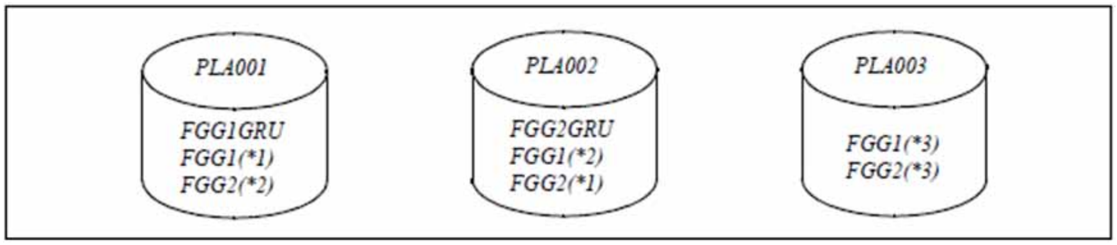


Figure 17: Importing file generation groups

`GRU` = group entry

File generations and group entries are not yet in the system catalog.

After execution of the commands

```
/IMPORT-FILE VOLUME=PLA001,DEVICE-TYPE=D3435
/IMPORT-FILE VOLUME=PLA002,DEVICE-TYPE=D3435
/IMPORT-FILE VOLUME=PLA003,DEVICE-TYPE=D3435
```

the following generations or group entries exist in the catalog:

```
FGG1 (FGG)  FGG2 (FGG)
FGG1 (*1)   FGG2(*1)
FGG1 (*2)   FGG2(*3)
FGG1 (*3)
```

The generation FGG2(*2) can be added later, e.g. by means of an IMPORT-FILE command.

Note that, in the above example, the commands could not be specified in an order that would enable all generations to be cataloged.

10.7 Reserving file generation groups

There is always a chance that various generations of a file generation group are processed, created or deleted simultaneously by parallel jobs. Such uncontrolled simultaneous access can very easily lead to inconsistencies within the sequence of generations or to data inconsistencies if the file generations result from a series of processing steps, each of which is based on the previous one.

One way of avoiding the problems inherent in concurrent access is to reserve the file generation group by means of the SECURE-RESOURCE-ALLOCATION command (operand FILE). This does not lock the volumes, but simply prevents access to the group entry and the file generations. This reservation option should be used whenever multiple concurrent access to an FGG cannot be ruled out and would result in inconsistencies. It is not necessary, for example, if only read access to the generations is permitted.

The following example illustrates potential problems in the case of simultaneous access from different jobs without reservation.

Example: parallel access to an FGG

Initial situation: the FGG GROUP1 consists of up to three generations, currently generations 4, 5 and 6; generation 5 is defined as the base for relative indexing (BASE-NUM=5). Jobs 0001 and 0002 are started at the same time and initiate the following activities:

Job 0001:

```
/CREATE-FILE-GEN GEN-NAME=GROUP1(*7)
/MODIFY-FILE-GROUP-ATTRIBUTES GROUP-NAME=GROUP1,
     GENERATION-PARAMETER=*GENERATION-PARAMETER(BASE-
     NUMBER=ABSOLUTE(NUMBER=0))
```

Job 0002:

```
/ADD-FILE-LINK LINK-NAME=EINGABE,FILE-NAME=GROUP1(-1)
```

If the ADD-FILE-LINK command of the second job is entered after execution of the CREATE-FILE-GENERATION command, the user will receive the error message “File not found”. Even if the absolute generation number (*4) had been entered in the ADD-FILE-LINK command, the same error message would have been issued.

If the MODIFY-FILE-GROUP-ATTRIBUTES command is also completed before the user issues the ADD-FILE-LINK command, DMS will find a file generation GROUP1(-1). However, this is not the generation job 0002 expects (namely *4), but generation (*6). The subsequent processing of the file will lead to incorrect results.

11 OPEN processing

Each program which wants to work with a file must initiate the following actions:

- If the file is not yet cataloged and has no storage space allocated: create the file (by means of a FILE macro or CREATE-FILE command).
- Open the file (by means of the OPEN macro): the system checks whether the calling job may access the file and, if so, prepares the actual access to the contents of the file.
- Access to the file contents (with the DMS access methods): when requested, the system transfers the data between the calling job and the file.
- Close the file (using the CLOSE macro): the system logically disconnects the calling job from the file; after this, any further requests for data transfer are rejected.

A job may open, access and close a file as many times as it wants. As opening and closing a file are time-consuming operations, a job should open files only when it can process them and close them only after all processing has been completed.

11.1 Information on OPEN processing

What information does DMS need when opening a file?

- Which file is to be opened?
- Does the calling job fulfill the conditions imposed by the protection attributes of the file (e.g. passwords)? Please refer to chapter “File and data protection”
- With which access method is the file to be processed? How is the file organized (block size, record length, etc.)?
- Is the file to be opened as an input file or an output file? If it is to be an output file, is it to be replaced or extended?
- In which part of the address space are the data buffers to be created: in user memory (class 6) or in system memory (class 5)?
- How are the records to be passed to the job and received from the job? Is the address at which the logical record begins in the buffer to be passed (locate mode) or is the record to be passed in a user area (move mode)?
- May other jobs access this file at the same time (shared update processing)?
- If the file exists or is to be created on private volumes, which private volumes are required?
- At which address is processing to be continued in the program in the case of an error or other event to which the program must react?

Where does DMS find the necessary information?

- In the **TFT entry** which contains the same file link name as the file control block of the program (FCB; the TFT entry is created by using the LINK-NAME operand in the ADD-FILE-LINK command or the LINK operand in the FILE macro). The TFT is the central administration table of DMS.
- In the **file control block** (FCB), i.e. the file declaration in the program (if no appropriate specification can be found in the TFT or if no TFT is available). If no TFT is present, the file name must be entered in the file control block (FILE operand of the FCB macro).
- In the **catalog entry** for the file.
- In the password table of the job. The password can be specified in the PASS operand of the FCB macro or in the ADD-PASSWORD command.
- The OPEN mode is specified in the OPEN macro (which has priority over the information in the TFT and the FCB).

The following applies to information which is available from more than one source:

- Specifications for the OPEN mode in the OPEN call have precedence over information in the TFT and FCB (there is no catalog entry for OPEN mode).
- Specifications in the TFT override the specifications in the file control block and the catalog entry.
- Specifications in the file control block (FCB) override those in the catalog entry.
- If the null operand was used in the FILE or FCB macro or the operand value *BY-CATALOG was defined for an operand in the ADD-FILE-LINK command, the corresponding values from the catalog entry are used.

  Fields which are transferred from the TFT entry to the FCB:

- BLIM
- BLKSIZE*)
- BLKCTRL*)
- BUFOFF*)
- CHAINIO
- CHKPT
- CLOSE
- CLOSMSG
- CODE
- DUPEKY
- FCBTYPE*)
- FILEFSEQ*)
- IOPERF*)
- IOUSAGE*)
- KEYLEN*)
- KEYPOS*)
- LABEL
- LARGE_FILE
- LINK
- LOCKENV
- LOGLEN*)
- POOLLNK
- OPEN
- OVERLAP
- PAD
- RECFORM*)
- RECSIZE*)
- RETPD
- SECLEV
- SHARUPD
- TAPEWR
- TPMARK
- TRANS
- VALLEN*)
- VALPROP*)
- WRCHK
- WROUT

The fields marked with an asterisk (*) can be specified as null operands or as BY-CATALOG.

- DMS always takes the information about the retention period for an existing file directly from the catalog entry or from the HDR1 label.
- If the specifications in the TFT (FILE macro, ADD-FILE-LINK command) and the FCB conflict with the contents of the catalog entry, DMS aborts the open operation with an error code.

FCB (file control block)

The file control block describes the file and/or the file attributes in a user program. An FCB can be created or modified at the following stages:

- Before opening the file: by means of the FCB macro when the program is written: this macro reserves storage space for the file control block and optionally supplies it with information.
- During the program run and prior to opening the file: by directly supplying the fields of the FCB with current values. Support for this is provided by the IDFCB and IDFCBE macros, which generate a DSECT for the FCB.
- When the file is being opened, the user has the option of checking, and possibly modifying the file control block after certain processing steps performed by the OPEN routine (EXLST macro, operands OPENX and OPENZ).

As long as a file remains open, the file control block contains all the information describing the current status of the file. This information is of significance to the user if an error occurs and has to be analyzed in the program (see the EXIT operand of the FCB macro and the EXLST macro). The file control block then contains a DMS error code and an identifier for the error exit.

Information sources for DMS

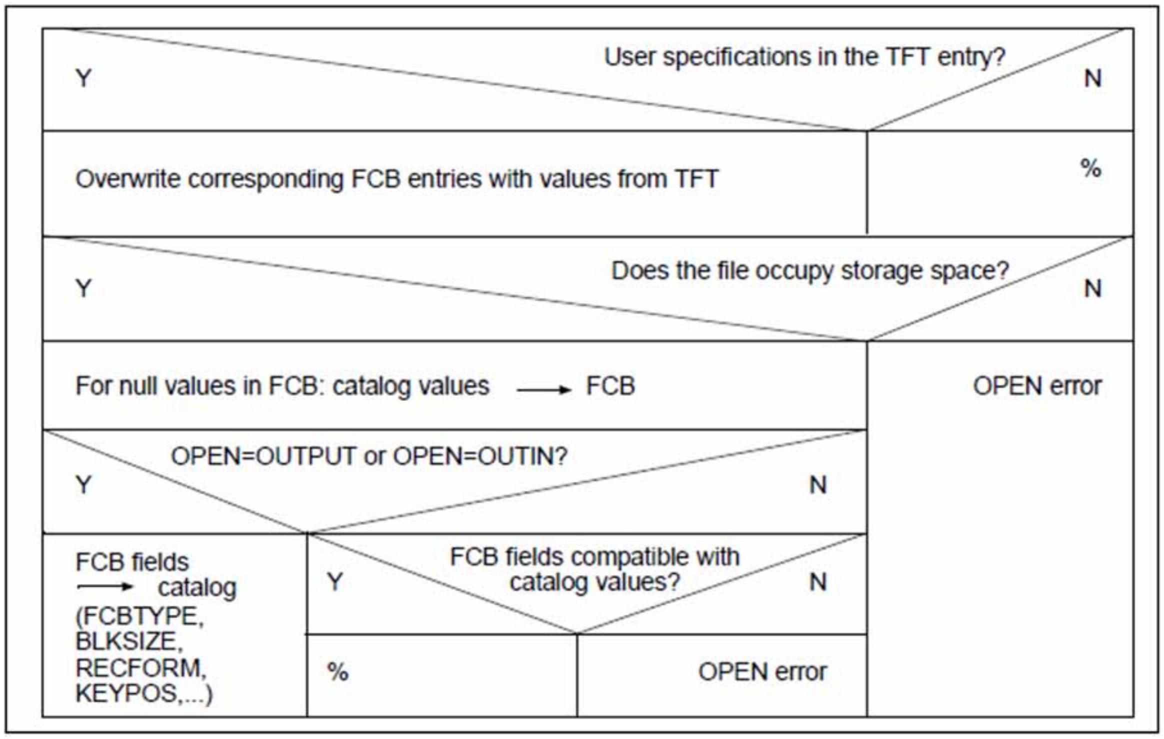

User specifications in the TFT entry?
Y
N
Overwrite corresponding FCB entries with values from TFT
%
Does the file occupy storage space?
Y
N
For null values in FCB: catalog values → FCB
OPEN error
OPEN=OUTPUT or OPEN=OUTIN?
Y
N
FCB fields → catalog (FCBTYPE, BLKSIZE, RECFORM, KEYPOS,...)
FCB fields compatible with catalog values?
Y
N
%
OPEN error

Figure 18: Information sources for DMS

11.2 Sequence of OPEN processing

OPEN processing varies slightly, depending on whether or not the user the is working with the XS interface (31-bit addressing). The following section describes the sequence for XS OPEN processing.

The special features of the 24-bit interface are described in section “XS and non-XSinterfaces”.

DMS executes the following steps:

- Add missing information to the FCB from the TFT and the catalog entry.
- Request private volumes (if needed).
- Link the FCB to the routines of the access methods. Store in the FCB the addresses of the logical routines (TU logicals) which handle the blocking and unblocking of records.
- Construct the privileged internal file control block (TPR FCB): the user cannot access this control block; it is used internally by DMS.
- Allocate buffer areas.
- Update the catalog entry.
  In particular with node files the file attributes CHANGE-DATE, FILESIZE, Highest-Used-Page (LPP) and LBP are obsolete when a foreign system has modified the files. These values are updated automatically during OPEN processing. Please note that in the case of successive OPEN calls (without any CLOSE), the CE will only be updated during the first OPEN. The following OPEN receive the current values of the LPP and LBP in the FCB. The catalog entry can, however, also be updated manually with IMPORT-NODE-FILE REPLACE = *NODE-FILE-UPDATE. In this case, the updated values for FILESIZE and LPP are only estimated for SAM node files.

Note

The catalog entry for a file is not complete until the file has been opened as an output file and closed again at least once. A distinction is therefore made between cataloged files and existing files.
If DMS macros refer to “cataloged files”, for instance, then catalog entries must exist for these files.
If the macros refer to existing files, then these files must have been opened at least once as output files. Such files may still be empty, however, i.e. may contain no records accessible to the user.
The latest page pointer (last-page pointer) and the “last-byte pointer” are not transferred to the catalog entry until the file is closed.
The last-page pointer is only relevant for PAM files and SAM node files opened in UPAM-RAW mode.

The main actions performed in the file opening sequence for disk and tape files are as follows:

- Search for a TFT entry with the file link name contained in the FCB. The file link name is either taken from an FCB macro or written directly into the control block. Every FILE/ADD-FILE-LINK or CHNGE/CHANGE-FILE-LINK call creates or updates an entry in the TFT. Each such entry includes all the parameters required to update the file control block and establishes a link between the application program and the file via a link name (LINK operand). There is no search for a TFT entry if the LINK operand of the FCB macro contains a file link name consisting of 8 blanks (8X'40'; 8C'_').

  The appropriate TFT entry is used to update corresponding fields in the file control block. If no entry with the specified file link name exists or if the link name in the FCB consists of 8 blanks (8X'40'; 8C'_'), a TFT entry is created during OPEN processing. This TFT entry is automatically released by the system (implicit RELEASE) at the CLOSE. Only the TFT entries generated with FILE (LINK operand) are retained until explicitly released (REMOVE-FILE-LINK command or RELTFT macro).

  Fields specified as null operands in the FILE macro are assigned the entry “BY-CATALOG” in the TFT. The corresponding fields in the FCB are completed later with information from the catalog entry.
- Complete the file name. This means that if the file name was not specified in its complete form by the user, the catalog ID and the user ID are added to it, and the completed file name is stored in the form catid:$userid. filename in the associated TFT entry and the FCB. If no user ID is specified, the user ID of the SET-LOGON-PARAMETERS command is used. If no catalog ID is specified, the default catalog ID that was defined for the user ID is used.
- Read the catalog entry for the requested file. If the file cannot be found under the explicitly specified catalog ID and user ID, and is not located under the default ID of a default catalog, OPEN processing is aborted with an error message. If the OPTION=GLODEF operand was specified in the FCB call. no error occurs; a secondary read will be performed instead.

  An example of “secondary read”, which is enabled by means of the FCB operand OPTION=GLODEF, can be found on "Access via the system default user ID".

  As far as possible, undefined fields of the FCB are filled with information from the associated catalog entry. In the case of files with the status FOREIGN (tape files being exchanged), the parameters are taken from the tape labels.
- Compare the volume list of the catalog entry with the volume list recorded in the TFT. If no such entries are recorded in the TFT, they are created from the specifications in the catalog, and the required volumes are then requested via Device Management.
- Assign tape devices.

  If a TSET is defined for the file, DMS compares the volume lists of the TFT entry and the related TST (tape set table) entry. It thereby is assuming that the file begins on the tape specified as the current tape in the TST. If the TFT volume list contains the current VSN, any VSNs which precede this in the TFT volume list are not transferred to the catalog entry. If the TFT volume list does not contain the current VSN, the current VSN and all VSNs which follow this in the TST list replace the TFT volume list and are transferred to the catalog entry.

  If no TSET is defined for the file, but the TFT entry contains a temporary volume list (TVSN), DMS assigns the required tape on the basis of this TVSN. The volume list in the catalog entry is ignored. The OPEN is rejected if the file is an output file. If no TSET is defined, but VSEQ is specified in the FILE macro, the file must not be opened with OPEN INPUT/EXTEND. The first or last tape in the TFT volume list is assigned.

  If neither TSET nor TVSN or VSEQ is defined, the volume list of the catalog entry is updated to match that of the TFT entry and the first or last tape in the list is assigned.
- Create the internal privileged file control block (TPR FCB) and initialize it with values from fields of the FCB and the catalog entry.

  The original FCB with its field values is stored in a save area at the end of the TPR FCB before the OPEN sequence.

- Branch to the OPENX exit of the EXLST macro (if provided in the program). At this point the program can check whether the validity of the values set thus far. If the program changes the fields of the file control block, processing continues with these values. If a new file is being created, the values are also transferred to the labels and the catalog entry. The file name, link name and shared-update mode can no longer be changed. Processing is resumed by means of the EXRTN macro.
- Branch to the OPENZ exit of the EXLST macro (if provided in the program). At this point the program has a further opportunity to update the file control block, but this time the changes are not transferred to the catalog entry. It is thus possible, for example, to process a file using a different access method from the one entered in the catalog. The file name, link name, OPEN mode, and shared-update mode can no longer be changed. Processing is resumed by means of the EXRTN macro.
- Check I/O performance data and existing cache configuration to determine whether a cache is to be used when processing the file or not. This is subject to the following considerations:

  If the file has already been opened by another job, the attributes currently set also apply to this OPEN.

  If this is the first time the file has been opened, the static performance attributes (PERFORMANCE, USAGE and DISK-WRITE) are checked against the cache attributes, and the values set accordingly are stored in the catalog entry. (Example: if DISK-WRITE=*IMMEDIATE and USAGE=*WRITE/*READ-WRITE, write caching is not permitted unless a secure (nonvolatile) cache is used).
  Dynamic replacement of the static PERFORMANCE and USAGE values defined in the catalog entry by lower values is possible using TFT or FCB entries, permitting caching to be deactivated for the current file processing; the attributes stored in the catalog entry are not modified by this operation.
- Check user specifications against the catalog entry of the file. (The file must be an input file, i.e. the OPEN mode must not be OUTPUT or OUTIN.)
- Transfer FCB values to the catalog entry. If the file to which the FCB refers is an output file (OPEN OUTPUT /OUTIN), the FCB values are transferred to the catalog entry.
- Check access specifications concerning expiration dates, access rights, and passwords for files. This check is performed for previously created files.
- When creating a new file (OPEN mode OUTPUT or OUTIN) on an SM pubset, check whether the attributes of the current location are compatible with the requested file attributes (e.g. file format BLKCTRL= PAMKEY). If they are incompatible, the file must be reallocated with a compatible location (without data transfer).
- Create check labels for tape processing (see "Label processing when opening tape files"). When a tape file is newly created, the file header labels HDR1, HDR2, and HDR3 are initialized with values and written before the first block of the file. When the file is accessed for reading, the contents of these labels are used to identify the file and to check the access authorization. Label information must be structured in compliance with the DIN 66029 standard to ensure normal processing. If the tape file does not contain standard labels in accordance with the above DIN standard, the checking of the tape file will need to be handled by user routines.
- Assign I/O areas or validate buffer addresses if they have been specified by the user.
- Load the connection routines (TU logicals) if the access method SAM or ISAM is being used.
- Set up a connection between file control block and access methods: the addresses of the logical routines are placed in the FCB.
- Update and rewrite the catalog entry if required.

Notes

The file will be destroyed if an error occurs during OPEN processing after reallocation of the file.

Furthermore, reallocation may result in modifications of the extent list during OPEN processing; this means that the information supplied by the FSTAT macro or the SHOW-FILE-ATTRIBUTES command will be different after the OPEN.

The OPEN will be rejected with return code DMS0D80 if no suitable location can be found within the SM pubset.

11.3 OPEN in configurations with files > 32 GB

The OPEN interface checks whether files are permitted to extend beyond 32 GB and whether the creation of files >= 32 GB and access to such files are permitted.

There are two aspects associated with this:

1. Rejection of access to or the creation of large files for access methods that do not allow processing of large files.
2. Indication that a program can create or open files >= 32 GB.

Incompatible interface variants

Interfaces where 3-byte block numbers are used are never able to work with files >= 32 GB. This affects the following cases:

- All files in key format (BLKCTRL=PAMKEY): The logical block numbers in the PAMKEY are only 3 bytes wide.
- 24-bit interface of UPAM:
  The field for logical block numbers in the UPAM parameter lists and in the TU FCB is only 3 bytes wide.
- 24-bit interface of SAM:
  The logical block numbers are affected as part of the retrieval address.
- 24-bit interface of ISAM

In all the cases described above, the following applies:

- Access to files >= 32 GB is rejected with the return code `X'0000D9D'` or `X'00000D00'`, depending on the size of the storage space allocated to the file.
- Exceeding a file size of 32 GB as a result of secondary allocation is prohibited.

Semantic incompatibilities

It is possible that applications use interfaces that employ 4-byte fields for the data fields described above, but that these 4-byte fields are implicitly or explicitly subject to the former limitation to values less than `X'00FFFFFF'`.

The following lists a number of examples of potential problems:

- The highest 3-byte block number `X'FFFFFF'` has a special meaning.
- “Block numbers” greater than `X'00FFFFFF'` represent objects other than blocks.
- Overflow can occur in the case of calculations with block numbers or counters greater than `X'00FFFFFF'`.
- The number of digits in input or output fields is not sufficient for displaying such large block numbers or counters.
- When converting hexadecimal numbers to decimal numbers, the field length is too small for the decimal number.
- It is assumed that data structures whose size depends on a file size always find space in virtual memory. This assumption can apply to files less than 32 GB, but not if this size is exceeded.

For detailed, related information on the subject of “large files” refer to the “Files and Volumes larger than 32 GB” manual [18 (Related publications)].

11.4 XS and non-XS interfaces

In addition to the XS interface, which permits 31-bit addressing, the “old” non-XS interface, with 24-bit addressing, is supported for reasons of compatibility. This means that programs with 24-bit addressing can be executed without conversion, depending on the Assembler presettings.
The user can then profit from the larger user address space of 16 Mbytes. Conversely, programs which use 31-bit addresses can still be executed in the lower address space. Programs with 24-bit addressing can also be linked with programs with XS capability.

11.4.1 FCB handling during OPEN processing

If a program uses the 24-bit interface, file processing is executed with a 24-bit TU FCB. In contrast to the 31-bit TU FCB (XS), space is reserved in the 24-bit TU FCB for the logical routines, which are actually copied into this space. Loading of the logical routines can be suppressed by means of the FORM=SHORT operand in the FCB macro. In this case, however, the user must handle all blocking and unblocking of records himself/herself. For normal processing of SAM and ISAM files, the file cannot be opened if FORM=SHORT is specified.

11.4.2 Generation mode

The choice of DMS interface is controlled by the generation mode. It is preset by the assembler, but can also be selected globally by means of the GPARMOD macro or separately for each macro by means of the PARMOD operand.

Although it is possible to select different interfaces for different files within one program, all the macros for one file /FCB (OPEN, CLOSE, EXLST, FCB, action macros) must be assembled using the same generation mode.

Example

For SHARUPD processing, different programs can process the same file in a different generation mode. Where PAM macros are chained, various generation modes can be selected within the chain.

```
START
             USING *,3
             LDBASE 3,ORG=YES
             :
             OPEN  FCB1,PARMOD=24
             OPEN  FCB2,PARMOD=31
             :
             GET   FCB1,PARMOD=24
             PUT   FCB2,PARMOD=31
             :
             CLOSE ALL,PARMOD=31
             TERM
             :
   FCB1      FCB   PARMOD=24,EXIT=EXLST1
   EXLST1    EXLST PARMOD=24
             :
   FCB2      FCB   PARMOD=31,EXIT=EXLST2
   EXLST2    EXLST PARMOD=31
             :
             END
```

Example

Chaining PAM macros with different generation modes

```
ELEM1    PAM   FCB2,CHAIN=ELEM2,PARMOD=31
ELEM2    PAM   FCB1,CHAIN=ELEM3,PARMOD=24
ELEM3    PAM   FCB2,PARMOD=31
```

11.4.3 Addressing mode

Existing object modules and programs that were assembled using the old generation mode (corresponds to PARMOD=24) are still executable, but only in the 24-bit address space, i.e. addressing is 24-bit-dependent.

Programs assembled using the new generation mode for 31-bit addressing are executable in both 24-bit and 31-bit addressing mode, i.e. they are “addressing-mode-independent”.

Here too, it is possible for one file to be processed in 24-bit addressing mode and another in 31-bit addressing mode in the same program, or even for a mix of both modes to be used. Note, however, that the addressing mode applicable to the FCB must be set before issuing a DMS macro.

11.4.4 Conversion notes 24/31-bit

PARMOD=31 expansions of the DMS macros usually differ from the PARMOD=24 expansions, which may occasionally result in base register overflows, etc. All PARMOD=31 expansions begin with an 8-byte standard header. This standard header is described in the appendix of the "DMS Macros" manual [1 (Related publications)].

Access methods SAM and ISAM

All action macros generated with the 31-bit interface destroy the contents of register 15 in all cases. The contents of registers 0, 1 and 14 may also be destroyed, e.g. in error situations.

The retrieval address supported by SAM can be found, for the 31-bit interface, in the FCB fields ID1BLK# (block counter) and ID1REC# (record counter). In the 24-bit TU FCB, this address is kept in field ID1RPTR (see section “FCB retrieval address”).

12 Label processing when opening tape files

During OPEN processing, and depending on the label structure of the file, the labels are checked in the sequence:

1. Volume header labels
2. File header labels

If the result of a check is negative, DMS issues a message which informs the operator or user about what is wrong and gives him/her a chance to react to the error. Possible reactions:

- repeat the label check
- branch to an exit routine in the program, or
- abort the program.

Whether or not error messages are issued depends on the TAPE-ACCESS flag in the user catalog and on the SECLEV operand in the FILE/FCB macro or the PROTECTION-LEVEL operand in the ADD-FILE-LINK command.

For both input and output files, the sequence of label checking varies slightly for the various values in the LABEL and LABEL-TYPE operands of the macro and command, respectively. With LABEL=STD or LABEL=(STD,n) in the macro, or LABEL-TYPE=*STD or LABEL-TYPE=*STD(DIN-REVISION-NUMBER=n) in the command, the user specifies that a file with standard labels is to be processed. The specifications LABEL=NSTD and LABEL-TYPE=*NON-STD indicate that the file to be processed will have nonstandard labels, and label checking by DMS is executed accordingly. The same applies to the specifications LABEL=NO and LABEL-TYPE=*NO: DMS then expects a file with no labels.

In the following sections, label handling during OPEN processing is described in the following order: label handling for input files is described first, followed by label handling for output files, in each case in the sequence: files with standard labels, files with nonstandard labels, and files without labels.

The structure of the routines in which user labels or nonstandard labels are processed is described in section "EXLST exits for processing labels for tape files".

12.1 Prerequisites for label processing

- TAPE-ACCESS authorization (TPIGNORE)
- BYPASS handling
- Owner identifier
- DMS messages during label processing
- Repeat label checking (RETRY)

12.1.1 TAPE-ACCESS authorization (TPIGNORE)

“TPIGNORE” stands for “ignore error on tape” and refers to the right of the user (or of the user ID) to ignore error messages issued by the label processing routines.

If (s)he is recorded in the HDR3 label as the owner, the file owner can ignore all such error messages or even bypass label checking (SECLEV operand in the FILE/FCB macro; PROTECTION-LEVEL operand in the ADD-FILE-LINK command). Users wishing to access “foreign” tape files can ignore error messages or bypass label checking only if systems support has granted them the appropriate TAPE-ACCESS authorization. This authorization is registered in the field TAPE-ACCESS in the user catalog. The user can obtain information on the entries in the user catalog for his/her user ID with the SHOW-USER-ATTRIBUTES command.

The following table shows the various levels of the TAPE-ACCESS authorization.

TAPE-ACCESS value	**Meaning**
STD	The user is not authorized to ignore error messages.
PRIVILEGED	The user may ignore error messages; in batch mode, DMS suppresses certain messages if PROTECTION-LEVEL=LOW applies; if PROTECTION-LEVEL=HIGH is set, the operator must decide whether the error may be ignored(ADD-FILE-LINK command, PROTECTION-LEVEL operand).
READ	The user may ignore error messages which refer to input files; label checking is not deactivated.
BYPASS-LABEL	The user may bypass label checking and/or ignore error messages for input files.
ALL	The user may bypass label checking and/or ignore all error messages.

Table 30: TAPE-ACCESS authorization

Messages concerning the mounting of tapes or checking of volume labels are sent to the operator. The operator then decides how tape processing is to proceed. Messages concerning the checking of file labels are sent to the user or to the calling job. The user can then activate suitable error routines in his program.

The TAPE-ACCESS authorization also affects BYPASS handling.

12.1.2 BYPASS handling

If, when a tape input file is opened (OPEN INPUT or REVERSE), BYPASS=LP is specified in the FILE macro or BYPASS-LABEL-CHECK is specified in the ADD-FILE-LINK command, and if the user has the appropriate authorization, all label checks are bypassed. The tape is positioned as specified in the second entry in the BYPASS operand or according to the value specified for the BYPASS-LABEL-CHECK operand, as appropriate.

If the authorization is missing, OPEN processing is aborted with an error message.

Mounting of the tape is monitored by device management in accordance with the user's specifications.

As all label checks are bypassed, the system cannot check which code is used in the file. If the code is neither EBCDIC nor ISO 7-bit code, the user must provide his/her own code table.

If the BYPASS function is specified together with LABEL=NSTD in the macro or with LABEL-TYPE=*NON-STD in the command, only the system checks of the volume header labels are bypassed. The user routines for label handling are activated in the normal manner. The positioning information in the BYPASS operand is evaluated for LABEL=NSTD or LABEL-TYPE=*NON-STD only if there are no UVL labels to check.

12.1.3 Owner identifier

The volume header label VOL1 contains an “owner identifier” which is used for checking the access authorization.

The owner identifier is normally empty or simply specifies the computer center as the owner of the tape. In this case, any user may access the tape.

If the owner identifier shows that the tape belongs to a specific user, the “access indicator” is taken from the HDR1 label when the first file on the tape is created if this file is created by the owner of the tape. If a restriction is noted in the access indicator, only the tape owner may access the tape. If the tape owner and the owner identifier are the same, or if the owner identifier is empty, the access indicator is assumed to denote unrestricted access.

12.1.4 DMS messages during label processing

During label processing, DMS may react with the following messages to conflicts detected during label checking:

```
DMS0DA1  WRONG VSN ON DEVICE (&00). MOUNT VSN (&01) FOR FILE (&02) VSEQ '(&03)'. REPLY
           (0=EXIT; 1=RETRY; 2=DISPLAY LABEL; I=IGNORE)
DMS0DA2  NO VOL1/HDR ON DEVICE (&00). MOUNT VSN (&01) FOR FILE (&02) VSEQ '(&03)'. REPLY
           (0=EXIT; 1=RETRY; 2=DISPLAY LABEL)
DMS0DA3  WRONG OVERWRITE ACCESS: VSN '(&00)' FOR FILE '(&01)' VSEQ '(&02)'. REPLY (0=EXIT;
           1=RETRY; 2=DISPLAY LABEL; I=IGNORE)
DMS0DA5 WRONG FILE IDENTIFICATION: VSN '(&00)' FOR FILE '(&01)' VSEQ '(&02)'. REPLY (0=EXIT;
           1=RETRY; 2=DISPLAY LABEL; I=IGNORE)
DMS0DD6  WRONG FILE SET IDENTIFIER ON VSN '(&00)' FOR FILE '(&01)' VSEQ '(&02)'. REPLY
           (0=EXIT; 1=RETRY; 2=DISPLAY LABEL; I=IGNORE)
DMS0DD7  ILLEGAL ACCESS BY FOREIGN USER: VSN '(&00)' FOR FILE '(&01)' VSEQ '(&02)'. REPLY
           (0=EXIT; 1=RETRY)
DMS0DD8  STD FILE LABELS NOT FOUND ON VSN '(&00)' FOR FILE '(&01)' VSEQ '(&02)'. REPLY
           (0=EXIT; 1=RETRY; 2=DISPLAY LABEL)
DMS0DF6  INPUT TAPE WITH VSN (&00) HAS STD LABELS; BUT SHOULD BE TREATED AS NSTD OR NO.
           REPLY (0=EXIT; C=CONTINUE)
DMS0DFB  VSN '(&03)' ON '(&00)'FOR FILE '(&01)' VSEQ '(&02)' OK? REPLY (TSN.0=ERROR EXIT;
           TSN.1=RETRY; TSN.2'XXXXXX'=NEW VSN; TSN.=ACCEPT)
```

The texts in the messages have the following meanings:

`EXIT`	if available, the exit routine in the program is activated and can intercept the error and terminate the program normally.
`RETRY`	The label check is repeated (see section “Repeat label checking (RETRY)”).
`DISPLAY LABEL`	The contents of the HDR label being checked are sent via system file SYSOUT to the screen or, for batch processing, written into the SYSOUT file.
`NEW VSN`	After a change of tape, label checking is to be repeated for the new tape (see section “Repeat label checking (RETRY)”).
`IGNORE`	Depending on the TAPE-ACCESS authorization and the value of the SECLEV operand in the FILE/FCB macro or the PROTECTION-LEVEL operand in the ADD-FILE-LINK command, the error can be ignored and program execution can be continued.
`TSN`	The task sequence number of the job for which label checking was executed.
`VSN`	The volume serial number of the tape on which label checking was executed or the VSN of the next tape to be mounted.

12.1.5 Repeat label checking (RETRY)

Normally, the correct volume is already mounted on the tape device if the VOLUME operand of the FILE macro or the VOLUME operand of the CREATE-FILE, IMPORT-FILE or CREATE-TAPE-SET command was specified correctly.

However, in exceptional cases, the operator may have mounted the “wrong” tape. The message “RETRY WITH ANOTHER VOLUME” is then issued during label checking.

The user can now retry label processing in interactive mode. If the error recurs, the user is requested to enter a new VSN so that a different tape is mounted. In the case of output and foreign files, the catalog entry is updated with this new VSN as soon as the mount operation has been completed.

Before the operator answers the error message on the console for continued tape processing, he/she can change the tape in the event of an error. The VSN of the new tape is normally identical with the VSN in the catalog entry.

If the VSN of the new tape differs from that in the catalog entry, the catalog entry is updated. This happens in the following cases:

- the file was created (OPEN OUTPUT/OUTIN) with or without specification of a TSET, and is being used for the first time or after tapes have been changed
- the file is being imported (STATE=FOREIGN or IMPORT-FILE command)
- the file is being extended onto a new tape since the existing tapes are full.

In these cases, an “acknowledge VSN” is issued at the console as soon as the operator has answered the “RETRY” message.

12.2 Label processing for input files

The first step in the processing of labels is usually to check the volume header labels. This step is omitted if the VOL1 check has already been executed successfully for the tape. In this case, the tape is immediately positioned to the start of the file and label checking is continued with the HDR labels.

12.2.1 Files with standard labels

The processing of input files with standard labels is requested by means of the operand LABEL=(STD,n) of the FILE macro or the operand LABEL-TYPE=*STD(DIN-REVISION-NUMBER=n) of the ADD-FILE-LINK command.

Checking volume header labels

1. Is the first record on the tape the volume header label VOL1?

The volume header label is identified by the character string VOL1 in the fields label name and number (the first four characters on a tape with standard labels).

If the tape does not contain this string, i.e. if the first record is not a VOL1 label, DMS issues message `DMS0DA2` and the operator must react to this message.

2. Is the file already cataloged or is it a foreign or imported file? Has the user already requested a tape for the foreign or imported file?

If the file is already cataloged, its catalog entry contains the list of tapes it occupies. For foreign files (i.e. files to be imported which have not yet been cataloged), you must specify the VSNs of the required tapes in the VOLUME operand of the FILE macro or in the VOLUME operand of the IMPORT-FILE or CREATE-FILE command. If you have not done this, DMS issues message `DMS0DFB` on the console at teh time OPEN processing takes place so that the operator can check the mounted tape.

3. Does the VSN in the VOL1 label match the VSN specified in the FILE macro or in the IMPORT-FILE/CREATE-FILE command?

DMS compares the contents of the third field in the VOL1 label (the VSN of the tape) with the VSN specified by the user. If they do not match, DMS issues message `DMS0DA1` on the console. Now the operator can check the label, mount a new tape, abort the program (OPENER routine), or ignore the message (if the job has the appropriate TPIGNORE authorization).

4. Is access permitted only for the tape owner? If so: is the user ID the same as the tape owner identifier?

DMS evaluates the contents of the VOL1 fields “access indicator” and “owner identifier”. If the user is not authorized to access the tape, DMS sends him/her message `DMS0DD7`. The user can now terminate his/her program or have label checking retried (with a different tape in the case of foreign files).

5. Does a non-BS20000 tape contain further VOL labels?

DMS ignores the volume header labels VOL2 to VOL9, i.e. it does not evaluate them and skips to the next step of OPEN processing.

6. Are the standard volume labels followed by user volume labels (UVLs)?

UVL labels must be processed by special routines in the user program, to which the program branches via the OPENV exit of the EXLST macro. If the tape contains UVL labels (UVL1 through UVL9), DMS activates the OPENV routine for each such label; the address of the label is passed in register 0. In each case, label checking is terminated by means of the LBRET macro. If the program does not have an OPENV routine for processing UVLs, DMS ignores these labels.

Positioning

Next, DMS positions the tape to the start of the file to be read. It is determining the position of the file on the tape from the FILE/FCB macro (operand FSEQ), from the ADD-FILE-LINK command (operand FILE-SEQUENCE), or from the catalog entry (field FSEQ). If the file is a foreign file whose tape position is unknown, the user can specify FSEQ=UNK (= unknown) in the FILE/FCB macro or FILE-SEQUENCE=*UNKNOWN in the ADD-FILE-LINK command. In this case, DMS positions the tape to behind the volume header labels and then searches the whole tape. Once the tape has been positioned correctly, DMS checks the file header labels HDR1 through HDR3 and UHL1 through UHL255.

Checking file header labels

1. Is the first record of the file, or the first record after the volume header labels, the HDR1 label?

The HDR1 label has the character string HDR1 in the first four bytes. If DMS does not find this string, it sends message `DMS0DD8` to the user. After correction, the user can have the label check retried. If he wanted to read a foreign file, he has to search for it on another tape.

If DMS finds a HDR1 label, it then reads the HDR2 label. It does not read the HDR3 label unless it can see from the system code of the HDR1 label that the tape file was created in BS2000. Only in this case are the steps described under 2 and 3 carried out.

2. Do the file names in the HDR1 and HDR3 labels match the file name entered by the user?

DMS compares the file names in the HDR1 and HDR3 labels with the file name specified in the FILE macro or in the IMPORT-FILE/CREATE-FILE command. If they do not match, DMS sends message `DMS0DA5` to the user, who can react with “Terminate program”, “Retry label checking”, “Print label”, or “Ignore” (depending on his/her TPIGNORE authorization).

3. Is file access permitted only for the file owner? If so: does the owner identifier in the HDR3 label match the user ID of the job?

The HDR1 field “access indicator” shows whether a file is shareable. If the file is not shareable, DMS compares the owner identifier in the HDR3 label with the user ID of the job. If these do not match, DMS sends message `DMS0DD7` to the user, who can react as described for VOL1 label checking, namely by terminating the program or repeating the label check (with another tape in the case of foreign files).

4. Does the file section number in the HDR1 label match the number specified in the FILE macro or ADD-FILE-LINK command?

The file section number, which is kept in the HDR1 label, shows the position of a file section within a volume set. With the VSEQ operand of the FILE macro or the VOL-SEQUENCE-NUMBER of the ADD-FILE-LINK command, the user has specified which section of a multivolume file he/she wants to process. If these two entries do not match, DMS sends message `DMS0DD6`. The user can – if authorized – ignore it or react to it as he/she sees fit.

5. Are there passwords to be transferred to the catalog entry?

For foreign files cataloged by means of a FILE macro or IMPORT-FILE command, any passwords in the HDR3 label are transferred to the new catalog entry.

6. Does the file have a retention period or is it write-protected?

If the file is to be opened with OPEN INOUT or OPEN EXTEND (see OPEN modes in the “DMS Macros” manual [1 (Related publications)]), DMS checks the HDR3 field “access type”, which corresponds to the ACCESS specification in the catalog entry. It also checks the field “expiration date”

in the HDR1 label, which indicates whether a retention period still exists for the file or whether it can be overwritten. A retention period may have been defined via the operand RETPD of the FILE/FCB macro or the RETENTION-PERIOD operand of the ADD-FILE-LINK command when the file was created.

If either the “access type” or the “retention period” indicates that the file is write-protected, DMS sends message `DMS0DA3` to the user. The User can – if she/he is authorized – ignore this message and continue processing, or have label checking retried (with a different tape in the case of foreign files), or have his program terminated correctly via an exit routine (OPENER).

7. Are the HDR labels followed by user header labels (UHLs)?

User header labels (UHL0 through UHL255) have to be evaluated by an exit routine in the user's program, to which the program branches via the LABGN exit of the EXLST macro. DMS activates this routine for each such label; the label address is passed in register 0. In each case, label checking is terminated by means of the LBRET macro. If the program has no facilities for user header label processing, DMS ignores these labels.

12.2.2 Files with nonstandard labels

The processing of input files with nonstandard labels is requested by means of the operand LABEL=NSTD of the FILE macro or the operand LABEL-TYPE=*NON-STD of the ADD-FILE-LINK command.

Checking labels

DMS first checks whether the tape starts with a VOL1 label. If not, file access is permitted without further checking (processing continues with step 2, "Files with standard labels").

If the tape has a VOL1 label, DMS checks, just as for files with standard labels, whether the VSNs in the label and in the FILE macro or ADD-FILE-LINK command agree. If not, DMS sends message `DMS0DA1` to the operator. For foreign files for which no VSN was specified in the FILE macro or ADD-FILE-LINK command, DMS issues message `DMS0DFB`.

If the VSNs match, the contents of the HDR labels determine whether or not file access is permitted. Just as for files for which LABEL=(STD,n) or LABEL-TYPE=*STD(DIN-REVISION-NUMBER=n) was specified, DMS first reads the HDR1 label and then the HDR2 and HDR3 labels.

The “access indicator” in the HDR1 label is checked, and if the file is not shareable the owner identifier is compared with the user ID of the job (error message: `DMS0DD7`). If the file is to be opened with OPEN INOUT, the HDR3 field “Access type” (ACCESS=READ) and the HDR1 field “Expiration date” are checked (error message: `DMS0DA3`).

Evaluating labels

The tape is first rewound to the BOT (Beginning-Of-Tape) mark. DMS then branches to the LABGN exit routine of the user program in which the volume header labels are to be checked. This is done with the aid of BTAM macros.

If no LABGN exit is defined in the EXLST macro, the labels are ignored.

If the labels are found to be invalid, the program is terminated with an OPEN error.

The LABGN routine is terminated by means of the LBRET macro.

Positioning

The user is responsible for correct positioning of the tape, i.e. (s)he must ensure that the tape is positioned to the file header labels before returning control to DMS. The user program must also evaluate the file header labels. You are also responsible for positioning these correctly.

Tape marks

Tape marks are ignored unless TPMARK=YES is specified in the file control block or TFT entry. If this is the case, DMS positions the tape after the tape mark. If TPMARK=NO is specified in the file control block or TFT entry, DMS assumes that the user has already positioned the tape to the first record (see above).

12.2.3 Files without labels

The processing of input files without labels is requested by means of the operand LABEL=NO of the FILE macro or the operand LABEL-TYPE=*NO of the ADD-FILE-LINK command.

Regardless of the user entry for LABEL or LABEL-TYPE, the volume header label is always checked unless the VOL1 check has already been performed for this tape.

Checking labels

DMS first checks whether the tape starts with a VOL1 label. If not, the job can access the file with no further checks (see section “Files without labels” below). If, in spite of the specification LABEL=NO or LABEL-TYPE=*NO, the tape starts with a VOL1 label, the validity of the VSN is first checked. As for files with standard labels, DMS issues either message `DMS0DA1` or, for foreign files, message `DMS0DFB` to the operator.

Next, DMS uses the VOL1 access indicator and owner identifier to check whether the user ID may access the tape (error message: `DMS0DD7`). DMS then reads the HDR labels and checks the access authorization on the basis of the HDR1 access indicator and the HDR3 owner identifier (error message: `DMS0DD7`). If the file is to be opened with OPEN INOUT/EXTEND, the HDR3 access type and the HDR1 expiration date are checked (error message: `DMS0DA3`). In interactive mode, DMS issues error message `DMS0DF6` to inform the user that the LABEL specification does not match the tape contents. The user can then decide whether he/she still wants to process the file, i.e. that the program is to be continued, or that the program is to be terminated with an OPEN error.

Positioning

If the first record on the tape is not a VOL1 label, but a tape mark, and if TPMARK=YES (see the FILE/FCB macro) or TAPE-MARK-WRITE=*YES (see the ADD-FILE-LINK command) was specified, DMS positions the tape after this tape mark. Within the tape, DMS executes positioning as specified in the FSEQ field of the catalog entry or the FILE or FCB macro, or as specified in the FILE-SEQUENCE operand of the ADD-FILE-LINK command.

12.3 Label processing for output files

Just as for input files, the volume and file header labels must be checked for output files. If the new file is the first file on a previously unused tape, the volume and file header labels (VOL1, HDR1, HDR2) were written during initialization of the tape.

Label processing depends – as for input files – on the LABEL specification in the FILE macro or the FCB, or on the LABEL-TYPE operand in the ADD-FILE-LINK command.

12.3.1 Files with standard labels

The processing of output files with standard labels is requested by means of the operand LABEL=(STD,n) of the FILE macro or the operand LABEL-TYPE=*STD(DIN-REVISION-NUMBER=n) of the ADD-FILE-LINK command.

As for input files, DMS skips to step 5 to process the file header labels if the VOL1 label of the tape has already been checked ("Files with standard labels").

Checking volume header labels

1. Is the first record on the tape a volume header label VOL1?

DMS checks the first four characters for the character string “VOL1”. If it does not find this string, the first record is not a VOL1 label, i.e. not a standard label. DMS issues message `DMS0DA2` on the console, and the operator must react to this message.

2. Has the user requested a specific tape or a work tape?

The user can (e.g. by means of DEVICE=WORK in the FILE macro or DEVICE-TYPE= WORK in the CREATE-FILE command) request a work tape, which the operator must allocate. The operator must also allocate a tape if the volume list which the user has defined for the file via the FILE macro (VOLUME operand) or via the VOLUME operand in the CREATE-FILE command contains no further tapes. In both cases, DMS issues message `DMS0DFB` on the console, and the operator must check the MOUNT operation again.

3. If the user has requested a specific tape: has the correct tape been allocated?

DMS checks whether the VSN in the VOL1 label matches the VSN which the user specified in the FILE macro (VOLUME=vsn) or in the CREATE-FILE command (VOLUME operand). If these two VSNs do not match, DMS issues message `DMS0DA1` on the console. Now the operator must decide how processing is to be continued.

4. Is access permitted only for the tape owner? If so: are the tape owner and the user ID the same?

DMS evaluates the contents of the VOL1 fields “access indicator” and “owner identifier”. If the user is not authorized to access the tape, DMS sends him/her message `DMS0DD7`. The user can now terminate his/her program or have label checking retried.

When a tape is initialized, the owner identifier is normally left empty or the computer center is entered as the tape owner and the tape is marked as shareable.

5. Is the next record to be read a HDR1 label?

After the VOL1 label check, DMS positions the tape as specified in the FSEQ operand of the FILE macro or in the FCB, or as specified in the FILE-SEQUENCE operand in the ADD-FILE-LINK command.

If DMS does not find a HDR1 label at this position, is starts the “measures to prevent implicit overwriting” (see step 9, "Files with standard labels"). If DMS finds a HDR1 label, it reads the HDR2 label and, if an existing file is to be overwritten, also the HDR3 label. If the tape is to be written afresh from the start, the next step is updating of the volume header labels (see "Files with standard labels").

6. May the requesting job (over)write the file?

DMS evaluates the access indicator (in the HDR1 label) and – if a HDR3 label has already been read – the owner identifier and checks whether the job is authorized to access the file. If this is not the case, DMS sends message `DSM0DD7` to the user.

7. May the file be overwritten?

DMS checks the expiration date and the access type in the HDR1 and HDR3 labels. If a retention period is defined (expiration date in HDR1 > current date) or if the access type in HDR3 indicates that the file is “read-only” (corresponding to ACCESS=READ), message `DMS0DA3` is sent to the user. If authorized, the user can ignore this message and proceed with file processing. Otherwise, he can have his program terminated with an error routine, have the contents of the label displayed, or have label checking retried, possibly with a different tape (for foreign files).

If the file may be overwritten, the password fields in the HDR3 label are not checked.

i In file sets, any files following a file which is overwritten are implicitly overwritten or erased. Their catalog entries still exist, but the data is no longer accessible.

8. Do the fields “file section number” and “file sequence number” in the HDR1 label contain valid values?

If an existing multivolume file is to be overwritten, the file section number in the HDR1 label must match the file section number specified in the FILE macro (VSEQ=) or in the ADD-FILE-LINK command (VOL-SEQUENCE-NUMBER operand). For the tape on which the file begins, VSEQ or VOL-SEQUENCE-NUMBER, as appropriate, must be 1. If VSEQ or VOL-SEQUENCE-NUMBER does not contain a valid value, OPEN processing is aborted unless FSEQ=0/FSEQ=1 or FILE-SEQUENCE=0/FILE-SEQUENCE=1 is also specified (explicitly or implicitly), i.e. if the new output file is not the first file of a file set.

9. Measures to prevent implicit overwriting

If “overwrite protection” (OPR) was specified in the SECLEV operand of the FILE macro or in the FCB, or with the OVERWRITE-PROTECTION=*YES operand of the ADD-FILE-LINK command, and if the current file is to be written as an element of a file set after an existing file (FSEQ=n, where n > 1), DMS compares the protection attributes “expiration date” and “access type” defined for the new file with those of the preceding file (FSEQ=n-1). It is taking the values from the appropriate fields of the labels EOF1 and EOF3. If the preceding file has no EOF3 label, ACCESS=WRITE is implied, i.e. write access is permitted. The tape is rejected if a retention period is defined for both the new and the preceding file and the expiration date for the new file is greater. The tape is also rejected if ACCESS=READ (only read access permitted) is defined for the new file but the preceding file (FSEQ=n-1) has ACCESS=WRITE (read and write access permitted).

Updating volume header labels

If a tape is to be written from the beginning (first file of a file set or processing of a continuation tape), and if the tape owner and the file owner are identical, the fields “access type” and “exchange level” of the VOL1 label are updated. The latter field indicates which labels are supported by default in accordance with the DIN 66029 exchange levels.

If the user program includes an OPENV routine for writing user volume labels (UVL), DMS then branches to this routine. If no OPENV exit is defined in the EXLST macro, no UVL labels are written. In register 0, DMS provides the address of an 80-byte area in which the labels can be created sequentially. The OPENV routine is terminated by means of the LBRET macro.

Writing/updating file header labels

The file protection attributes defined in the catalog entry are written into the HDR labels of the new file, together with the file set information specified in the FILE macro or ADD-FILE-LINK command: file set identifier of the first file, file section number (VSEQ/VOL-SEQUENCE-NUMBER) and file sequence number (FSEQ/FILE-SEQUENCE).

If the user program includes an LABGN routine for writing user header labels (UHL), DMS then branches to this routine. If no LABGN exit is defined in the EXLST macro, no UHL labels are written. When DMS branches to the LABGN routine, it provides, in register 0, the address of an 80-byte area in which the labels can be created sequentially. The LABGN routine is terminated by means of the LBRET macro.

Writing a tape mark

By default, DMS writes a tape mark after the file header labels.

12.3.2 Files with nonstandard labels

The processing of output files with nonstandard labels is requested by means of the operand LABEL=NSTD of the FILE macro or the operand LABEL-TYPE=*NON-STD of the ADD-FILE-LINK command.

For the most part, DMS handles files created with LABEL=NSTD exactly like files with standard labels (LABEL=(STD,n)).

Checking labels

If the first record on the tape is not a VOL1 label, DMS permits file access without further checks; the tape is immediately positioned to the beginning of tape (see step 2, "Files with standard labels").

If the first record is a VOL1 label, DMS checks the VSN in this label, and in the case of an error DMS issues message `DMS0DA1` on the console. If the job has the necessary TPIGNORE authorization, the operator can ignore this message; otherwise, he/she must take the appropriate action.

If the user has requested a work tape, or if the volume list for the file is exhausted, the operator is requested via message `DMS0DFB` to assign a tape.

If the VOL1 label contains the correct VSN, DMS then checks file header labels HDR1 through HDR3. Thereby evaluating the fields “access indicator” (HDR1), “file owner” (HDR3), “expiration date” (HDR1) and “access type” (HDR3). If the job is not authorized to access the file, the user receives DMS message `DMS0DD7` or `DMS0DA3`.

If he/she has the necessary TPIGNORE authorization, he can ignore the error message and proceed with file processing. Otherwise, he/she must react in a suitable manner to the message (terminate program, display label, retry label checking).

Positioning

If file access is permitted, the tape is rewound to the BOT mark.

Writing labels

First, DMS branches to the OPENV routine, which writes the nonstandard user volume labels (UVLs). In register 0, DMS provides the address of the area in which the labels can be created sequentially. The format for nonstandard user volume labels is not fixed, i.e. it can be defined by the user. The OPEN routine is terminated using the EXRTN macro. However, the tape must be positioned correctly before file processing can be continued. If the user program has no OPENV routine for writing UVL labels, DMS assumes that the tape is already correctly positioned.

Writing a tape mark

If TPMARK=YES or TAPE-MARK-WRITE=*YES was specified for the file, DMS now writes a tape mark. If no tape mark is wanted, the user must specify TPMARK=NO in the FILE macro or in the FCB, or TAPE-MARK-WRITE=*BY-PROGRAM in the ADD-FILE-LINK command.

12.3.3 Files without labels

The processing of output files without labels is requested by means of the operand LABEL=NO of the FILE macro or the operand LABEL-TYPE=*NO of the ADD-FILE-LINK command.

DMS expects a file without labels to be created. Nevertheless, it first checks whether the tape already contains standard labels. If a VOL1 label is found, DMS proceeds to check the other labels. If there is no VOL1 label on the tape, the tape is immediately positioned and the user can proceed with file processing.

Checking labels

If the tape contains a VOL1 label, DMS first checks its VSN (and issues message `DMS0DA1` to the operator in the case of an error). If the user has not requested a specific tape (DEVICE or DEVICE-TYPE=WORK), or if the volume list (VOLUME=) is exhausted, the operator is requested via message `DMS0DFB` to assign a tape to the job.

DMS then checks the HDR labels of the first file on the tape in order to determine whether file access is permitted. In the event of an error, DMS sends message `DM0DD7` to the user. If authorized, the user can ignore this message.

Next, the access type (HDR3) and the expiration date (HDR1) are checked. If a retention period still exists for the file on the tape, or if the access type is read-only (corresponding to ACCESS=READ), DMS sends message `DMS0DA3` to the user. If authorized, the user can ignore this message and proceed with file processing.

Finally, the operator is asked whether the user may write a file without labels on a standard tape. The operator can permit or forbid this; in the latter case, an error routine in the user program is called and the program is terminated.

Positioning and writing a tape mark

The tape is positioned as specified in the FSEQ operand of the FILE command, in the FILE-SEQUENCE operand of the ADD-FILE-LINK command, or according to the FSEQ value in the catalog entry.
If FSEQ=1 or FILE-SEQUENCE=1 applies, i.e. if this is the first file on the tape, the tape is positioned to BOT (Beginning-Of-Tape) and, if TPMARK=YES or TAPE-MARK-WRITE=*YES applies, a tape mark is written.

13 CLOSE processing

Each file which is opened by a program must be closed after file processing has been completed. In the CLOSE macro, the programmer can decide whether he/she wants to close only one or several of the files still open in the program.

If the programmer “forgets” to write the necessary CLOSE macro, all files opened in a program but not explicitly closed are closed automatically when the program is terminated. It is only in the case of a system crash that a file may not be closed correctly.

By means of the VERIF macro or the CHECK-FILE-CONSISTENCY and REPAIR-DISK-FILES commands, the user can “recover” such files to such a degree that file access is again possible (see section “File recovery”).

13.1 CLOSE processing for disk files

DMS waits for any outstanding write operations to be completed and then releases all PAM pages which are still locked for the calling job in the file to be closed. Finally, it restores the contents of the file control block to the state which they had before the OPEN call and releases any TFT entry created automatically by OPEN for the file.

DMS releases all work areas in class 5 memory which were occupied during file processing. Such as the input /output areas which were automatically allocated to the file when it was opened.

The date and time of access (ACC-DATE and ACC-TIME fields) and the access counter (ACC-COUNT field) are updated in the catalog entry of any file to be closed.

If the file was *opened for writing*, the date and time of the last update (CHANG-DATE and CHANG-TIME fields), the highest used page number (HIGH-US-PA field), and the number of the highest valid byte in the last logical block (last-byte pointer) are also updated. The highest used page number is the last PAM page used by the file and thus indicates the logical file size.

For *newly created files*, i.e. files created with OPEN mode OUTIN or OUTPUT, the date and time at which the file was created (CRE-DATE and CRE-TIME fields) and the date and time of expiration (EXPIR-DATE and EXPIR-TIME fields) are filled in as well.

The SHOW-FILE-ATTRIBUTES command or the FSTAT macro can be used to determine the above-mentioned fields and other fields from the catalog entry.

13.2 CLOSE processing for tape files

CLOSE processing for tape files runs along the same lines as that described above for disk files. However, for tape files, labels have to be written and the tape positioned. For output files, DMS automatically writes the standard EOF labels as specified in the LABEL operand in the FILE/FCB macro or in the LABEL-TYPE operand in the ADD-FILE-LINK command. The tape is then positioned as specified by the user in the CLOSE macro.

If user labels (UTL) are to be written, the program must contain the necessary LABEND routine for this. However, DMS does not branch to this LABEND routine if several files are closed at the same time (CLOSE ALL).

Just as for OPEN processing, the way the labels are processed when a file is closed depends on the LABEL operand in the FILE macro or in the FCB, or on the LABEL-TYPE operand in the ADD-FILE-LINK command, and on the OPEN mode of the file.

If a file is closed with LEAVE specified in the CLOSE macro, the tape remains positioned at the logical end-of-file. If the following file is opened with the same file link name, the tape is repositioned to the start of the tape due to the implicit device release and the tape rewind operation initiated by the FILE macro with the LINK specification or by the ADD-FILE-LINK command with the LINK-NAME specification. The user can suppress tape rewinding by creating a TFT entry for a dummy file with a different file link name before the start of processing or by using different file link names when processing the files.

Closing input files

Label processing

For input files (OPEN INPUT), label processing is not necessary, since no file attributes are changed. The tape is simply positioned as specified in the CLOSE macro.

BYPASS handling

Input files can be positioned with the aid of the BYPASS operand of the FILE macro or the BYPASS-LABEL-CHECK operand of the ADD-FILE-LINK command. However, the user must remember that the use of the BYPASS operand affects how the tape is positioned at the end of CLOSE processing: the LEAVE operand must be specified; otherwise, the tape is rewound to the beginning. If the DISCON operand is specified, the tape is unloaded after it has been rewound.

Processing of tapes without EOF/EOV labels

A user job which processes a tape with standard labels, but without correct EOF/EOV labels, is normally returned a label error (EXLST exit LABERR) when a tape mark is detected. As in exceptional cases (non-Fujitsu) tapes have standard labels but no EOF/EOV labels, BS2000 supports the processing of such tapes The files can be processed with the SAM access method if, for example, the Assembler program contains routines for error handling (LABERR error).

Processing of tape files with standard labels and without EOF/EOV labels is possible only under the following conditions:

- The file must be opened correctly, i.e. the tape must not be positioned, at OPEN time, to a file which follows a file without EOF labels.
- The file must not be opened with OPEN REVERSE or EXTEND.

Closing output files

When an output file is closed, it is generally necessary to write labels as specified in the LABEL operand of the FILE macro or in the FCB or as specified in the LABEL-TYPE operand of the ADD-FILE-LINK command.

File with standard labels:
*LABEL=(STD,n) or LABEL-TYPE=*STD(DIN-REVISION-NUMBER)*

DMS automatically writes tape marks and – except in the case of BTAM files opened in SINOUT mode – the labels EOF1 through EOF3; labels EOF4 through EOF9 are not written under BS2000. With BTAM files which were opened in SINOUT mode, no trailer labels can be written because there was no label processing at the time of opening (OPEN) and therefore no information is available on the creation of trailer labels. If the file is to have user trailer labels (UTL), the program must have a LABEND routine to write them. As the BLIM function and BLOCK-LIMIT function in the ADD-FILE-LINK command are not effective, no checkpoint file is created. The tape is positioned as specified in the CLOSE macro.
If DESTOC=YES was specified in the FILE macro, or if DESTROY-OLD-CONTENTS=*YES was specified in the ADD-FILE-LINK command, or if the catalog entry contains DESTROY=YES, the remainder of the tape is erased.

File with nonstandard labels:
*LABEL=NSTD or LABEL-TYPE=*NON-STD*

Nonstandard labels must be created by a LABEND routine in the user program. Before calling this label routine, DMS writes a tape mark. If TPMARK=YES was specified in the FILE macro or TAPE-MARK-WRITE=*YES in the ADD-FILE-LINK command, DMS writes a double tape mark. If TPMARK=NO was specified in the TFT entry and in the file control block, DMS writes only one tape mark. The tape is then positioned as specified in the CLOSE macro.
If DESTOC=YES was specified in the FILE macro, or if DESTROY-OLD-CONTENTS=*YES was specified in the ADD-FILE-LINK command, or if the catalog entry contains DESTROY=YES, the remainder of the tape is erased.

File without labels:
*LABEL=NO or LABEL-TYPE=*NO*

DMS writes a double tape mark to indicate end-of-file and end-of-volume. If DESTOC=YES was specified in the FILE macro, or if DESTROY-OLD-CONTENTS=*YES was specified in the ADD-FILE-LINK command, or if the catalog entry contains DESTROY=YES, the remainder of the tape is erased.

14 EOV processing

EOV (End Of Volume) processing is initiated for SAM tape files whenever a tape swap is required while reading or writing a tape file which extends over several tapes. It can be initiated explicitly for BTAM or SAM tape files by means of the FEOV macro. Just like label processing when a file is opened or closed, EOV processing depends on the OPEN mode of the file (input or output file) and on the specification in the LABEL operand of the FILE macro or the FCB, or on the specification in the LABEL-TYPE operand in the ADD-FILE-LINK command.

The structure of the routines for creating UTL, UVL and UHL labels is described in section “EXLST exits for processing labels for tape files”.

14.1 Input files

EOV processing is initiated implicitly when DMS detects a tape mark instead of a data block or record. The sequence of EOV processing depends, amongst other things, on the BYPASS operand of the FILE macro or the BYPASS-LABEL-CHECK operand in the ADD-FILE-LINK command (see section “BYPASS handling”). If it is initiated explicitly, DMS positions the tape behind the next tape mark.

Which labels are to be checked during EOV processing depends on the direction in which the file is being read. If the file is being read normally from the start of the file toward the end, the EOF/EOV/UTL labels are checked. If the file is being read normally from the start of the file towards the end, the EOF/EOV/UTL labels are checked. If the file is being read backwards (REVERSE), the HDR and UHL labels are treated like end-of-volume and endof-file labels. If errors are detected, DMS activates a CLOSER routine.

14.1.1 Files with standard labels

The processing of standard labels is requested by means of:

- the operand LABEL=(STD,n) of the FILE macro
- the operand LABEL-TYPE=*STD(DIN-REVISION-NUMBER=n) of the ADD-FILE-LINK command

DMS checks the record which follows the tape mark:

- If this is neither an end-of-file nor an end-of-volume label (EOF1 or EOV1), DMS activates the LABERR routine. If the LABERR exit of the EXLST macro is not specified, an appropriate error message (DMS0DE9) is sent to the user. The user can then either terminate his/her program with a CLOSER routine or opt to continue processing in spite of the error.
- If the tape mark is followed by EOF or EOV labels and a double tape mark, DMS compares the contents of the label field “Block counter” (in EOF1 or EOV1) with the current value of the block counter within DMS. If the two values are not the same, DMS activates a LABERR routine in the user program or issues error message `DMS0DE6`. The user can then either terminate his/her program with a CLOSER routine or continue processing.

Any further standard EOF or EOV labels are skipped. If the tape contains user trailer labels (UTL), DMS activates the LABEND or LABEOV routine in the user program. It distinguishes between a UTL used as an end-of-file label and a UTL used as an end-of-volume label on the basis of the preceding group of standard labels (EOF or EOV).

If DMS has read an EOF label, it initiates a tape swap only if a temporary volume list was created via the TVSN operand of the FILE macro or via the PROCESS-VOLUME operand of the ADD-FILE-LINK command, and this list contains further VSNs. If this list is exhausted, or if the appropriate operand was not specified in the macro or command, DMS detects the “end-of-file” condition. At this time, the tape is positioned to the tape mark following the EOF labels.

If DMS has read an EOV label, it initiates a tape swap if the volume list of the TFT contains further VSNs. This TFT volume list is affected by the VSEQ operand of the FILE macro and by the VOL-SEQUENCE-NUMBER operand of the ADD-FILE-LINK command. If the input file is a foreign file, a tape swap is always initiated.

DMS activates an EOVCTRL routine (if defined in the EXLST macro) in which the user can, for example, request writing of a checkpoint. The EOVCTRL routine is terminated by means of an EXRTN macro.

The operator must change the tapes. For this purpose, DMS issues a message
(DMS0DE4) on the console. The operator can then assign the next tape and allow processing to continue, or he
/she can terminate the user program (or activate the NODEV exit of the EXLST macro).

After the tape has been swapped, processing of the header labels on the new tape is initiated (see section “Label processing for input files”).

14.1.2 Files with nonstandard labels

The processing of nonstandard labels is requested by means of:

- the operand LABEL=NSTD of the FILE macro
- the operand LABEL-TYPE=*NON-STD of the ADD-FILE-LINK command

DMS controls EOV processing by activating the appropriate routines in the user program, which then handle label processing.

When DMS detects a tape mark, it activates the user routine in order to evaluate the EOV labels. After the labels have been evaluated, DMS – if necessary executes the tape swap with messages to the operator, etc. Once the tape has been swapped, the header labels on the new tape are checked before the user program can continue reading the file.

If no tape swap is necessary, DMS activates the LABEND routine in order to evaluate the end-of-file labels. After this has been done, DMS positions the tape backwards by one tape mark and then initiates end-of-file processing (using the EOFADDR exit of the EXLST macro).

14.1.3 Files without labels

Processing without labels is requested by means of:

- the operand LABEL=NO of the FILE macro
- the operand LABEL-TYPE=*NO of the ADD-FILE-LINK command

After it has detected the first tape mark, which initiated EOV processing, DMS checks whether a second tape mark follows and whether the volume list of the FILE macro or ADD-FILE-LINK command contains further volumes.

If there is no double tape mark on the tape, and if the volume list is not yet exhausted, DMS initiates a tape swap. It then checks the beginning of the new tape and then permits the user program to continue processing the file.

If DMS finds a double tape mark on the tape, or if no further tape is needed, DMS positions the tape after the last tape mark which was read and activates the end-of-file routine in the user program (using the EOFADDR exit of the EXLST macro).

14.1.4 BYPASS handling

When a tape mark is detected in a SAM file, SAM initiates EOV processing internally. It can also be initiated explicitly by means of the FEOV macro.

Checking of EOF labels is suppressed, which means that no distinction is made between EOF and EOV labels and that the double tape mark which follows the last file without standard labels on the file is not detected.

A tape swap is executed only if the TFT volume list contains a further VSN. In this case, the new tape is positioned to the start of its data, depending on the TPMARK or TAPE-MARK-WRITE specification:

- TPMARK=NO to the beginning of tape
- TPMARK=YES after the first tape mark

If LABEL=(STD,n) or LABEL-TYPE=*STD(DIN-REVISION-NUMBER) applies, TPMARK is ignored).

If the TFT volume list contains no further VSN (= end of file), the tape is positioned behind the last tape mark which was read.

14.2 Output files

- Files with standard labels
- Files with nonstandard labels
- Files without labels

14.2.1 Files with standard labels

The processing of standard labels is requested by means of:

- the operand LABEL=(STD,n) of the FILE macro
- the operand LABEL-TYPE=*STD(DIN-REVISION-NUMBER=n) of the ADD-FILE-LINK command

In the case of SAM files, the EOV processing routine is started implicitly when DMS detects a tape mark or when the block limit specified in the BLIM operand of the FILE/FCB macro or in the BLOCK-LIMIT operand of the ADD-FILE-LINK command is reached. The routine can also be started explicitly by means of the FEOV macro.

Processing steps

- Process outstanding inputs and outputs; for SAM files, the last data block that has been written to the tape may be only partially filled.
- Write a tape mark.
- Write EOV labels (to match the HDR labels).
- Write UTL labels (LABEOV routine).
- Write a double tape mark.
- If requested: write a checkpoint at the end of the tape.
- If DESTOC=YES (from the FILE call), DESTROY-OLD-CONTENTS (from the ADD-FILE-LINK command), or DESTROY=YES (in the catalog entry) has been specified, overwrite the remaining data on the tape.
- Initiate a tape swap: depending on the contents of the volume list in the TFT/TST, the next tape is requested or the operator is asked to mount a work tape.
- Initiate volume header processing for the new tape.

If a non-BS2000 tape file is extended (OPEN INOUT/EXTEND), it should be noted that the existing part of the file may contain standard labels which are not supported by BS2000, such as labels HDR4 through HDR9. BS2000 writes a maximum of three EOV labels (EOV through EOV3) at the end of the new file section and a maximum of three file header labels (HDR1 through HDR3) at the beginning of any following file sections (and, correspondingly, a maximum of three EOF labels at the end of the file).

14.2.2 Files with nonstandard labels

The processing of nonstandard labels is requested by means of:

- the operand LABEL=NSTD of the FILE macro
- the operand LABEL-TYPE=*NON-STD of the ADD-FILE-LINK command

EOV processing is started either implicitly when the end-of-tape mark is detected or explicitly by an FEOV macro.

Processing steps

- Process any outstanding inputs and outputs and write the last block to the tape.
- Write a tape mark if TPMARK=YES or TAPE-MARK-WRITE=*YES is specified.
- Branch to the LABEOV routine in the user program.
- Write a double tape mark if TPMARK=YES or TAPE-MARK-WRITE=*YES is specified.
- If DESTOC=YES was specified in the FILE macro, or DESTROY-OLD-CONTENTS was specified in the ADD-FILE-LINK command, or if the catalog entry contains DESTROY=YES: erase the remainder of the tape.
- Change the tape: request the next tape specified in the volume list or ask the operator to mount a work tape.
- Execute volume header processing for the next tape.

File processing is continued only after completion of the volume header processing for the newly assigned tape.

14.2.3 Files without labels

Processing without labels is requested by means of:

- the operand LABEL=NO of the FILE macro
- the operand LABEL-TYPE=*NO of the ADD-FILE-LINK command

EOV processing is started either implicitly if DMS detects the end-of-tape mark or explicitly by means of the FEOV macro.

Processing steps

- Process any outstanding inputs and outputs and write the last block to the tape.
- Write a double tape mark.
- If DESTOC=YES was specified in the FILE macro, or DESTROY-OLD-CONTENTS was specified in the ADD-FILE-LINK command, or if the catalog entry contains DESTROY=YES: erase the remainder of the tape.
- Change the tape: request the next tape specified in the volume list or ask the operator to mount a work tape.
- Execute volume header processing for the next tape.

File processing is continued only after completion of the volume header processing for the newly assigned tape.

15 Handling of errors and special events

During file processing, situations which require a decision may occur, such as end-of-file during reading, an error or a conflict when two or more jobs attempt to update the same record concurrently.

In order to permit DMS to communicate with the user job in such situations, an exit address must be specified in the FCB macro (operand EXIT). If this address is not specified, DMS will abort the program.

15.1 Exit address in the FCB

Pointer to a global program routine

The global routine is responsible for processing all errors which can occur. In this case, the address must be enclosed in parentheses, i.e. EXIT=(addr).

The exit routine can evaluate the FCB fields ID1XITB and ID1ECB: ID1XITB indicates the cause of the error, while ID1ECB contains a DMS error code (see “Error message codes” in the appendix of the “DMS Macros” manual [1 (Related publications)]).

The routine can also be used to terminate the program normally (closing the files) or to request memory dumps (using the `CDUMP2` or `TERM, DUMP=Y` macros, see the “Executive Macros” manual [2 (Related publications)]).

Pointer to the EXLST macro

The exit address in the FCB can also address an EXLST macro. In this case, the address in the EXIT operand must be specified without parentheses, i.e. EXIT=addr. The EXLST macro can be used, for example, to check for specific events or errors and to initiate the necessary special measures, while all other events and errors are handled globally in a common routine.

Events for which no separate EXLST exit is to be defined can be intercepted by means of the COMMON operand. The COMMON routine then executes error handling, etc. Specification of the COMMON operand also avoids abortion of the program if an event occurs for which no separate EXLST exit is defined.

Note

If the FCB is found to be incorrect at OPEN time (e.g. storage area not allocated, currently active file processing with the same FCB address, etc.), this mechanism does not take effect. In such a case the program is always terminated abnormally with error code `DMS0D9F`.

15.2 Exit routines: EXLST macro

Using the operands of the EXLST macro, the user can define an error exit for each event, i.e. he/she can specify the address of a program routine in which the error is handled or, at least, the program is terminated normally (closing the open files, ...). If the EXLST exits are used, it is not necessary to evaluate the FCB field ID1XITB, since each EXLST exit corresponds to precisely one event.

DMS places the address of the file control block in general register 1 before passing control to the user program. This makes it possible to use the same error handling routines for different files.

DMS also enters the address of the instruction which would have been processed next (if no error had occurred) in field ID1RTNAD of the FCB. This address points to the next instruction in the user program.

The type of event is indicated by the contents of field ID1XITB in the FCB. The codes in this field are shown in the table (see the “DMS Macros” manual [1 (Related publications)]).

For the EXLST exits EOVCTRL, ERROPT, LABERR, OPENV, OPENX, OPENZ and WLRERR, the EXRTN macro must be called to resume processing; for LABEND, LABEOV and LABGN, the LBRET macro is used.

If a special status occurs during the processing of a file, DMS enters the appropriate values in fields ID1XITB and ID1ECB of the associated TU FCB. These fields are not automatically reset, but are simply updated if another special status is detected.

15.2.1 EXLST exits for OPEN processing

On the one hand, errors may occur during OPEN processing and these can be intercepted by means of the EXLST operands. On the other hand, the user can – again with the aid of EXLST operands – modify the FCB or create tape labels at specific points during OPEN processing.

LOCK	The file is to be opened with write access, but it has already been opened by another user.
NODEV	The volumes on which the file is located cannot be readied.
NOSPACE	There is not enough memory for the secondary allocation specified with OPEN=EXTEND.
OPENC	The file was opened as an output file, but was not closed properly.
OPENER	Error when opening a file; the user can analyze the error with the aid of the error code in the FCB.
OPENV	For tape files: the user program checks or writes UVL labels or nonstandard volume header labels.
OPENX	The contents of the FCB can be checked and modified; the changes are included (for output files) in the catalog entry and the labels.
OPENZ	For output files, the FCB can be modified without changing the catalog entry or the labels.
PASSER	An incorrect password was specified for a protected file.

15.2.2 EXLST exits for processing labels for tape files

The following applies to all exits described here:

- DMS places the address of an 80-b0te buffer in register 0. For input files, DMS places the contents of the user labels in this buffer. For output files, it takes the contents of the user labels from this buffer.
- To resume program execution, the LBRET macro must be called after the processing of UVL, UHL or UTL labels; in all other cases, the EXRTN macro must be called.
- For files with nonstandard labels, the volume and file labels are processed using BTAM macros, regardless of the access method selected for file processing.
- For standard user labels, writing is executed automatically.

EXLST exits control the writing of user labels or nonstandard labels and can also be used to intercept any errors which occur during label checking.

OPENV	The user program checks or writes UVL labels or nonstandard volume header labels.
LABGN	The user program checks or writes UHL labels or nonstandard file header labels.
LABEND	The user program checks or writes user/nonstandard end-of-file labels. DMS distinguishes between UTL end-of-file labels and UTL end-of-volume labels on the basis of the preceding group of system labels (EOF, EOV).
LABEOV	The user program checks or writes user/nonstandard end-of-volume labels. EOV processing is initiated either automatically (for SAM only) or by means of an FEOV macro (for SAM and BTAM). For BTAM processing, the user must evaluate the field ERRBYTE, which indicates “end-of-volume” and then initiate a tape swap by means of the FEOV macro (unless he/she has already initiated a tape swap).
LABERR	The LABERR routine is activated if a label error occurs during the processing of files with standard labels, e.g. if EOF/EOV labels are missing or, for input files, if the value of the block counter in the EOF/EOV label differs from that of the block counter within DMS. The user must place an “action code” in register 0 to control further processing. The LABERR routine is terminated by means of the EXRTN macro. If the LABERR exit is not defined, DMS issues an error message. The user can then decide whether to continue program execution or to activate a CLOSE error routine.

15.2.3 EXLST exits and action macros

The table below provides an overview of the possible action macros which you can call in the various EXLST exits.

Detailed information on the error exits and the action macros is provided in the “DMS Macros” manual [1 (Related publications)].

Error exit	**Possible action macros**
CLOSEPOS	BTAM, CLOSE, EXRTN
DLOCK	FCB
DUPEKY	INSRT, PUT
EOFADDR	CLOSE, GET, GETR, GETFL (LIMIT = END), FEOV
ERRADDR	BTAM, FILE
ISPERR	FILE
LABEND	LBRET, CLOSE, EXRTN
LABEOV	FEOV, EXRTN, LBRET
LABERR	EXTRN, CLOSE
LABGN	LBRET, OPEN, EXRTN
LOCK	OPEN
NOFIND	GETKY, ELIM (with KEY specification), GETFL (LIMIT=KEY)
NOSPACE	OPEN (OUTPUT/EXTEND)
OPENC	VERIFY
OPENV	EXRTN, BTAM
PGLOCK	PUTX, ELIM (without KEY specification), RETRY, FCB
SEQCHK	PUT
USERERR	PUTX, ELIM (without KEY specification), GETFL, SETL

Table 31: EXLST exits and action macros

15.3 DMS error codes

The DMS error codes permit precise analysis of errors, even within programs. They are entered in the FCB field ID1ECB.

The DMS error codes are documented in the description of the IDEMS macro.
However, the texts describing the various errors are relatively short. More detailed information can be requested using the command or SDF standard statement HELP-MSG-INFORMATION (see the “Commands” manual [3 (Related publications)]).

16 Access methods

The access methods of DMS transfer data between a file and the address space of a job. Technically, this is a data transfer between the peripheral devices and the main memory of the CPU. DMS handles the devices and the volumes and provides the user with an interface for access to records and/or data blocks.

For access by a user program to the contents of a file, a distinction can be made between record-oriented and block-oriented access and between the operating modes “move” and “locate”.

For all access methods, DMS transfers data blocks between the peripheral device and main memory. The area of main memory in which one or more blocks are stored, or from which blocks are transferred to the peripheral device, is called a buffer. The buffer is part of the address space of the job which initiates the I/O operation. In the case of NK-ISAM processing, the buffers are located in ISAM pools (see "ISAM pools"). For the record-oriented processing of a file in move mode, DMS transfers the records from the buffer to the work area of the program.

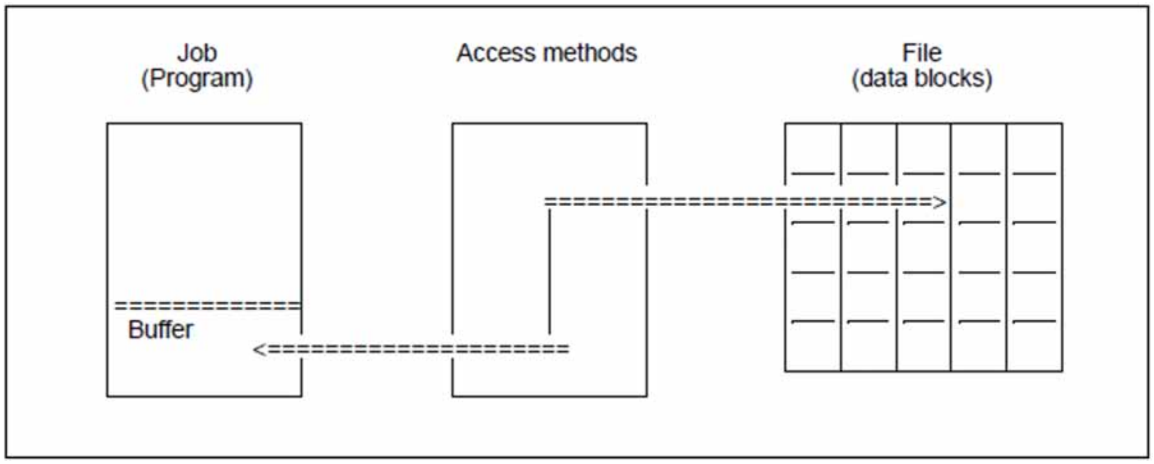


Figure 19: Transfer of data blocks

16.1 Privileged access methods (PPAM/PTAM)

The two main BS2000 access methods are PPAM (Privileged Primary Access Method) and PTAM (Privileged Tape Access Method). PPAM and PTAM are used by DMS to handle file access. All file access requests are passed on to PPAM/PTAM by DMS.

PPAM handles all disk file access requests. PPAM uses standard blocks, known as PAM pages (also called PAM blocks or half-pages (HPs)). It supports the access methods ISAM, SAM, UPAM, FASTPAM, DIV and EAM for disk files.

PTAM handles all tape file access requests. PTAM supports not only the tape access method BTAM but also the access methods SAM and UPAM if these methods are used for processing tape files. PTAM functions independently of PPAM. From the viewpoint of PTAM, the content of a tape consists of a sequence of data blocks and tape marks.

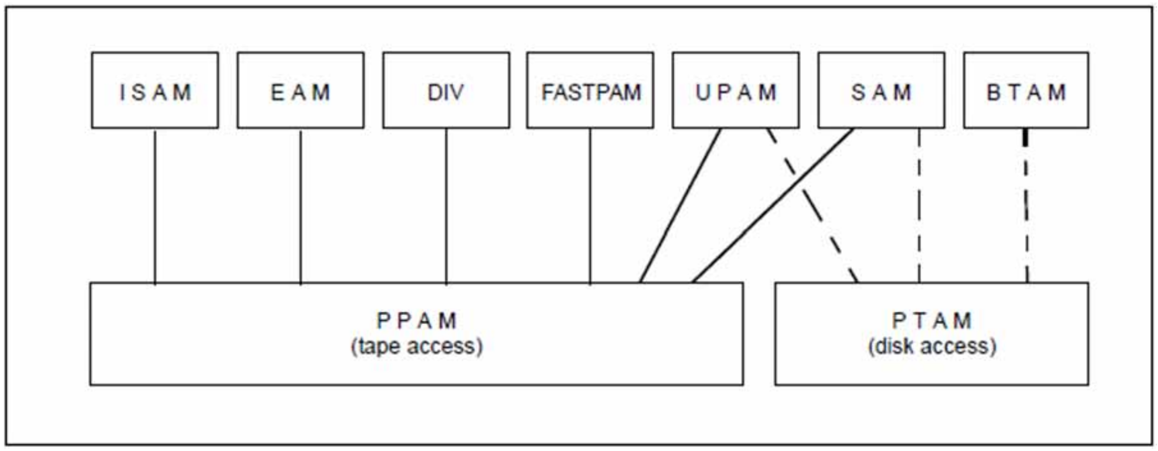


Figure 20: PPAM / PTAM

16.2 Overview of nonprivileged access methods

The selection of an access method depends on the requirements for each specific problem to be solved and on the special features of the individual access methods. These access methods are dealt with in detail in corresponding sections A comparative overview of the features of the nonprivileged access methods is provided below.

Selection based on volume

Volumes	ISAM	SAM	BTAM	UPAM	FASTPAM	DIV	EAM
Public disk							
K disk	x	x	-	x	x	x	x
• NK2 disk	x	x	-	x	x	x	x
• NK4 disk	x	x	-	x	x	x	
Private disks							
• NK2 disk	x	x	-	x	x	x	x
• NK4 disk	x	x	-	x	x	x	x
Tape		x	x	x			

Table 32: Access method selection based on volume

Selection based on access mode

Access method	ISAM	SAM	BTAM	UPAM	FASTPAM	DIV	EAM
Block-oriented	-	-	x	x	x	x	x
Record-oriented	x	x	-	-	-	-	-

Table 33: Access method selection based on access mode

Selection based on record type

Record type	ISAM	SAM
Variable	x	x
Fixed	x	x
Undefined	-	x

Table 34: Access method selection based on record type

Selection based on data structure

Definition of data structure	ISAM	SAM	BTAM	UPAM	FASTPAM	DIV	EAM

by DMS	x	x	-	-	-	-	-
by the user	-	-	x	x	x	x	-

Table 35: Access method selection based on data structure

Selection based on lifespan of file

Lifespan of the file	ISAM	SAM	BTAM	UPAM	FASTPAM	DIV	EAM
Permanent file	x	x	x	x	x	x	
Temporary file	-	-	-	-	-	-	x

Table 36: Access method selection based on lifespan of file

Selection based on multi-user mode

Multi-user mode	ISAM	SAM	BTAM	UPAM	FASTPAM	DIV	EAM
Processing	x	-	-	x	x	x	-

Table 37: Access method selection based multi-user mode

Advantages and uses of individual access methods

ISAM – Indexed-Sequential Access Method

ISAM is a record-oriented access method for disk files which allows records to be accessed via predefined (primary and secondary) keys. Files can also be processed sequentially with ISAM.

SAM – Sequential Access Method

SAM is an access method that supports the sequential reading and writing of logical records.

UPAM – User Primary Access Method

UPAM is a block-oriented access method in BS2000 for random access to disk files. Read or write access to any block of a file is possible at any time. The basis for file processing is a standard block; record structures are not supported.

BTAM – Basic Tape Access Method

The access method BTAM is used to save and retrieve block-oriented data in a sequentially organized tape file. BTAM can also be used to process tape files which were not created in a BS2000 system, provided such files comply with the hardware conventions for recording data on magnetic tapes.

FASTPAM – Fast Primary Access Method

FASTPAM (FAST Primary Access Method) is a block-access method for NK4 disk files. It is characterized by the following features:

- reduced path lengths for I/O execution,
- a clear and efficient interface, and
- support for I/Os in data spaces.

DIV – Data-In-Virtual

DIV is an access method that differs from the access methods ISAM, SAM and UPAM in that it does not require files to be structured in records or blocks and works without I/O buffers and operations such as GET, PUT, etc.

Parts of a file or an entire file are mapped in a virtual address space by the DIV function MAP. Then data from the file can be accessed using CPU instructions (MVC, CLI,...).

EAM – Evanescent Access Method

EAM is the access method for EAM files in BS2000. EAM files are not cataloged. As a result of this, no disk access is necessary when an EAM file is opened. EAM files are automatically deleted on completion of a task.

16.3 Access modes

- Block-oriented access
- Record-oriented access

16.3.1 Block-oriented access

The user program processes the file block-by-block (e.g. with the access method UPAM). Blocking and deblocking of records is thus not necessary. The user can speed up processing by using:

- exchange buffer operation or overlapping input/output

  While the program is processing the contents of one buffer, DMS executes the next I/O operation. If, for example, the program is reading buffer A, DMS can transfer the next data block to buffer B; when the program has finished processing buffer A, it can simply switch to buffer B, and so on.

- chained input/output

  DMS transfers up to 16 PAM pages or blocks between the peripheral device and the main memory at one time; time is saved by the reduced number of input/output operations needed for this.

16.3.2 Record-oriented access

The basis of file processing is the record. However, data blocks are still transferred between the peripheral device and the buffer area. DMS handles the necessary blocking and deblocking of records and makes the required records available to the user program. If the record required by a read operation is not in the buffer, or if a write operation fails because the buffer is full, a data transfer is initiated: DMS reads the next data block or writes the buffer contents as a new data block, respectively.

In the case of record-oriented access, a distinction is made between move mode and locate mode.

Move mode

An I/O area is defined in the program. For read operations, DMS transfers the required record from the buffer to this I/O area. For write operations, it transfers the record from the I/O area to the buffer. The figure below illustrates a write operation in the form of a diagram.

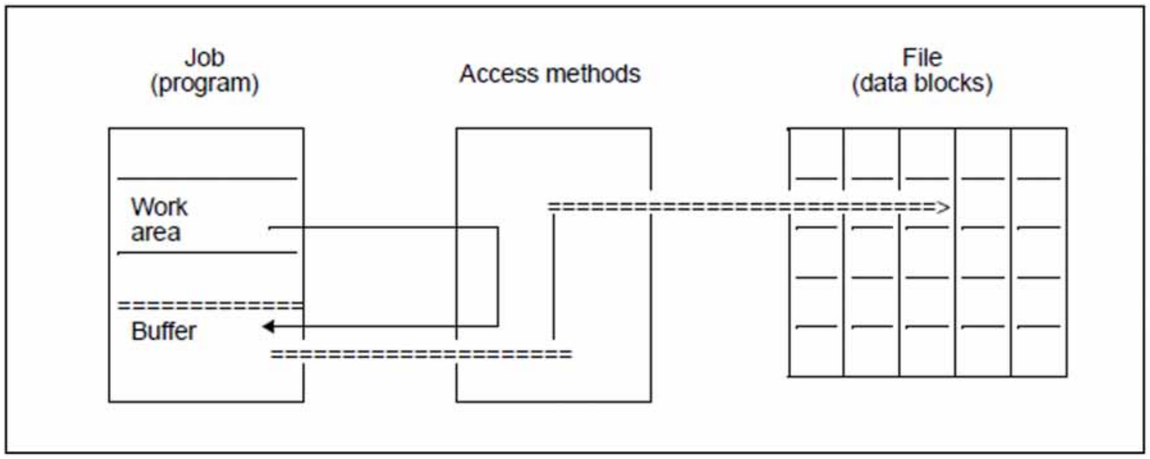


Figure 21: Move mode for record-oriented access

Locate mode

There is no I/O area defined in the program. No records are transferred between the buffer and an I/O area. Instead, the records are processed directly in the buffer. For a read operation, DMS returns the address of the required record. For write operations, it returns the address at which the new record is to be written. The figure below illustrates a write operation in the form of a diagram. A pointer points to the address in the buffer at which the record is to be created.

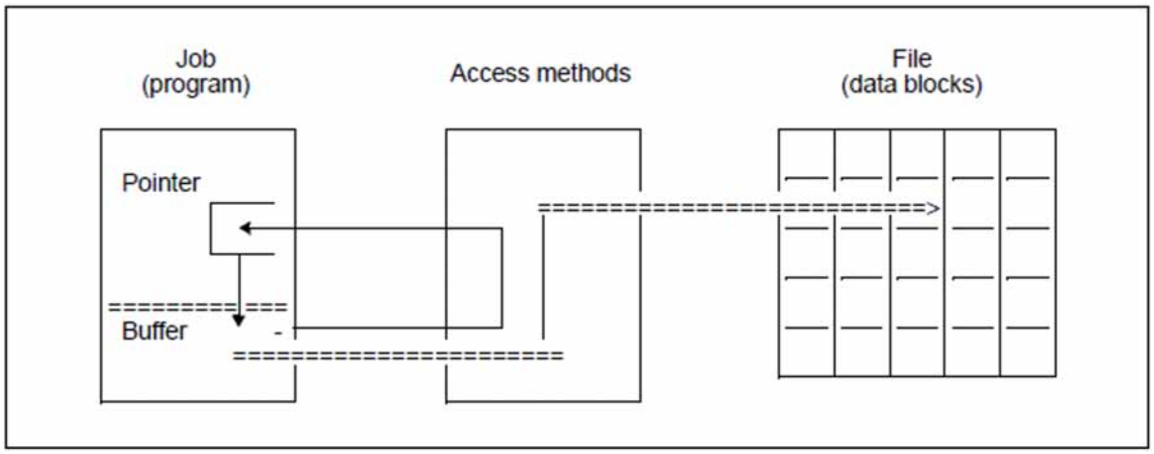


Figure 22: Ortungsbetrieb bei satzorientiertem Zugriff

16.4 File and processing attributes

For all access methods, the file attributes are defined when a file is created, i.e. when it is opened as an output file (OPEN OUTPUT/OUTIN). The file attributes are placed in the catalog entry for the file and are valid for all subsequent processing of the file. They can be defined either by means of an ADD-FILE-LINK command (LINK-NAME operand) or in the file definition in a program.

If an existing file is to be processed, the null operand function of the ADD-FILE-LINK command (operand *BY-CATALOG) can be utilized (see also "Access via the file link name" and "Information on OPEN processing").

In contrast to the file attributes, the processing attributes can be modified each time the file is opened. They are also defined by operands of the ADD-FILE-LINK command and by the file definition in the program.

Null file

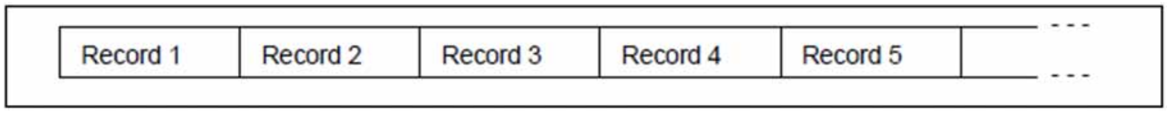
A null file is the result of opening a file for output (OPEN OUTPUT/OUTIN) and then immediately closing it again. If a read operation is initiated for a null file, the “end-of-file” condition is returned; if the null file is an ISAM file, the “key not found” condition may also occur.

16.4.1 Record formats

DMS supports three different record formats:

Record format F	with records of fixed length
Record format V	with records of variable length (default format)
Record format U	with records of undefined length

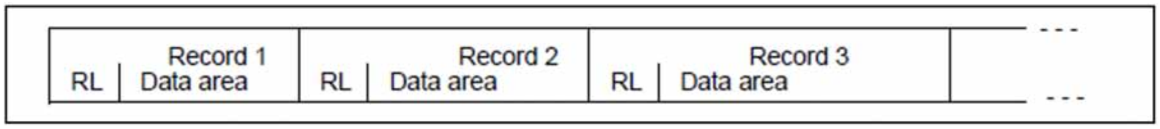
Format F (fixed-length records)



A file has the attribute “fixed-length records” if it is created with the operand RECORD-FORMAT=*FIXED in the ADD-FILE-LINK command or with the corresponding specification in a program. The record length must be defined via the RECORD-SIZE operand.

All records of the file have the same length, which is stored in the file's catalog entry. Since the system already knows the record length, the user does not have to worry about a record length or control field (see below) for input and output.

Format V (variable-length records)

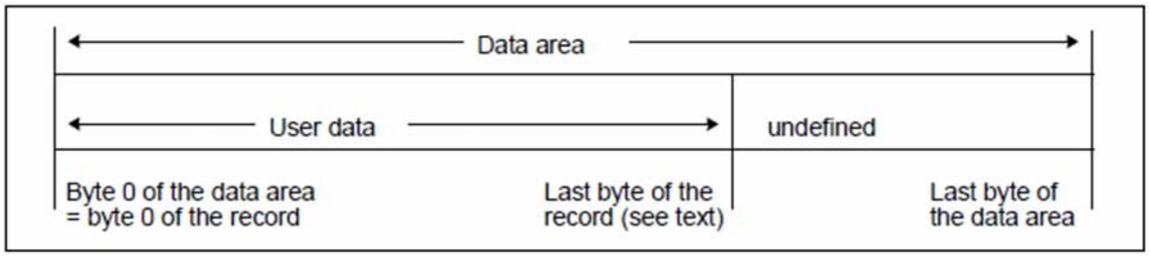


RL: record length = length of data area + 4

A file has the attribute “variable-length records” if it is created with the operand RECORD-FORMAT=*VARIABLE in the ADD-FILE-LINK command; RECORD-FORMAT=*VARIABLE is the default value if no record format is specified.

In variable-length records, there is always a 4-byte control field in front of the actual data, and the first two bytes of this field contain, in binary form, the length of the record, including the control field. The other two bytes of the field are reserved.

Format U (undefined-length records)


Data area
User data
undefined
Byte 0 of the data area = byte 0 of the record
Last byte of the record (see text)
Last byte of the data area

A file consists of “undefined-length records” if it is created with the operand RECORD-FORMAT=*UNDEFINED in the ADD-FILE-LINK command.

A file with record format U contains precisely one record per data block. Therefore, the current record length, which, when writing, the user has to always specify in a general-purpose register, is the same as the block length. Because BS2000 automatically stores the block length outside of the data area in the block-specific management information (see for example the section "Block formats for disk files"), it is not necessary to save the record length separately. Therefore, as with fixed record lengths, the user does not have to worry about record length or control fields: A record only contains the net data.

16.4.2 Block formats for disk files

Due to new developments in the architecture of disks, a number of additional disk formats have been introduced. Corresponding file formats have been created to support them. The format of the disk being used must be taken into account defining the block format for disk files.

16.4.2.1 Disk formats

In order to increase the net storage capacity and to improve the effective data transfer rate, new disk formats have been added. The following three criteria of a disk format are relevant for DMS:

- PAM key
  Disks with a PAM key (K disks) / disks without a PAM key (NK disks)
- Minimum allocation unit (min. AU) (minimum file size) 6 Kbytes / 8 Kbytes / 64 Kbytes
- Minimum transfer unit between disk and main memory 2 Kbytes / 4 Kbytes

The following disk formats must be differentiated from the viewpoint of DMS:

- K disk
- NK disk
- NK2 disk
- NK4 disk

i With respect to allocation, a Net-Storage volume behaves like an NK2 disk with the minimum allocation unit of 8 KB.

K disk

Each 2-Kbyte data block on the disk is accompanied by a field outside the block reserved for block-specific management data, the so-called PAM key (16 bytes).
The information stored in the PAM key is typically evaluated by the DMS access methods SAM and ISAM; it is not required by the UPAM access method (unless the user part of the PAM key is accessed by users themselves). The minimum allocation unit is 6 Kbytes. K disks can be used to store files of any format. K disks are supported by all previously released BS2000 versions.

NK disk

NK (Non-Key) disks are formatted without a PAM key. Block-specific management information must either be maintained in the data block itself or dropped entirely. The term “NK disk” is a generic term for NK2 and NK4 disks.

NK2 disk

The minimum transfer unit between the disk and main memory is 2 Kbytes. In other words 2 Kbytes of data or a multiple thereof is transferred with each read/write operation. For NK2 disks, a further distinction is made on the basis of the minimum allocation unit.
There are disks with a minimum AU of 6 Kbytes, 8 Kbytes, and 64 Kbytes, respectively. The disks with a minimum AU of 8 or 64 Kbytes can be used for both NK2 and NK4 files.

NK4 disk

The minimum transfer unit between the disk and main memory is 4 Kbytes. In other words 4 Kbytes of data or a multiple thereof is transferred with each read/write operation. The minimum allocation unit is 8 Kbytes or 64 Kbytes. Only NK4 files can reside on NK4 disks.

The table below shows the various disk designations.

Designation of disk	**PAM key** **K = Key** **NK= Non-Key**	**Minimum allocation unit (min. AU)**	**Minimum transfer unit (min. TU)**
K disk	K	6 Kbytes	2 Kbytes
NK2 disk (6K) 1) = NK2 disk with min. AU of 6 Kbytes	NK	6 Kbytes	2 Kbytes
NK2 disk (8K) = NK2 disk with min. AU of 8 Kbytes	NK	8 Kbytes	2 Kbytes
NK2 disk (64K) = NK2 disk with min. AU of 64 Kbytes	NK	64 Kbytes	2 Kbytes
NK4 disk (8K) = NK4 disk with min. AU of 8 Kbytes	NK	8 Kbytes	4 Kbytes
NK4 disk (64K) = NK4 disk with min. AU of 64 Kbytes	NK	64 Kbytes	4 Kbytes

Table 38: Overview of various designations/formats for disks

1) The minimum allocation unit is only indicated in the designation of the disk if it isrelevant for the described case. Otherwise, the terms NK2 disk and NK4 disk are used.

Notes

- The disk format is defined by systems support.
- The disk format in an SF pubset or volume set is homogeneous. For each disk format, there is also a corresponding pubset format.
- A user can obtain information on the formatting of his or her pubset with the SHOW-MASTER-CATALOG-ENTRY command or the STAMCE macro (on the program level).
- Only the K and NK2 formats with a minimum allocation unit of 6 Kbytes are supported for private disks.

16.4.2.2 File formats

The following terms must be clearly differentiated in order to understand the various block and file formats described below:

- PAM block
- logical block (data block)
- transfer unit

PAM block: A PAM block comprises 2048 bytes plus a PAM key of 16 bytes. Following the introduction of the NK disk format, it has now become customary to treat the size of a PAM block as 2048 bytes without the PAM key. A PAM block is also referred to as a PAM page or a 2-Kbyte data block.

Logical block: A logical block is a contiguous area containing data or records. Its size is specified by the BUFFER-LENGTH operand of the ADD-FILE-LINK command or the BLKSIZE operand of the FILE macro. This specification can be given in the form
*STD(SIZE=n) or (STD,n), or as a specific number of bytes.
(STD,n) means that a logical block is made up of n units of 2048 bytes each. (n must be an even number for files to be allocated on NK4 disks.)
In the case of tape files and cartridges, the length can also be explicitly specified in bytes. However, this specification only defines the maximum length of the logical block. The actual length, which must be less than or equal to n bytes, is determined by the data itself.

Transfer unit: A transfer unit indicates the amount of data that can be transported as a unit between main memory and a disk with one physical I/O operation.

The transfer unit for disks is equal to the length of a logical block. For NK2 disks, the transfer unit is a multiple of 2 Kbytes (which means that the smallest transfer unit is 2 Kbytes); for NK4 disks, a multiple of 4 Kbytes (so the smallest transfer unit is 4 Kbytes).

Two different cases must be distinguished for tapes:

- The block length of a file is specified as BLKSIZE=(STD,n) in the macro or as BUFFER-LENGTH=*STD(SIZE=n) in the command. This defines a logical block length of n * 2 Kbytes for tape files. The transfer unit in this case is 2 Kbytes. This means that n * 2048 bytes are transported as a unit. In other words, n I/Os of 2 Kbytes each are executed.
- If the block length of a file is specified as BLKSIZE=n in the macro or as BUFFER-LENGTH=n in the command. This specification defines a logical block length of n bytes for tape files. So n bytes are transferred in each I/O. In this case, the logical block length and the transfer unit are identical for tape files as well.

DMS offers a number of different file/block formats to support the various disk formats. The two basic block formats are:

- block format with PAM keys, referred to below as the K (key) file format
- block formats without PAM keys, also known as NK (non-key) file formats.

K file format

The block-specific management information (block control information) is held in the PAM key (16 bytes). 8 bytes of the PAM key is used by the system, the other 8 bytes are used by the higher-level access methods SAM and ISAM. The user can only access this “user” part of the PAM key with UPAM. The user should, however, refrain from using this part of the PAM key so as to simplify subsequent conversion of his applications to NK format.

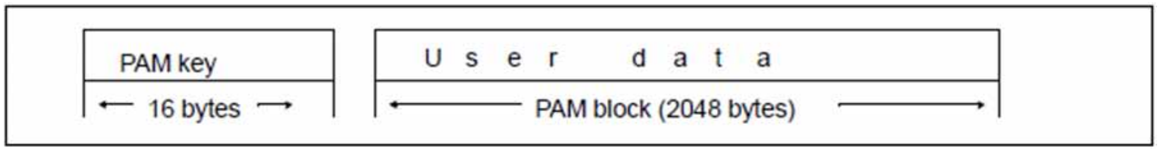
This block format is supported by all access methods except DIV and FASTPAM. A file can be created in K format if the volume allows the PAM key to be recorded (K pubset or K private disk).

See the sections on the individual access methods for more details.

*BLOCK-CONTROL-INFO=*PAMKEY*

A K format is defined by means of the operand BLOCK-CONTROL-INFO=*PAMKEY of the ADD-FILE-LINK command or the operand BLKCTRL=PAMKEY of the FILE macro.

The structure of a PAM block with the associated PAM key is shown below:



NK file formats

A distinction is made between three different NK file formats in which block-specific management information is held within each data block (BLOCK-CONTROL-INFO= *WITHIN-DATA-BLOCK/*WITHIN-DATA-2K-BLOCK/*WITHIN-DATA-4K-BLOCK in the ADD-FILE-LINK command).
This area is not available for user data.
Besides the above formats, the user may also select an NK file format in which no blockspecific management information is stored (BLOCK-CONTROL-INFO=*NO).

When the UPAM and SAM access methods are used, the block specific management information is always 16 bytes long (12 bytes block control field and 4 bytes data length field).

In the case of an NK2-ISAM file (BLOCK-CONTROL-INFO=*WITHIN-DATA-2K-BLOCK), the block control information is stored in the first 16 bytes of each 2-Kbyte block.
For an NK4-ISAM file (BLOCK-CONTROL-INFO=*WITHIN-DATA-4K-BLOCK), it is stored in the first 16 bytes of each 4-Kbyte block.

The following must be observed when converting a K file to an NK file:
if the maximum record length (16 PAM blocks = 32 Kbytes) is exhausted, incompatibilities may occur, since there is no space available for the block control information.

Only NK4 files may be stored on an NK4 disk.

See the sections on the individual access methods for more details.

*BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK*

An NK format is defined by the operand BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK of the ADD-FILE-LINK command or the operand BLKCTRL=DATA of the FILE macro. Either an NK2 or an NK4 file is created in the process:

When a file is created, its format is determined as follows:

PAM and SAM files:

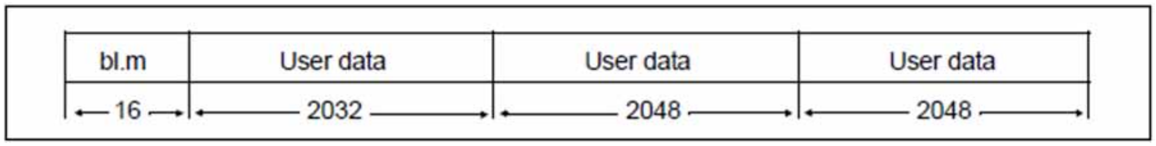
If the blocking factor n is an odd number, an NK2 file is created.
If the blocking factor n is an even number, an NK4 file is created.

ISAM file:

If the block format of the ISAM file is not explicitly specified (see below), the format of the disk on which the ISAM file is created determines whether an NK2-ISAM or NK4-ISAM file is created.

A file that has already been opened can be opened without regard to the block format.

The following diagram illustrates the logical data block of an NK2-SAM file (with length 6 Kbytes and blocking factor n=3; bl.m stands for block-specific management information):

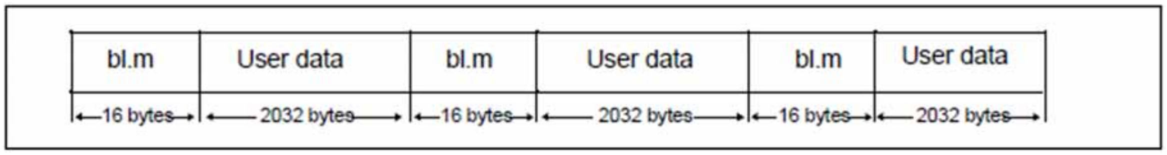


*BLOCK-CONTROL-INFO=*WITHIN-DATA-2K-BLOCK (for NK-ISAM files)*

An NK2 file can be created explicitly by using the operand BLOCK-CONTROL-INFO= *WITHIN-DATA-2K-BLOCK in the ADD-FILE-LINK command or the operand BLKCTRL=DATA2K in the FILE macro (this specification can also be made for SAM and PAM; it then functions like *WITHIN-DATA-BLOCK).

The block-specific management information is held in the first 16 bytes of each 2-Kbyte data block.

The following diagram illustrates the logical data block of an NK2-ISAM file (with length 6 Kbytes and blocking factor n=3; bl.m stands for block-specific management information):



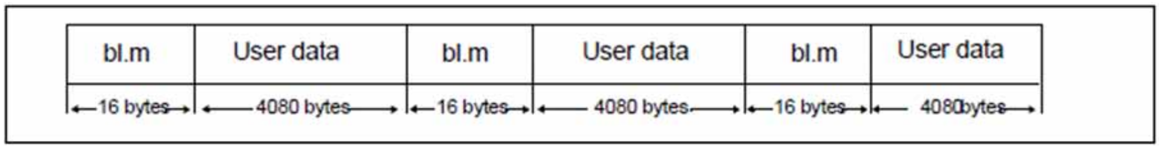
*BLOCK-CONTROL-INFO=*WITHIN-DATA-4K-BLOCK (for NK-ISAM files)*

An NK4 file can be created explicitly by using the operand BLOCK-CONTROL-INFO= *WITHIN-DATA-4K-BLOCK in the ADD-FILE-LINK command or the operand BLKCTRL=DATA4K in the FILE macro (this specification can also be made for SAM and PAM; it then functions like *WITHIN-DATA-BLOCK).

The block-specific management information is held in the first 16 bytes of each 4-Kbyte data block.

An NK4-ISAM file can be created on any disk.

The following diagram illustrates the logical data block of an NK4-ISAM file (with length 12 Kbytes and blocking factor n=6; bl.m stands for block-specific management information):



*BLOCK-CONTROL-INFO=*NO*

An NK format can also be defined by using the operand BLOCK-CONTROL-INFO=*NO in the ADD-FILE-LINK command or the operand BLKCTRL=NO in the FILE macro.

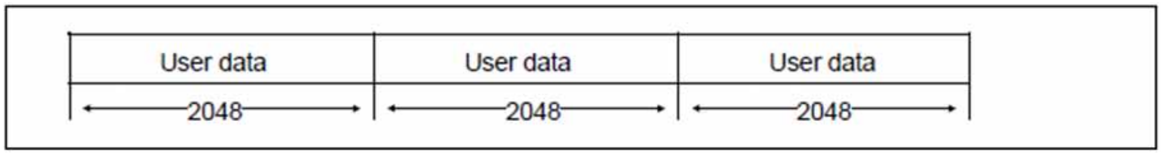
In this case, the system does not store any block-specific management information in the PAM key or within the data block.

This format is only available for UPAM disk files, UPAM tape files and SAM tape files.

In the case of SAM disk files and ISAM disk files, the operands BLOCK-CONTROL-INFO=*NO and BLKCTRL=NO are internally converted to BLOCK-CONTROL-INFO= *WITHIN-DATA-BLOCK and BLKCTRL=DATA, respectively.

See the sections on the individual access methods for more details.

The following diagram illustrates the logical data block of an NK2-PAM file without blockspecific management information (with length 6 Kbytes and blocking factor n=3):



DMS offers a number of different file/block formats to support the various disk formats. The interactions and interdependencies among the following file attributes should be taken into account when specifying the file format:

- file structure
- block structure
- logical block length (blocking factor)

The following file formats and other file attributes may be combined:

File format	File structure (ACCESS-METHOD)	Block structure (BLOCK-CONTROL-INFO)	Logical block length (BUFFER-LENGTH)	Blocking factor (n)	Disk format
K file	PAM	PAMKEY			
	SAM				
	ISAM				
NK2 file	PAM	NO or DATA	STD	odd	K disk
	SAM	DATA	STD	odd	
	ISAM	DATA2K			
		DATA			NK2 disk (6K/8K /64K)
NK4 file	PAM	NO or DATA	STD	odd	K disk
	SAM	DATA	STD	odd	NK2 disk (6K/8K /6KB)
	ISAM	DATA4K	STD	odd	NK4 disk

Table 39: Overview of the various file formats

The minimum allocation unit has no effect on the file format.

Only files with an even blocking factor can be stored on NK4 disks.

The K file format is supported by all previously released BS2000 versions.

16.4.3 Block formats for tape files

A physical block is the unit of data transfer from or to an I/O device. The data between two inter-block gaps on a tape forms one physical block. Blocks of this type never extend beyond a tape boundary. There are two block formats for tape files: standard block formats and nonstandard block formats. Tape files are always processed by the device driver PTAM.

Standard block format

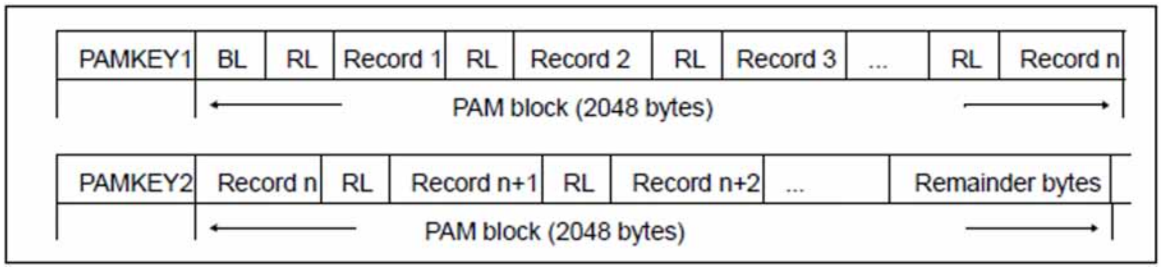
*BLOCK-CONTROL-INFO=*PAMKEY, BUFFER-LENGTH=*STD(SIZE=n)*

A standard block corresponds to n PAM pages. The PAM key is placed in front of the PAM page by the data chaining facility; it is not stored separately on tape. PTAM is the only driver available for tape files.

The specified BUFFER-LENGTH determines how many PAM pages make up a standard block. When the SAM access method is used, a standard block is not always completely filled by logical blocks. The area between the end of the last logical record and the end of the block will have undefined contents (see “remainder bytes” in the example below). Chained I/O is not possible for tape files.

Example

```
BUFFER-LENGTH=*STD(SIZE=2), RECORD-FORMAT=*VARIABLE
```



Nonstandard block formats

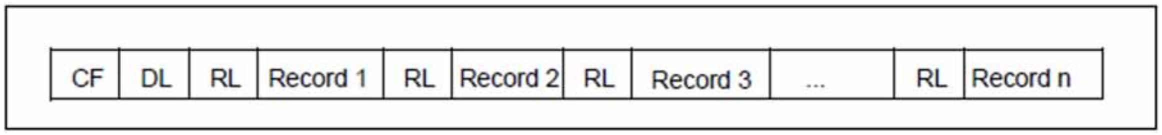
A nonstandard block is a data block on magnetic tape that is not preceded by a PAM key. Nonstandard blocks must never be longer than specified in the BUFFER-LENGTH operand in the ADD-FILE-LINK command (maximum length: 32768 bytes). Nonstandard blocks are generated whenever BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK or *NO is specified, even with BUFFER-LENGTH=*STD(SIZE=n). In the case of UPAM processing, the BUFFER-LENGTH value must be a multiple of 2048.

*Data block with management information (BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK)*

This block is not preceded by a PAM key. Instead, SAM and UPAM place a 12-byte block control field (CF) at the start of each logical block, to which SAM adds a 4-byte data length field (DL). These 12 (CF for UPAM) or 16 (CF plus DL for SAM) bytes are handled as a buffer offset.

Example

```
BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK,BUFFER-LENGTH=*STD(SIZE=2),
RECORD-FORMAT=*VARIABLE, ACCESS-METHOD=*SAM
```

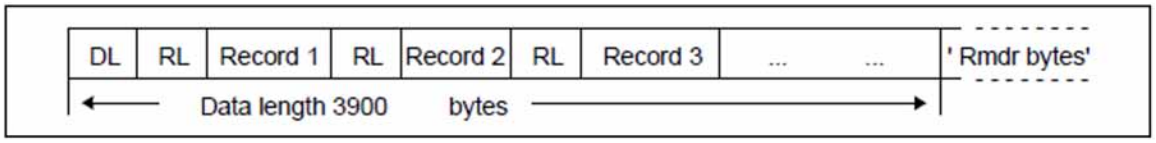


*Data block without management information (BLOCK-CONTROL-INFO=*NO)*

This block contains no block-specific management information. This format is only available for UPAM and SAM files.

Example

```
BLOCK-CONTROL-INFO=*NO,BUFFER-LENGTH=4096,RECORD-FORMAT=*VARIABLE,
ACCESS-METHOD=*SAM, Datenlänge 3900 Byte
```



Only the data is written to tape (in this example, 3900 bytes).

The data blocks within a file may have different lengths. However, no block may exceed the maximum block length specified in the BUFFER-LENGTH operand of the ADD-FILE-LINK command.

The following point applies to SAM files:

The data length field (DL) is not part of the record as defined in the DIN standard. For format V (RECORD-FORMAT=*VARIABLE) records, there is thus a buffer offset (BLOCK-
OFFSET) of 4 bytes. If the user specifies the operand BLOCK-OFFSET=0 in the ADD-FILE-LINK command, no buffer offset is executed, i.e. the block length field is ignored.

The extension of blocks by means of filler characters is supported by the SAM access method and by the label handling routines when reading (e.g. when reading non-BS2000 tapes).

Structure of the block and record length fields (for SAM files only):

in EBCDIC code:	2-byte binary number (the block or record length), followed by two blanks (2X'40')
in ISO code:	a four-digit decimal number (corrsponds to format D)

As the decimal number in format D cannot exceed 9999, the maximum block length is 9999 Byte.

This block format is the block format for data exchange in accordance with DIN 66029 and will remain upwards and downwards compatible.

Example: SAM file with variable-length records and nonstandard blocks

The block (block length = 198 bytes) contains 3 records with lengths of 76, 84 and 34 bytes. The block and record length fields (BL and RL) contain the following:

	BL	SL-1	SL-2	SL-3
EBCDIC code (format V)	00C64040	004C4040	00544040	00224040
ISO code (format D)	F0F1F9F8	F0F0F7F6	F0F0F8F4	F0F0F3F4

Access methods and block format for tape files

The block format (standard or nonstandard blocks) is determined by the BLOCK-CONTROL-INFO, BUFFER-LENGTH or LABEL-TYPE operand (block length or tapestandard type) and by the access method. The following table shows which values may be specified for BLOCK-CONTROL-INFO, depending on the access method:

BLOCK-CONTROL-INFO	UPAM	SAM	BTAM
*PAMKEY	+	+	
*WITHIN-DATA-BLOCK	+	+	
*NO	+	+	+

Table 40: Tape files: block format and access methods

As shown in the table above, BTAM can only process files with nonstandard blocks (BLOCK-CONTROL-INFO=*NO). When working with SAM or UPAM, files with any block format (standard and nonstandard blocks) (BLOCK-CONTROL-INFO=*PAMKEY/ *WITHIN-DATA-BLOCK/*NO) can be processed.

A file cannot contain standard and nonstandard blocks at the same time.

16.4.4 Relationships between record, block and PAM page

From the point of view of DMS, a record is an indivisible data element. Both with regard to locating data in a file and to transferring data from and to the calling program.

This means that each data block must be at least as long as the longest record in the file. For disk files, the block format must be compatible with the block format of PPAM.

A data block is made up of one or more PAM pages. A tape file may consist of standard or nonstandard blocks. The maximum data block length for a disk or tape file is 32768 bytes.

Examples of record/block/PAM page relationships for SAM

1. File attributes:

`RECORD-FORMAT=*FIXED`	fixed-length records
`RECORD-SIZE=100`	record length 100 Byte
`BUFFER-LENGTH=*STD(SIZE=2)`	block length = 2 PAM pages = 2*2048 Byte = 4096 Byte

   Each data block contains 40 records, and 96 bytes are left free in each block. Each data block consists of 2 PAM pages. DMS stores records within a data block without regard for the boundaries between PAM pages. In this example, the first 48 bytes of record 21 are in the first PAM page, and the remaining 52 bytes are in the second PAM page.

2. File attributes:

`RECORD-FORMAT=*FIXED`	fixed-length records
`RECORD-SIZE=1500`	record length 1500 Byte
`BUFFER-LENGTH=*STD`	block length = 1 PAM page

   This is a very inefficient combination, because 548 bytes are “wasted” in each block. A better block length would be 6144 bytes (3 PAM pages, BUFFER-LENGTH= (*STD(SIZE=3)), since the data block would then contain 4 records and 6000 of the 6144 bytes would be used.

3. File attributes:

`RECORD-FORMAT=*FIXED`	fixed-length records
`RECORD-SIZE=8192`	record length 8192Byte
`BUFFER-LENGTH=*STD(SIZE=8)`	block length = 8 PAM pages

   Each data block contains 2 records and each record occupies 4 PAM pages.

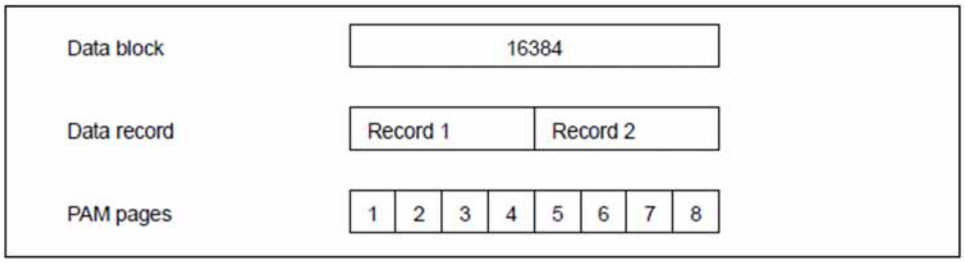


Figure 23: Data block/PAM page/record

17 BTAM - Basic Tape Access Method

BTAM provides the user with an effective and versatile means of storing and retrieving the blocks of sequentially organized tape files. It can be used to process magnetic tapes, tape cartridges and tapes created on non-BS2000 systems.

BTAM can also process tape files which have nonstandard labels or no labels or which were created in accordance with DIN 66029, interchange level 1 (corresponding to LABEL=(STD,1) in the FILE/FCB macro or LABEL-TYPE=*STD(DIN-REVISION-NUMBER=1) in the ADD-FILE-LINK command).

By means of the BTAM macro, the user can position tapes to any desired block or file section. An important feature of BTAM is that a program may read and/or write data blocks as desired and change the processing direction, in both cases without having to close the file beforehand.

Codes supported by BTAM:

- EBCDIC
- ISO 7-bit code
- OWN code

As for SAM processing, the label specification can influence the code and vice versa.

BTAM functions

- Chained input/output
  (CHAINIO operand of the BTAM macro)

  Chained I/O economizes on CPU performance and saves time during data input and output.
- MAV mode
  (multi-job management, BTAMRQS/REQNO operand of the BTAM macro)

  Two or more tape I/O operations can be processed (asynchronously) in parallel. As MAV mode is extremely time-critical, it can be used to maximum advantage only with chained I/O: the greater the chaining factor, the better the device utilization and time behavior. Time behavior is enhanced by the fact that I/O operations can be performed continuously without the tape having to be repositioned for each operation. In the case of corresponding BTAM I/O operations, BTAM does not wait for the operation to be concluded, but returns control to the caller immediately. The caller can obtain the results of the I/O operation later with a WAIT/CHECK call. The BTAM I/O operation to which a subsequently issued WAIT/CHECK call belongs must be indicated in each asynchronous BTAM job by means of a request number (REQNO operand of the BTAM macro). The increase in performance derives from the fact that BTAM can execute several asynchronous operations in parallel. In other words, the next I/O operation does not have to wait until the previous operation has been completed. The highest request number which can be assigned (and thus the maximum number of asynchronous I/O operations which can be executed in parallel for a job) is defined by means of the BTAMRQS operand.
- Buffered input and output
  (TAPEWR operand of the BTAM macro)

  For files on magnetic tape cartridges.

BTAM macros

The following macros are available for processing tape files with the BTAM access method:

- all service macros (OPEN, CLOSE, etc.)

- the BTAM macro with the following functions:

Function	Meaning
CHK	Check the processing status of an I/O operation.
ERG	Generate an interblock gap.
POS	Position the tape.
RBID	Determine the current tape position.
RD/RDWT	Read data into main memory and wait for I/O termination.
RDBF	Read data from the save area of the tape cartridge buffer.
REV/REVWT	Read data in the reverse direction and optionally wait for I/O termination.
RT/RTL	Read with data transfer; with/without message if the length is less than anticipated.
RNT/RNTL	Read without data transfer; with/without message if the length is less than anticipated.
SYNC	Synchronize and determine the tape position.
WRT/WRTWT	Read data from main memory and wait for I/O termination.
WT	Wait for completion of the I/O operation.

Table 41: Functions of the BTAM macro

- The following control codes are provided for the positioning of tapes and the writing of tape marks (in the BTAM macro):

  BSF, BSR, FSF, FSR, REW, RUN, WTM.

17.1 BTAM record and block formats

BTAM is a block-oriented access method, i.e. BTAM does not recognize any record structures within blocks. BTAM therefore interprets the input RECFORM=F in the FILE macro or RECORD-FORMAT=*FIXED in the ADD-FILE-LINK command to mean that all the blocks in this file have a fixed block size.

BTAM supports the following record formats:

- F for fixed-length records
- U for records of undefined length
- V for variable-length records (formats V and U are treated identically)

BTAM only processes files with the format BLKCTRL=NO or BLOCK-CONTROL-INFO=*NO.

If RECFORM=U or RECORD-FORMAT=*UNDEFINED is specified, DMS takes the block size

- from the register defined with RECSIZE=reg in the FILE/FCB macro
- from the specification in the action macro (LEN operand)
- from the specification in the RECORD-SIZE operand of the ADD-FILE-LINK command

If RECFORM=F or RECORD-FORMAT=*FIXED is specified, the block size defined by BLKSIZE or RECORD-SIZE or specified in the LEN operand of the BTAM macro is used.

The block size may be: 18 bytes <= block size <= 32768 bytes.

Files created with SAM can be processed with BTAM. Each user, however, is then responsible for unblocking the records.

17.2 Opening a BTAM file

Opening a BTAM file depends on the OPEN mode. The table below provides an overview of the various modes and the resulting possibilities for processing a BTAM file.

OPEN mode	Processing
INOUT	Retrieve records from an existing file and add new records; no header labels are written, since it is assumed that the file already exists.
INPUT	Retrieve files from an existing file in the forward direction.
OUTIN	Create a new file and/or retrieve records; labels are written since a new file is being created.
OUTPUT	Create a new file.
REVERSE	As for INPUT, but the tape is positioned to EOF at OPEN time. Files which extend over several volumes can only be processed individually (with the help of the VSEQ operand in the FILE macro or the VOL-SEQUENCE-NUMBER operand in the ADD-FILE-LINK command).
SINOUT	As for INOUT, but the tape is not positioned; this is not permitted if the tape is positioned at the beginning of the tape.

Table 42: OPEN modes for BTAM files

The OPEN modes INPUT and REVERSE differ, at OPEN time, only in how the tape is positioned. An OPEN REVERSE followed by a RD or RDWT operation will not result in a reverse read.

17.3 Files on magnetic tape cartridge (MTC)

The tape operations during OPEN processing are not buffered. The same applies to EOV and CLOSE processing.

Once OPEN processing has been completed, the output mode is set to buffered processing unless TAPEWR=IMMEDIATE was specified in the FILE macro or in the FCB, or the operand TAPE-WRITE=*IMMEDIATE was specified in the ADD-FILE-LINK command.

As soon as EOV/EOF processing is initiated for output files, DMS executes a synchronization operation in order to ensure that all user data has been written to tape. Once the buffer is empty, the end labels are written in unbuffered mode.

If, during automatic synchronization at the start of EOV/EOF or CLOSE processing, an unrecoverable write error is detected for one of the blocks in the tape cartridge buffer, the EXLST exit CLOSER is activated, but the user can no longer access the data in the buffer. The user can avoid this problem by explicitly initiating synchronization with the SYNC command. In this case, unrecoverable write errors are signalled via the EXLST exit ERRADR. The user can rectify the problem since he/she can still access the data in the tape cartridge buffer.

18 DIV - Data In Virtual

Data In Virtual (DIV) is an access method that differs from the traditional access methods such as ISAM, SAM and UPAM because it does not require a file to be structured into records and blocks and works without I/O buffers, and operations such as GET or PUT. DIV is an object-oriented access method that is particularly suitable for processing unstructured data (Binary Large OBjects (BLOBs)).

The MAP function of DIV maps the contents of a file to an area in virtual address space (program space, data space). This area in virtual address space then forms a window in which pages of the file appear when they are accessed.

Data can thus be accessed using CPU instructions. Any data that has been modified in the window can be written back to the file with the DIV function SAVE.

DIV can be used to process NK-PAM files which contain no management information (BLOCK-CONTR=NO). Cross-system file access (RFA) is not supported.

Besides enabling object-oriented access, DIV reduces the number of accesses to disk and thus improves performance dramatically.

DIV is suitable for applications that do not require data to be structured into records and blocks. High performance gains are achieved when data that has already been read into the window by an earlier access operation is accessed repeatedly.

The FCT=*MAP and FCT=*UNMAP functions of the DIV macro are used to open and close windows for a program space. SPID=0 must always be specified for the SPID operand (data space ID).

DIV functions

DIV functions are used by calling the DIV macro. The following are the basic EAM functions:

Function	Meaning
OPEN	Open a file
MAP	Define a window (i.e. a work area) in the virtual address space
SAVE	write modified pages from the window back to the file on disk
RESET	undo changes in the window
UNMAP	release windows in virtual address space
CLOSE	close a file, releasing any existing windows with default values, if applicable

Table 43: Functions of the DIV macro

The Adapter Window Services can be used in some high-level programming languages to invoke DIV functions from programs via a CALL interface. These programming languages are:

- COBOL (Version 2.0 onward)
- FORTRAN (Version 2.2 onward)
- PL/1 (Version 4.1 onward)
- C (Version 2.0 onward)

ILCS linkage is required in order to call DIV functions via a CALL interface. The Adapter Window Services are supplied as a runtime library (OML) in a file called SYSLIB.DWS.120 under the systems support ID TSOS. When DIV is invoked from the Window Services, not all DIV functions are available or are fully available.

For detailed information on the CALL interface, see the appendix in the “DMS Macros” manual [1 (Related publications)].

18.1 Functionality of DIV

DIV actions are internally controlled via a page status indicator. The status of a page can have one of the following values: FRESHLY_OBTAINED (F), ACCESSED (A) or MODIFIED (M).

The current status of a page in a DIV window determines which actions are performed on it after it is accessed.

Control via page status

If the definition of a window includes the specification that the pages of the file should appear in that window (function MAP with the operand DISPOS=*OBJECT), all pages of that window will have the status FRESHLY_OBTAINED. This is also the status of a page when memory has been allocated using the REQM macro (initial state).

When such a page is accessed (by any CPU instruction), a page fault interrupt is generated, and DIV receives control from the memory management system in order to read the page from the file. If the page in question is located after the last logical page of the file, it is initialized with X'00'. The page will then no longer be in the initial state but in the state ACCESSED. It will be treated like any other page in virtual address space when subsequently accessed.

If the page is modified by the program, its status is set to MODIFIED. The contents of a page having this status are different from the corresponding page on file. The DIV function SAVE can be used to write modified pages to the file. Following the SAVE operation, the pages in question will no longer have the status MODIFIED and will not be written to file when SAVE is called again, unless they were accessed and modified again by the program.

The selection of window pages to be written to the file on disk also depends on which value was specified for the DISPOS operand (in the MAP function) when creating the window, and on the position of the page with respect to the logical end-of-file.

The DIV function RESET can be used to reset the status of a modified page to the initial state. In a window with the attribute DISPOS=*OBJECT, the page is read in from the file again when it is subsequently accessed (or initialized with X'00' if it lies beyond the logical end-of-file). For a window defined with DISPOS=*OBJECT, RESET therefore causes the pages from the file to reappear in the window.

The RESET function with the operand RELEASE=*YES causes not only unmodified pages, but all pages of a predefined region to be reverted to the initial state. As a result, all such pages are read in from the file when they are accessed.

MAP does not alter the status of pages in a window defined with DISPOS=*UNCHNG. If a page has never been accessed, it retains its initial state; otherwise, as far as DIV is concerned, its status is MODIFIED.

When DIV obtains control following access to a page in a window specified with DISPOS=*UNCHNG, the page is initialized with X'00' and is set to the status MODIFIED.

If a page has previously been written to file with SAVE and then set to the initial state by RESET, it will be read from the file even for a DISPOS=*UNCHNG window and will not be initialized with X'00'.

Note

Allocating address space: in order to use DIV, the user must request virtual address space. Address space in the program space is allocated using the REQM macro, address space in the data space with the macro DSPSRV. When pages are allocated in main memory, they are first initialized with X'00'.

18.2 File extension and file truncation

The SAVE function can modify the logical file-end.

18.2.1 Extending a file logically

Rules for logical file extension:

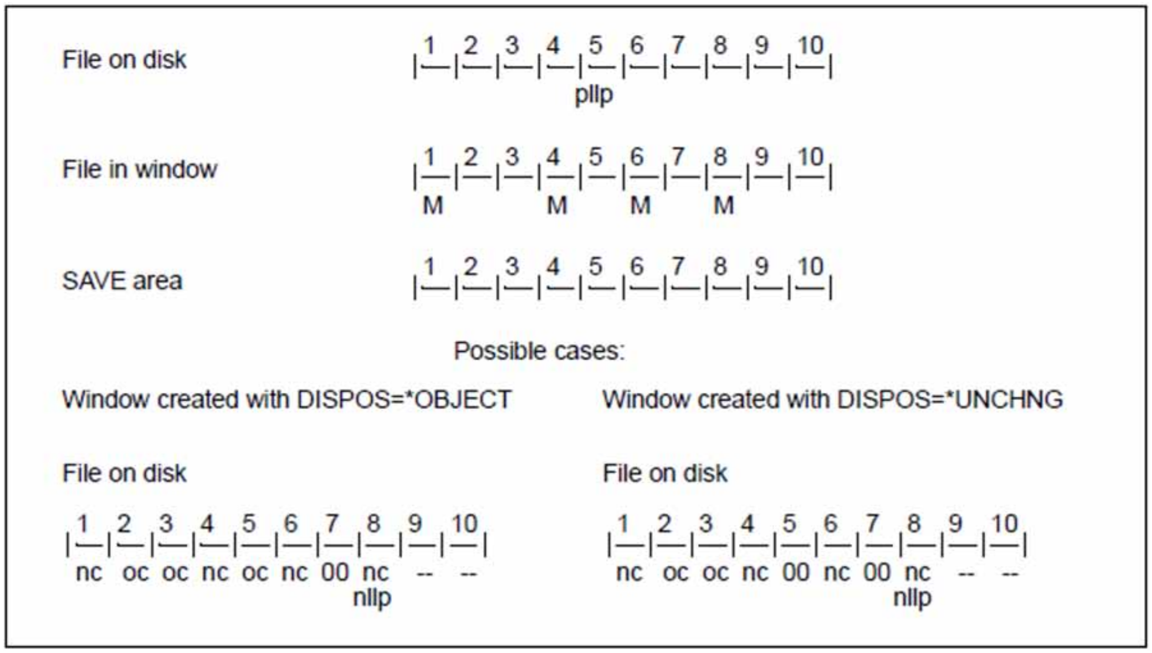
- A file is logically extended up to the last “MODIFIED” page which lies beyond the previous logical last page and is contained in both a save area and a window.
- Pages that lie in the new file area and form part of the extended data of the file are written to the file, provided they are contained in both the save area and a window.

  Unmodified pages in a new file region are filled with X'00'.

  Pages that are not contained in the save area as well as a window are not written to the file.
- The DISPOS attribute of the window(s) has no effect on the logical extension of a file.

The logical last page of a file is that page which your own task considers to be the logical last page of the file. If SHARUPD=*YES is specified, a foreign task may have modified the logical last page unnoticed by your own task. This can be used to cancel a file extension effected by a task running in parallel.

Example 1: extending a file logically



Key

1...10	page numbers (file/window)
M	modified
nc	new contents
M	new logical last page
oc	old contents
pllp	previous logical last page

--	undefined contents

Explanation

The physical length of the file is 10 pages (1 page = 4 Kbytes). The logical end-of-file is page 5. The SAVE area covers pages 1-10. The last modified window page in the SAVE area is page 8.

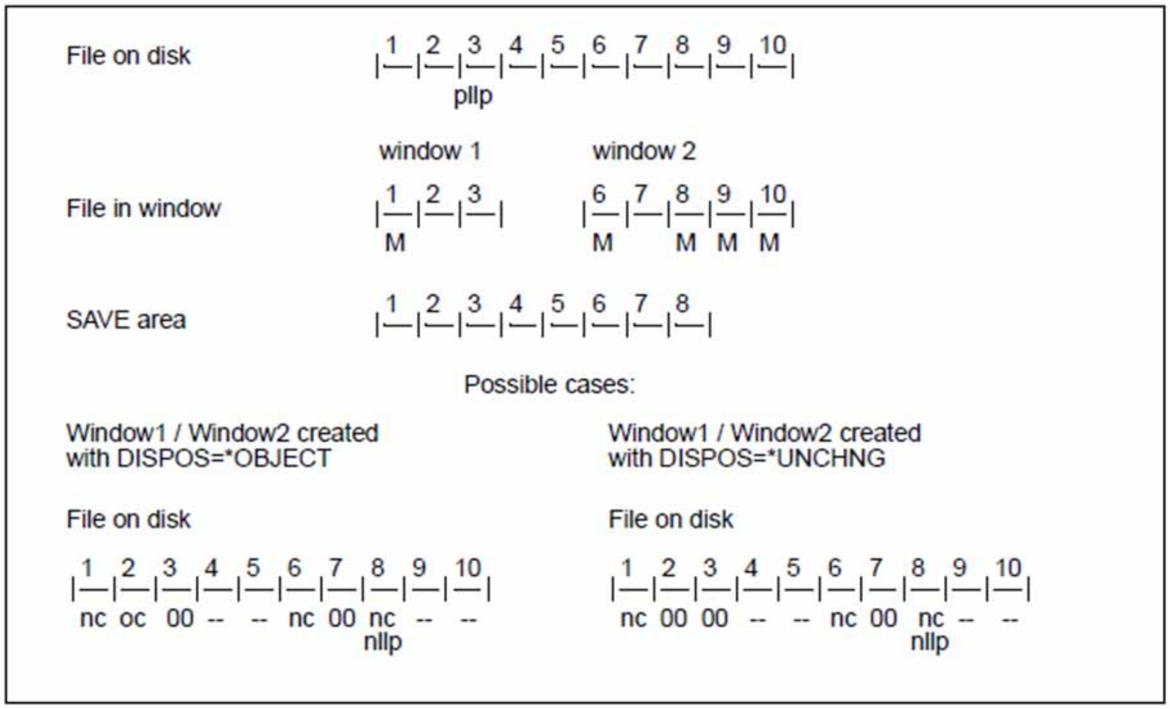
Irrespective of the DISPOS attribute of the window, all modified pages between the previous and the new last logical pages are written to the file on disk, and all unmodified pages are initialized with X'00' in the disk file.

The following cases must be differentiated for the remaining region (from page 1 to the previous logical last page):

- If DISPOS=*OBJECT: Only the modified pages are written to the file.
- If DISPOS=*UNCHNG: Modified pages are written to the disk file as well.

Unmodified pages that have not yet been written to file (with SAVE), are initialized in the file with X'00' (page 5). They are then set to the status SAVED. The above example assumes that pages 2 and 3 were written to the file (with SAVE) earlier. No action is taken for unmodified pages that have already been written to a file using SAVE (pages 2 and 3).

Example 2: extending a file logically



File on disk
pllp
window 1
window 2
File in window
M
SAVE area
Possible cases:
Window1 / Window2 created with DISPOS=*OBJECT
Window1 / Window2 created with DISPOS=*UNCHNG
File on disk
nc oc 00 -- -- nc 00 nc -- --
nllp
nc 00 00 -- -- nc 00 nc -- --

Key

1...10	page numbers (file/window)
M	modified
nc	new contents
M	new logical last page

oc	old contents
pllp	previous logical last page
--	undefined contents

Explanation

The file is 10 pages long. The logical end-of-file is page 2. Pages 1-3 have been assigned to window 1, pages 6-10 to window 2. An 8-page SAVE area is defined. It is covering window 1 and the first three pages of window 2. The last modified window page of the SAVE area is page 8. Page 8 lies below the previous logical end-of-file and therefore becomes the new logical EOF (new logical last page).

Irrespective of the DISPOS attribute of the window, all modified pages between the previous and the new last logical pages are written to the file on disk (pages 6 and 8), and all unmodified window pages (pages 3 and 7) are initialized with X'00' in the disk file.

The following cases must be differentiated for the remaining region (from page 1 to the previous logical last page):

- If DISPOS=*OBJECT: Only the modified pages are written to the file (page 1).
- If DISPOS=*UNCHNG: Modified pages are written to the disk file as well. Unmodified pages that have not yet been written to file (with SAVE), are initialized in the file with X'00' (page 2). They are then set to the status SAVED. (No action is taken for unmodified pages that have already been written to the file using SAVE, see also previous example).

Pages 4 and 5 remain unchanged in the disk file, as no window exists for them. Since they are located after the previous logical end-of-file, their contents are undefined.

Modifications to pages 9 and 10 in window 2 are not be written back to disk, since these pages are not in the SAVE area.

18.2.2 Logical truncation of files / erasing file contents

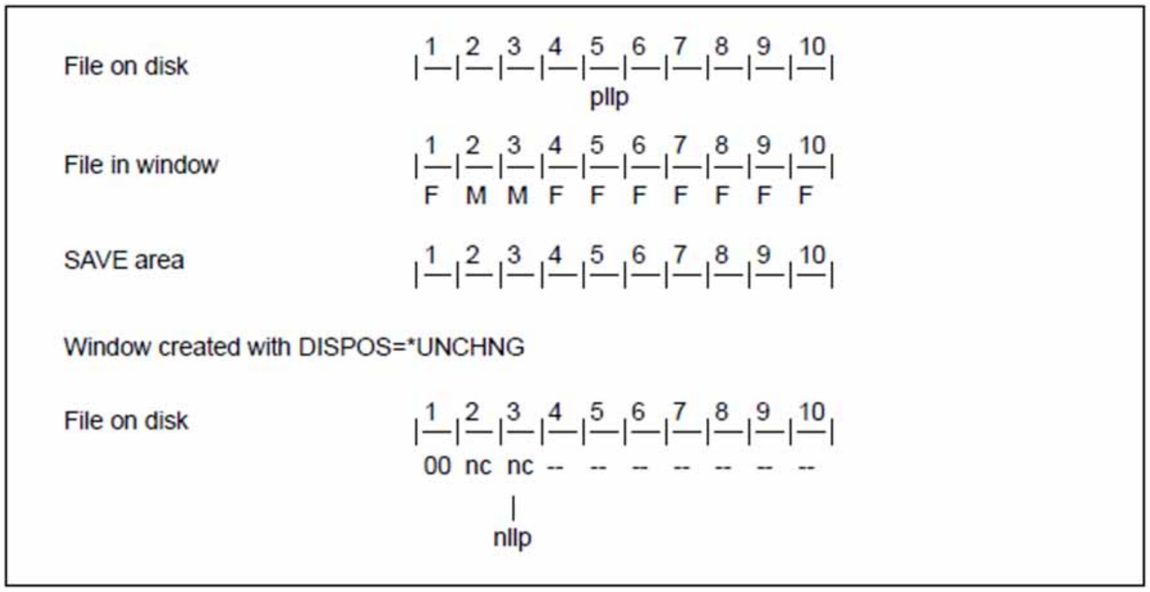
A file is logically truncated if all of the following conditions apply:

1. No condition for file extension applies.
2. The last logical page is located in a window with the attribute DISPOS=*UNCHNG and in the SAVE area.
3. At least the last page is in the initial state and has not yet been written to the file with SAVE.

The logical truncation of a file ends if any of the following results has been produced:

- The end of the SAVE area has been reached.
- At least one page is not in a window with the attribute DISPOS=*UNCHNG (i.e. if there is no window or if the window attribute is DISPOS=*OBJECT).
- At least one page is not in the initial state or has already been written to the file with SAVE.

Example 1: truncating a file logically



Key

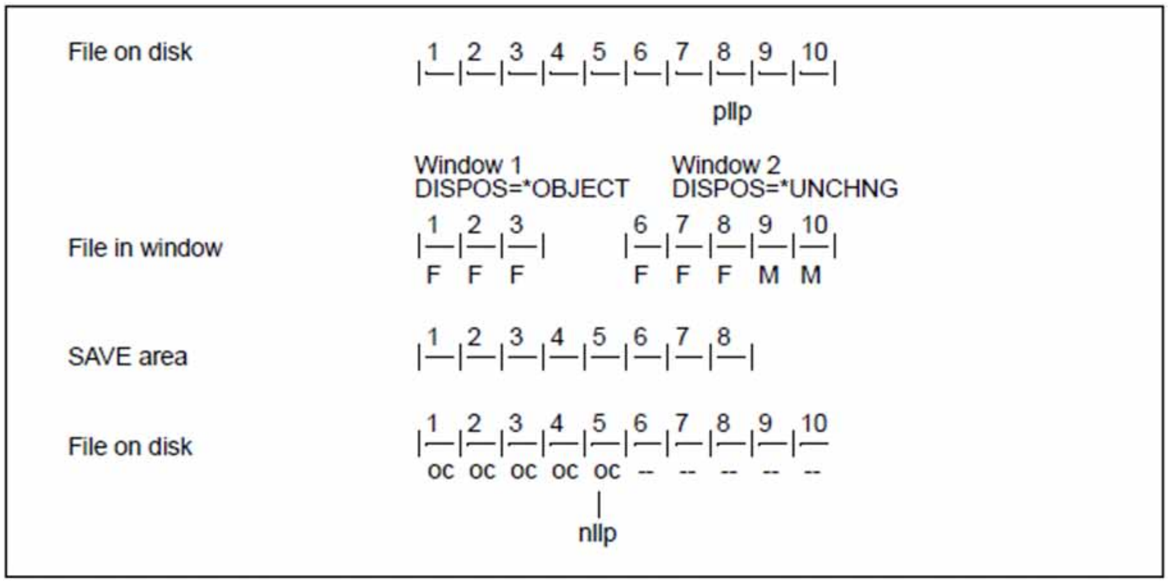
1...10	page numbers (file/window)
M	modified
nc	new contents
F	FRESHLY_OBTAINED
M	MODIFIED
oc	old contents
pllp	previous logical last page
--	undefined contents

Explanation

Window pages 4-10 were never accessed or were returned to the initial status (FRESH-LY_OBTAINED) by RESET. Page 5 constitutes the new logical EOF. The immediately preceding page (page 4) was never accessed. That is why pages 4 and 5 are logically truncated by SAVE.

Modified pages between the start of the file and the new logical EOF (pages 1 to 3) are written to the file. Pages which are in this area and have never been accessed (i.e. with page status FRESHLY_OBTAINED: not yet written to disk by an earlier SAVE operation) are deleted in the disk file, i.e. initialized with X'00'. Page 1 in the above example.

Example 2: truncating a file logically



Key

1...10	page numbers (file/window)
M	modified
nc	new contents
M	new logical last page
oc	old contents
pllp	previous logical last page
--	undefined contents

Explanation

There are no modified pages in the SAVE area beyond the last logical page (page 8). The condition for extending a file is therefore not fulfilled.
Since the last logical page, i.e. the last page of the SAVE area (page 8), and the preceding pages of the second window have not yet been accessed, these pages are still in the initial state. The file is truncated by these pages. Truncation ends when a page which does not appear in any window or which is not in a window with the attribute DISPOS=*UNCHNG is encountered.
In the example, pages 1, 2 and 3 have already been written to file with SAVE. They will hence retain their existing contents and will not be initialized with X'00'.

If page 9 belongs to the SAVE area, the file is extended up to this page (inclusive).

18.2.3 Extending or truncating a file physically

The physical end-of-file can be changed with the help of the MAP function.

Physical extension of files

When a window is defined with MAP, DIV ensures (for OPEN OUTIN | INOUT, but not INPUT) that disk space is allocated in 4K blocks for the file area that is represented by the window. For window pages that follow the physical end-of-file, DIV allocates space in 4K blocks.

Physical truncation of files

Blocks occupied by a file can be freed with the help of the FILE macro by specifying a negative SPACE value or with the command MODIFY-FILE-ATTRIBUTES ...RELEASE=. Blocks are released up to the last logical 4K page.

18.3 Multi-user mode

A PAM file can be created and processed using the following access methods:

- DIV
- UPAM (see chapter “UPAM – User Primary Access Method” (UPAM - User Primary Access Method))
- FASTPAM (see chapter “FASTPAM – Fast Primary Access Method” (FASTPAM - Fast Primary Access Method))

Note, however, that FASTPAM and DIV can process UPAM files with BLKCTRL=NO only. Whether a file may be processed by more than one user in parallel depends on the operand values specified for SHARUPD, MODE, LOCKENV and LOCVIEW when the file is opened.

Permissible combinations of opening a file in parallel

			USER B									
	SHARUPD=		***YES**			***NO**			***WEAK**			
		OPEN mode	INPUT	INOUT	OUTIN	INPUT	INOUT	OUTIN	INPUT	LMAP	INOUT	OUTIN
USER A	***YES**	INPUT	X	O		X			X	X		
		INOUT	O	O					X			
		OUTIN	O	O					X			
	***NO**	INPUT	X			X			X	X		
		INOUT							X			
		OUTIN							X			
	***WEAK**	INPUT	X	X		X	X		X	X	X	
		LMAP	X			X			X	X	O	
		INOUT							X	O		
		OUTIN							X			

Table 44: DIV: Permissible SHARUPD/OPEN combinations

LMAP:	INPUT LOCVIEW=MAP (only with DIV)
X:	OPEN permitted
O:	OPEN permitted only,

- if the same block-oriented access method is used by all (either UPAM/FASTPAM or DIV)
- **and** the same value is used by all for the LOCKENV operand (either LOCKENV=*HOST or LOCKENV=*XCS)
- **and** all are running on the same host or, with LOCKENV=*XCS, in an XCS network

Notes

- A file can be opened with SHARUPD=*WEAK for concurrent reading and writing.

  When users read with DIV-SHARUPD=*WEAK after specifying LOCVIEW=*MAP in the OPEN call, all window pages are read from the disk file into the window when executing MAP. DIV ensures that none of the file pages being read can be modified by any parallel SAVE from other DIV-SHARUPD=*WEAK users writing to the file.

  This protection against parallel writing does not exist for write operations with UPAM/FASTPAM.For this reason, reading with DIV-SHARUPD=*WEAK after specifying LOCVIEW=*MAP is compatible with writing with DIV-SHARUPD=*WEAK but not with any other write operations.

  In addition, the other conditions listed above for the 'O' entry must be fulfilled (all users opening the file with the same LOCKENV operand value and from the same host or, if from different hosts, with LOCKENV=*XCS).

  The conditions applying to reading with UPAM/FASTPAM-SHARUPD=*WEAK also apply to reading with DIV-SHARUPD=*WEAK after specifying OPEN LOCVIEW=*NONE.
- Users opening with DIV-SHARUPD=*YES are not compatible with users opening with UPAM/FASTPAM-SHARUPD=*YES.
- Read operations are always compatible irrespective of the access method, the values specified for SHARUPD and LOCKENV and the host.
- When files are accessed in SHARUPD=YES mode it is possible that processing will result in a file < 32 GB growing to become a file >= 32 GB.

  Here it is necessary to distinguish between two cases:

  - Callers who are prepared for this situation (with the specification LARGE_FILE=*ALLOWED in the FCB macro or EXCEED-32GB=*ALLOWED in the ADD-FILE-LINK command)
  - Unprepared callers (specification LARGE_FILE=*FORBIDDEN in the FCB macro or EXCEED-32GB=*FORBIDDEN in the ADD-FILE-LINK command).

  The size of the file in question is checked after every allocator call. If this check indicates that the file size is >= 32 GB and the LARGE_FILE=*FORBIDDEN attribute is set in the associated FCB, then processing is canceled and a corresponding return code is output in the local parameter list DIV(I) or a DMS message is issued.

18.4 Data consistency and update status

Consistency of data in the disk file in multi-user mode

By specifying SHARUPD=*YES in the OPEN function, a user can allow concurrent writes to a file.

i Write operations (SAVE) of different users are **not** coordinated by DIV, i.e. the user is responsible for synchronizing write operations.

Consistency of data in the disk file after a system crash

If SAVE is interrupted by a system crash or an aborted task, some modified pages may have been written to file, but not all.

In such cases, the contents of the disk file may be in an inconsistent state.

Consistency of data in a window in multi-user mode

Parallel tasks can lead to inconsistent data in a window only when another write-authorized user modifies the pages of the corresponding file region and one of the modified pages is read into a window following an access.

Users opening a file will not be affected by changes to window pages that are caused by concurrent writes if they:

- open with SHARUPD=*NO (no concurrent writes are permitted),
- open with SHARUPD=*WEAK for writing (no other user may write concurrently)
- open with SHARUPD=*WEAK for reading, provided LOCVIEW=*MAP was specified in the OPEN.

DIV then ensures that none of the file pages being read (when MAP is called) can be modified by any parallel SAVE from users writing to the file. Once the pages have been read, they may be modified by a parallel write operation.

i If the logical EOF is changed by a parallel write operation, it cannot be assumed that this modification will be taken into account for subsequent opens (e.g. when the window is accessed or in a call to SAVE).

Consistency of data in a window following a system crash

Data which has been modified only in the window but has not been written to the disk file will be lost.

Update status of data in a window

The data in a window will always be up-to-date unless a parallel write operation modifies the pages of the corresponding file region and one of the modified pages is read into a window following an access.

The following DIV features are significant in this context:

- A window page is read from the file to the window at the first access and will subsequently be modified by CPU instructions only. A subsequent access will cause the page to be read from the file only after a RESET.

- Whether an access will read a page from the file is also dependent on the fact whether the corresponding page exists in the file, i.e. whether the file is located before or after logical end-of-file.

  i The implementation is decisive for the question when a modification of the logical end-of-file by a write operation will take effect for another concurrent user of the same file. Users cannot rely on the fact that modifications of the logical end-of-file by a parallel write operation will be visible in their own windows.

19 EAM - Event Access Method

EAM is the access method for EAM files in BS2000 and has the following characteristics:

- EAM files are not cataloged. As a result of this, no disk access is necessary when an EAM file is opened.
- Each EAM file is automatically deleted when the job which opened it is terminated (temporary file).
- Communication between EAM and the user takes place only via the EAM control block (MFCB = Mini File Control Block). No facilities exist for modifying this control block at OPEN time.
- There is no label processing.
- EAM works exclusively with public volumes (pubsets). No distinction is made between disks with and without PAM keys (K and NK disks).
- The space requirements for the EAM routines and the runtimes for read and write accesses are less than for the standard access methods for cataloged files (UPAM, SAM, ISAM).
- An EAM file can be processed only by the calling job, but one job may open and process several EAM files.
- EAM is block-oriented and is based on blocks of 2048 bytes each. The system knows nothing about the record structure. For chained I/O, up to 16 sequential blocks may be transferred at one time.
- If a program is restarted using RESTART-PROGRAM, all existing EAM files belonging to the job are erased.

EAM functions

The EAM functions are available after calling the EAM macro. The following are the basic EAM functions:

Function	Meaning
CHECK CHECK_WAIT	Check and wait for I/O to terminate
CLOSE CLOSE_ALL	Close a file
ERASE ERASE_ALL	Erase a file
OPEN	Create and open a new file
READ	Read (blockwise, sequential or direct)
REOPEN	Open an existing file
WRITE	Write (blockwise, sequential or direct)

Table 45: EAM macro (functions)

The desired operation is selected by specifying a hexadecimal operation code in the MFCB, and initiated by the EAM macro. The effect is determined by the MFCB fields which EAM additionally evaluates after analyzing the operation code.

19.1 EAM processing

- Check operations
- Changing the processing attributes
- Overlapping input/output
- Handling object module files with EAM

19.1.1 Check operations

After a read or write call, control is returned to the user as soon as the requested operation has been accepted. In other words, there is no need to wait for this operation to be completed.

Before a read or write operation is initiated, however, the system waits for the preceding read/write operation (if any) to terminate (i.e. implicit check-and-wait operation). Similarly, when a close operation is requested, the system waits for the last read/write operation to terminate.

Thus, after the last in a series of read/write operations, a check operation is necessary only if the file is not immediately closed again or if, in the case of chained I/O, reading is continued until the end-of-file condition is reached (bit IDMFIB or IDMFEF of sense byte =1) and the number of blocks transferred is not equal to 0.

Example

In an EAM file, 3 read operations are performed. The file is not closed after this, because it is still required for later I /O operations. However, these I/O operations are not requested until after processing of the blocks that were read has been completed:

```
READ
:
CHECK/WAIT                                  Wait for termination of the last
:                                           I/O operation.
Processing of the blocks that were read
:
Further I/O operations
```

19.1.2 Changing the processing attributes

- chained/non-chained input/output
- number of blocks to be transferred.

If one of these values is to be changed during processing of the file, the following actions are required:

1. Close file
2. Modify fields in MFCB
3. Reopen file

The value for the number of blocks to be transferred should be changed if, for example, fewer blocks are to be transferred in the last write operation than was specified when the file was opened.

Example

99 blocks are to be written to an EAM file. Chaining is to be used, with 15 blocks being transferred per write call (byte IDMFNHP in the MFCB). The following operations are then requested:

```
OPEN (new file) -> WRITE -> WRITE -> WRITE -> WRITE ->
WRITE -> WRITE -> CLOSE *) -> REOPEN -> WRITE ->
REOPEN -> WRITE -> CLOSE
```

*) After 6 write operations, there are still 9 blocks to be written. The file must therefore be closed and then reopened, this time with a value of 9 set for the number of blocks to be transferred.

19.1.3 Overlapping input/output

A reduction in processing time can be achieved by means of asynchronous I/O operations. Once an I/O operation has been initiated, control is immediately returned to the program in order to enable other processing to take place in parallel with the physical I/O. The next I/O operation is then initiated using a second I/O area that does not overlap the first, and so on.

The figure 24 illustrates the overlapping of processing and input operations.

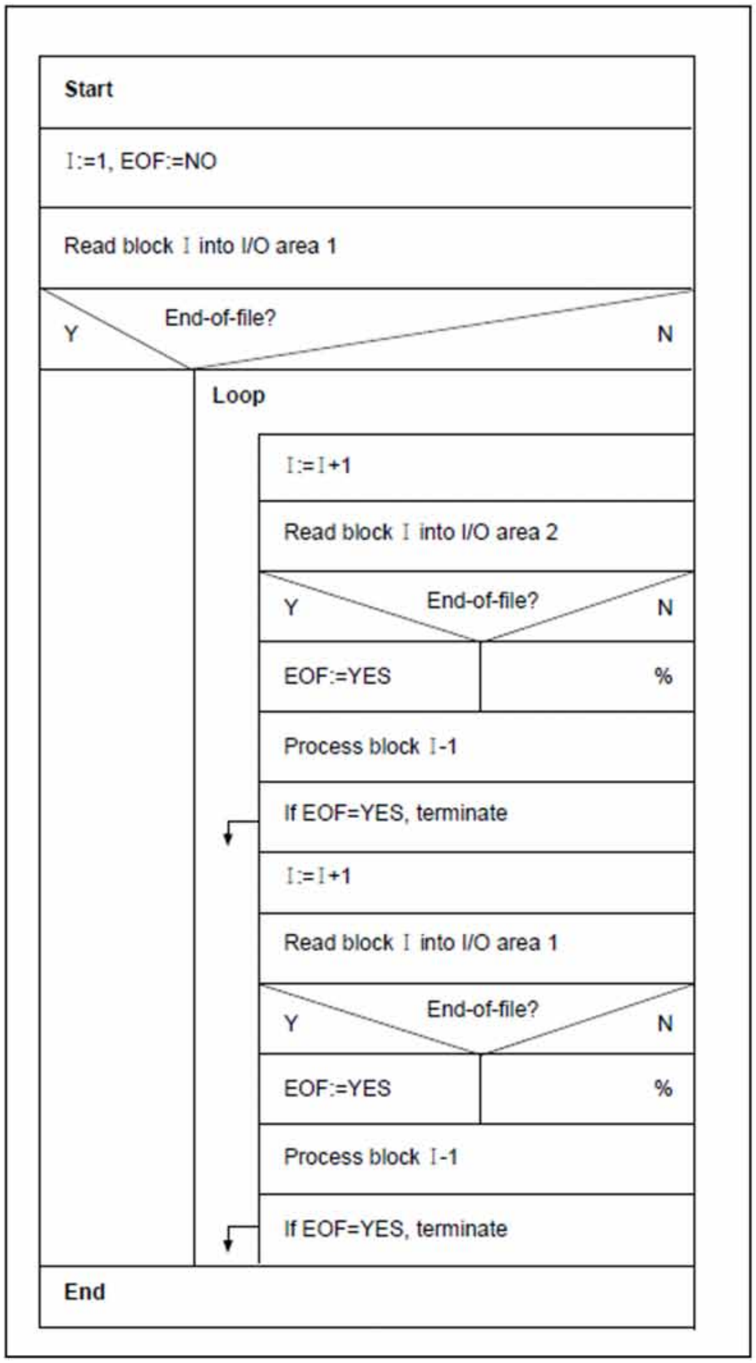


Figure 24: EAM: overlapping input/output

19.1.4 Handling object module files with EAM

Each job can process exactly one object module file. If bit IDMFOO of the option byte is set, all operations relate to the object module file. The actions involved in opening or reopening a file are illustrated in the following diagram:

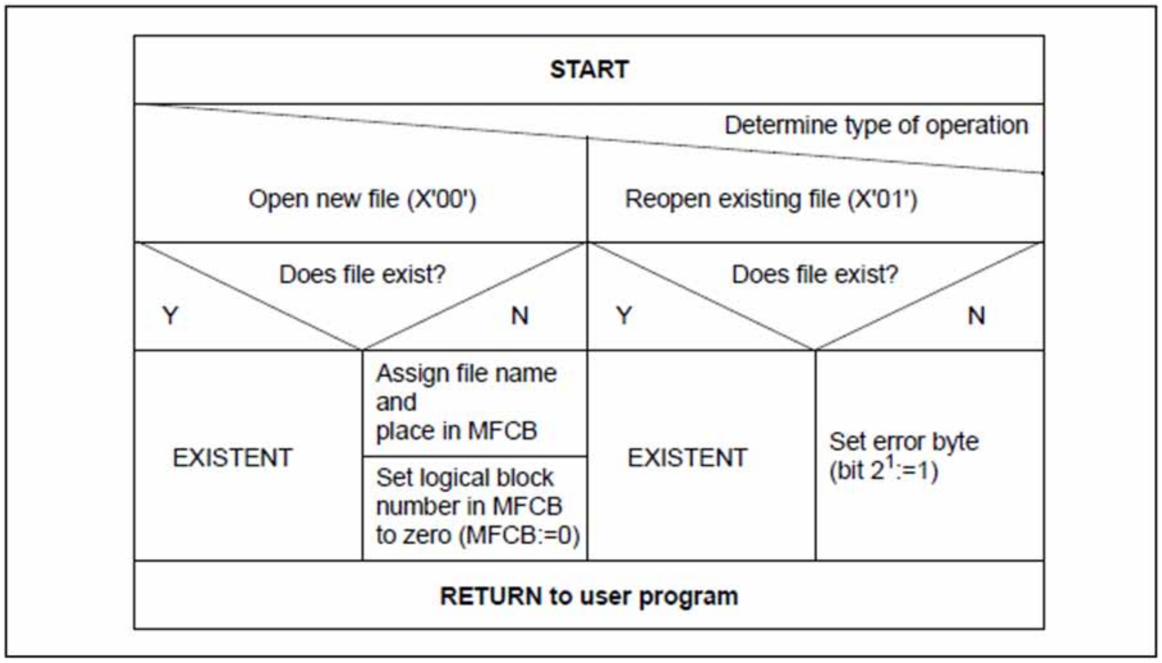

START
Determine type of operation
Open new file (X'00')
Reopen existing file (X'01')
Does file exist?
Y
N
Does file exist?
Y
N
EXISTENT
Assign file name and place in MFCB
Set logical block number in MFCB to zero (MFCB:=0)
EXISTENT
Set error byte (bit 2¹:=1)
RETURN to user program

Figure 25: Actions when an EAM file is opened

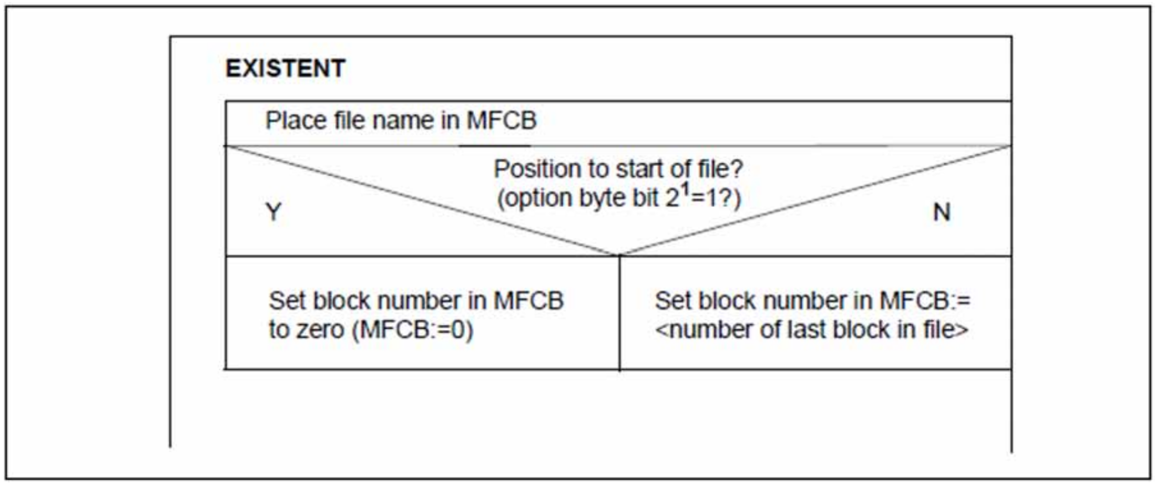

EXISTENT
Place file name in MFCB
Position to start of file? (option byte bit 2¹=1?)
Y
N
Set block number in MFCB to zero (MFCB:=0)
Set block number in MFCB:= <number of last block in file>

Figure 26: Sequence when opening an object module file

20 FASTPAM - Fast Primary Access Method

FASTPAM (FAST Primary Access Method) is a block-access method for NK4 disk files. It is comparable with UPAM in terms of functionality, but is far superior to it in terms of performance, especially with multi-server systems. FASTPAM is characterized by the following features:

- accelerated I/O due to a reduced number of CPU instructions,
- a clear and efficient interface, and
- support for I/O in data spaces.

FASTPAM offers a subset of the functionality of UPAM.

The performance gains are achieved by extracting preparatory I/O actions from the I/O path and executing them before the OPEN independently of each I/O operation. Such actions include:

- loading parameter lists into resident memory,
- preparing the I/O path, and
- creating memory-resident buffer areas.

The I/O paths consist of the entire resident memory that is required by the system for one I/O along with all the data that can be generated in advance when creating the environment. Channel programs are typically set up in this memory area before the inputs/outputs. Memory for the parameter list and I/O area pool is fixed by the user (i.e. is no longer pageable and cannot be released by the user).

Using FASTPAM management calls, the user creates two areas:

- the FASTPAM environment, and
- the FASTPAM I/O area pool.

The FASTPAM environment is used to hold the FASTPAM parameter lists with which I/O jobs are submitted. It also includes memory areas of the operating system (the so-called I/O paths).

The FASTPAM I/O area pool is an area with 4-Kbyte buffers.

If the user has the required FASTPAM authorization (entry in the user catalog), these areas can be optionally made memory-resident. FASTPAM areas may be used on a cross-file basis and, if they are in common memory pools, on a cross-task basis and thus allow for efficient usage of the resource of “resident memory”. The size of the FASTPAM environment is determined by the maximum number of I/O operations to be executed in parallel.

The FASTPAM interface is an SVC interface. Jobs are defined via a parameter list; result reports are returned via a return code in the parameter list (not via exits).

FASTPAM permits I/Os to be carried out directly via data spaces. I/O area pools are set up in data spaces for this purpose. Note, however, that these I/O area pools can only be nonresident.

Two groups of functions are offered at the FASTPAM interface:

- Management functions (FPAMSRV macro)
  - creating a new FASTPAM environment or joining an existing FASTPAM environment and FASTPAM I/O area pool
  - disconnecting and disabling a task from FASTPAM environments and FASTPAM I/O area pools
  - opening and closing files

- Access functions (FPAMACC macro)
  - synchronous reading and writing of logical blocks
  - asynchronous reading and writing of logical blocks
  - waiting for the end of asynchronous I/O jobs
  - reporting the end of asynchronous I/O jobs

FASTPAM is primarily useful for system support software applications in which I/O performance (path lengths, throughout) plays an important role.
Tape processing and remote file access (RFA) are not supported.

i In shared-update mode, several tasks can access a file concurrently. The FASTPAM access method **does not** provide a synchronization mechanism for shared access to a file. Appropriate locking mechanisms analogous to those offered by UPAM must therefore be supplied by the **user**.

FASTPAM macros and their functions

Two macros are available to the user for processing files (FPAMSRV and FPAMACC). These can be used to execute various functions and operations.

Macro	Function	Brief description
FPAMSRV	ENABLE ENVIRONMENT	Enable system environment for FASTPAM processing
	ENABLE IOAREA POOL	Enable I/O area pool for FASTPAM processing
	OPEN FILE	Open file for processing with FASTPAM
	CLOSE FILE	Close a file (opened with FASTPAM);the last-page pointer can optionally be specified.
	DISABLE IOAREA POOL	Enable I/O area pool for FASTPAM processing
FPAMACC	ACCESS FILE	Process a file (opened with FPAMSRV)

Table 46: Functions of the FASTPAM macro

20.1 FASTPAM areas

In order to process files with FASTPAM, the user must first create a FASTPAM environment and a FASTPAM I/O area pool.
During this process, system resources are prepared, and some preparatory actions such as prevalidating and possibly fixing user areas (parameter list, I/O buffers) are performed by the system.

The facility to create a FASTPAM environment and a FASTPAM I/O area pool in resident memory provides the user with an instrument to reserve system resources for an unrestricted amount of time. Special authorization is required for this purpose, however (see “FASTPAM authorization” (Processing files with FASTPAM)

i In the following description of the FASTPAM access method, the terms “FASTPAM environment” and “FASTPAM I/O area pool” are occasionally abbreviated and simply referred to as the “environment” and the “I/O area pool”.

20.1.1 FASTPAM environment (memory areas)

A FASTPAM environment consists of cross-file and cross-task system and user memory areas which are created before files are opened and repeatedly used whenever files are accessed.

- User memory area:
  The user memory area is a virtually contiguous memory area in which the FPAMACC parameter lists (... ACC=ACCESS) are located. The parameter lists generated with the FPAMACC macro (see below) are required for file access. They must lie in a contiguous area of the virtual address space. They are made resident by FASTPAM if the required FASTPAM authorization is present. The number of parameter lists determines the maximum number of parallel accesses to files opened with this environment.
- System memory areas (only with FASTPAM authorization):
  An I/O path is created by the system for each FPAMACC parameter list. This I/O path includes all the resident class 3 memory that is required for an I/O by the system. Its size depends on the value specified in the MAXIOLEN operand:
  - 1 Kbyte for MAXIOLEN = *MINI
  - 2 Kbytes for MAXIOLEN = *MAXI

Attributes

A FASTPAM environment is defined by the following fixed attributes:

- name
- scope
- address of the FPAMACC parameter lists
- number of FPAMACC parameter lists
- maximum I/O length for all file accesses with this environment
- eventing: yes/no; if yes: short ID of the event item (event ID)

A FASTPAM environment is uniquely identified by its name and scope. An environment can be used for any number of files by any number of tasks that lie in the scope of the environment. The scope of an environment is determined by the memory area in which the FPAMACC parameter lists are created:

- If the area lies in the task-local address space, the environment is valid for that task only, i.e. no other task can join the environment even if it has same parameters.
- If the area is in a common memory pool, the scope defined by the SCOPE operand in the ENAMP macro (ENAble Memory Pool) applies.

20.1.2 FASTPAM I/O area pool

A FASTPAM I/O area pool is a virtually contiguous memory area from which appropriate I/O buffers can be retrieved when accessing files. The I/O area pool may be located in the tasklocal address space, in a data space and, for multi-tasking systems, in a common memory pool. In the latter case, however, the memory pool must begin with the same virtual address (ENAMP macro; operand FIXED=YES) for each task.

Attributes

A FASTPAM I/O area pool is defined by the following fixed attributes:

- name
- scope
- address
- length

A FASTPAM I/O area pool is uniquely identified by its name and scope. Like the FASTPAM environment, an I/O area pool may be used for any number of files by any number of tasks which lie in the scope of the I/O area pool. The scope of an I/O area pool is determined by its address:

- If the address lies in the task-local address space, the I/O area pool is valid for that task only, i.e. no other task may join the I/O area pool even if it has the same parameters.
- If the address is in a common memory pool, the scope defined by the area specified in the SCOPE operand of the ENAMP macro (ENAble Memory Pool) is assumed.
- If the address is located in a data space, the scope that was defined by the SCOPE operand when creating the data space is assumed.

20.2 Processing files with FASTPAM

File format

FASTPAM will only process PAM files with BLKCTRL=NO/DATA and BLKSIZE=(STD,2n), where n=1,2,3...8. Files which are not of this format must first be converted. A block control field is added to files with BLKCTRL=DATA.

FASTPAM authorization

In order to receive resident memory via FASTPAM calls, the user must be authorized in the user catalog. To do so you must call the SHOW-USER-ATTRIBUTES command and there enter the value *EXCLUSIVE in the DMS-TUNING-RESOURCES field. Users who do not have such authorization may also use the FASTPAM access method, but no resident areas are maintained. FASTPAM behaves like UPAM in such cases: only a small, non-resident part of the I/O paths is created by the system; the area of the parameter lists and the I/O area pool are not fixed. In other words, the paths must be recreated, and user areas must be validated and fixed for each I/O, as a result of which performance gains typically achieved with FASTPAM are lost.

If no memory can be made resident, FASTPAM behaves as if the required FASTPAM authorization were missing. As a result a performance level equivalent to or better than that of UPAM is being offered.

Making memory areas resident

One of the primary purposes of FASTPAM is to enable rapid file access. This is done by making the required system environment available in resident memory before the first file is accessed. In order to do this, the memory areas containing the user parameter lists and the I/O areas (both of which must be supplied by the user) are made memory-resident by the “FASTPAM page fixing” mechanism.

This is essentially the same procedure that is performed by PPAM for the I/O area for every I/O operation when other access methods are used. The only difference with the other access methods is that the I/O area is released on completion of each I/O operation.

With FASTPAM, the user can define how long the parameter lists and the I/O areas are fixed (with ENABLE /DISABLE ENVIRONMENT and ENABLE/DISABLE IOAREA POOL). You can use them during that period. Validation is only required once at the beginning, since fixed areas cannot be released.

The ENABLE ENVIRONMENT function is also used to request the system memory that is required for I/O operations (once for each I/O operation that can be concurrently executed). A major part of this memory, i.e. the area used by IOCTRL, is always resident. This is also true for the other access methods, but the area is reallocated for each I/O and is not permanently reserved.

The rest of the system memory consists of a FASTPAM work area, which primarily contains the parameter list to call PPAM.

The fixing of memory areas as described above is performed only if the user ID has the required FASTPAM authorization.

In this case and if the appropriate memory areas are fixed, the resulting environment or I/O area pool is said to be “resident”.

A “resident environment” thus refers to:

- prevalidated parameter lists in resident memory
- system memory that is reserved in advance
- a resident FASTPAM work area.

Similarly, a “resident I/O area pool” implies:

- prevalidated I/O areas in resident memory.

Prerequisites for resident FASTPAM areas

- The user has specified the appropriate parameters (macro FPAMSRV, FCT=*ENAENV/*ENAIPO, operand RES=YES).
- The user has the required FASTPAM authorization.
- No data spaces are being used.
- An adequate amount of main memory is available.
- A sufficient number of resident pages were allocated on calling the program (command START-PROGRAM /LOAD-PROGRAM, operand RESIDENT-PAGES). When resident pages are allocated in the program call, the maximum value defined in the user catalog and the system-global limit for resident memory pages must not be exceeded.

20.3 Multi-user mode

A PAM file can be created or processed in parallel using the following access methods:

- FASTPAM
- UPAM (see chapter “UPAM – User Primary Access Method” (UPAM - User Primary Access Method))
- DIV (see chapter “DIV – Data In Virtual” (DIV - Data In Virtual))

Permissible combinations of opening a file in parallel

			USER B								
	SHARUPD=		***YES**			***NO**			***WEAK**		
		OPEN mode	INPUT	INOUT	OUTIN	INPUT	INOUT	OUTIN	INPUT	INOUT	OUTIN
USER A	***YES**	INPUT	X	O		X			X		
		INOUT	O	O					X		
		OUTIN	O	O					X		
	***NO**	INPUT	X			X			X		
		INOUT							X		
		OUTIN							X		
	***WEAK**	INPUT	X	X		X	X		X	X	
		INOUT							X		
		OUTIN							X		

Table 47: FASTPAM: Permissible SHARUPD/OPEN combinations

X: OPEN permitted

O: OPEN permitted only

- if the same block-oriented access method is used by all (either UPAM/FASTPAM or DIV)
- **and** the same value is used by all for the LOCKENV operand (either LOCKENV=*HOST or LOCKENV=*XCS)
- **and** all are running on the same host or, with LOCKENV=*XCS, in an XCS network

Notes

- A file can be opened with SHARUPD=*WEAK for concurrent reading and writing. (SHARUPD=*WEAK is possible with UPAM and DIV only.)

  Exception:
  When users read with DIV-SHARUPD=*WEAK after specifying LOCVIEW=*MAP in the OPEN call, parallel opening for writing with UPAM/FASTPAM is not permissible.

  The conditions applying to reading with UPAM/FASTPAM-SHARUPD=*WEAK also apply to reading with DIV-SHARUPD=*WEAK after specifying OPEN LOCVIEW= *NONE.

- Users opening with DIV-SHARUPD=*YES are not compatible with users opening with UPAM/FASTPAM-SHARUPD=*YES.
- Read operations are always compatible irrespective of the access method, the values specified for SHARUPD and LOCKENV and the host.
- Illegal combinations result in an OPEN error.
- When files are accessed in SHARUPD=YES mode it is possible that processing will result in a file < 32 GB growing to become a file >= 32 GB.

  Here it is necessary to distinguish between two cases:

  - Callers who are prepared for this situation (with the specification LARGE_FILE=*ALLOWED in the FCB macro or EXCEED-32GB=*ALLOWED in the ADD-FILE-LINK command)
  - Unprepared callers (specification LARGE_FILE=*FORBIDDEN in the FCB macro or EXCEED-32GB=*FORBIDDEN in the ADD-FILE-LINK command).

  The size of the file in question is checked after every allocator call. If this check indicates that the file size is >= 32 GB and the LARGE_FILE=*FORBIDDEN attribute is set in the associated FCB, then processing is canceled and a corresponding return code is output in the local parameter list FPAMSRV(I) or a DMS message is issued.

The following points apply to the FASTPAM access method:

- A file can be concurrently processed with FASTPAM by multiple tasks (multiple OPENs with SHARUPD=*YES and MODE=*OUTIN/*INOUT).

  i When a file is accessed in shared-update mode, appropriate synchronization routines must be supplied by the user if no such routines are built into the software product being used. In contrast to UPAM, FASTPAM does not provide any locking mechanism for this purpose.

- Opening with FASTPAM and UPAM

  A file can be opened in parallel by multiple tasks with FASTPAM as well as UPAM. Processing of the file is controlled by the operands MODE and SHARUPD (see below) of the OPEN function. Although FASTPAM does not support SHARUPD=*WEAK, it otherwise behaves exactly like UPAM: both for exclusive FASTPAM open operations and also when UPAM and FASTPAM are mixed.

  When a file is concurrently accessed with UPAM and FASTPAM, the UPAM user must also synchronize operations with the FASTPAM user, since UPAM page locks are only effective when used by both sides, and since FASTPAM has no page-locking mechanism.
- Opening with FASTPAM and DIV

  FASTPAM interacts with DIV exactly like UPAM. Parallel processing is permitted only if the file is opened with INPUT by all users.
- Files on shared pubsets (SPVS) are supported by FASTPAM: FASTPAM users can access an SPVS from different systems and read from it concurrently and can also read in parallel with UPAM and DIV users.
- Cross-system file access (RFA) is not supported.

20.4 Data consistency

Data consistency in multi-user mode

The FASTPAM access method does not provide a synchronization mechanism for shared access to a file (shared-update mode). Appropriate synchronization routines must therefore be supplied by the user if no such routines are included in the software product being used. If FASTPAM, UPAM and DIV applications are all operating in shared-update mode, a common synchronization mechanism must be used for all accesses.

Data consistency following a system crash

If an error occurs during an ACCESS FILE job, it is not possible specify whether and how much data has been transferred. The writing of a block cannot be treated as an atomic operation. The contents of the file may be inconsistent in such cases.

20.5 Functional differences between UPAM and FASTPAM

- FASTPAM can only be used to process PAM files with the following file attributes:
  - BLOCK-CONTROL-INFO=*NO or *DATA
  - BUFFER-LENGTH = *STD(2n), n=1,2...8
- The following functions are not supported by FASTPAM:
  - DUMMY files
  - tape processing
  - RFA
- FASTPAM supports synchronous and asynchronous read and write operations. The following operations are offered by UPAM, but are not supported by FASTPAM:
  - CHK
  - LOCK / UNLOCK
  - LRD / LRDWT / WRTWU
  - SETL
  - SYNC
- FASTPAM can handle I/Os in data spaces.
- The functionality of the UPAM operation SETLPP is included in the framework of FASTPAM CLOSE processing.
- The function SHARUPD=*WEAK is not supported (see also "Multi-user mode").
- An implicit WAIT is not possible.
- Within an OPEN / CLOSE bracket, asynchronous I/Os can either be terminated only by WAIT, or their termination can be reported only via the eventing mechanism.
- It is not possible to specify a relative page number.

21 ISAM - Indexed-Sequential Access Method

ISAM, like SAM, is a record-oriented access method for disk files. In contrast to SAM files, however, ISAM files can be processed directly as well as sequentially: With the aid of a key area in each record (known as the ISAM key or primary key), the records are placed in the data blocks of an ISAM file. DMS then manages these data blocks with the aid of index blocks such that each record can be accessed directly by way of its key.

Block formats

There are two versions of the access method ISAM, capable of processing files with two different block formats (see section “Block formats for disk files”).

- NK-ISAM (Non-Key ISAM) processes files with the block format “DATA”: These do not contain a separate PAM key. The DMS management information is kept in a block control field within the PAM page.
- K-ISAM (Key ISAM) processes files with the block format “PAMKEY”: these are characterized by the fact that DMS management information for each PAM page is kept in a separate PAM key located outside the page.

By means of the BLKCTRL operand in the macros FILE and FCB or the BLOCK-CONTROL-INFO operand in the ADD-FILE-LINK command, the user can select whether a K file or an NK file is to be processed:

- BLKCTRL=DATA/DATA2K/DATA4K specifies an NK-ISAM file
- BLKCTRL=PAMKEY declares a K-ISAM file

If you make no explicit specification for BLKCTRL, the BLKCTRL system parameter controls in general whether the file is created ond K disks with the BLKCTRL=PAMKEY or DATA/NO attribute.
On BS2000 OSD/BC V11.0 and higher and specifically for creating ISAM files, the ISBLKCTL system parameter controls whether the file is created as an NK-ISAM file on K disks.
Because in practice K-ISAM is becoming less and less important, the default setting for ISBLKCTL is c'NONKEY', e. g. by default, ISAM files are created as NK-ISAM on K disks as well.

NK-ISAM offers the user all functions of K-ISAM, with almost identical interfaces. In contrast to K-ISAM, NK-ISAM uses ISAM pools for read and write access: these are areas in main memory in which large portions of the ISAM files being processed are held. The resulting reduction in the number of physical I/O operations provides accelerated access, particularly in the case of non-sequential accesses. (Further information on ISAM pools can be found under “Overview of the most important functions of NK-ISAM”).

The records of NK-ISAM files may contain, in addition to the ISAM key (primary key), up to 30 so-called secondary keys (alternate indices), via which the user can locate the desired records in exactly the same way as via the primary key. This added function makes processing of ISAM files far more flexible than was possible via the primary key alone.

NK-ISAM files must be further differentiated into NK2-ISAM files and NK4-ISAM files (see also the section “NK4 format”).

This section describes the access method ISAM, i.e. the functions supported by NK-ISAM and K-ISAM. References are made at the appropriate points to special features and incompatibilities, and these are also summarized in section “Compatibility”.

The following sections first provide a brief description of the functions of NK-ISAM. This is followed by a description of the file structure of NK-ISAM and K-ISAM files, the processing of NK-ISAM files in ISAM pools, ISAM-specific processing attributes, shared-update processing, OPEN modes, and programming tips for ISAM processing. Further sections contain notes on the use of NK-ISAM and on conversion from K-ISAM to NK-ISAM and information about the “crash behavior” of ISAM files.

Overview of the most important functions of NK-ISAM

ISAM pools

The number of I/O operations can be reduced for non-sequential processing if large parts of the file can be kept resident in main memory. ISAM pools contain, in addition to the data and index blocks, the necessary management information. Users can create and manage their own ISAM pools via a command interface or a macro interface, thereby allowing them to optimize file processing. If the user does not make use of this option, NK-ISAM creates user-specific standard pools in which it then buffers the files to be processed.

Key compression

The length of the index entries is minimized by using compressed keys. This increases the number of keys which can be kept in an index block.

Locks

External locks are implemented as key locks or, for sequential processing, as extent locks.

Block merge

NK-ISAM guarantees that each data block will be at least 40% full and each index block at least 45% full. If the contents of a block drop below this limit, they are moved to neighboring blocks or merged with the contents of a neighboring block.

Secondary keys

In addition to the ISAM key (primary key), the user may define up to 30 secondary keys in a record. A record can then be accessed via any of these secondary keys (alternate indices). A command interface and a macro interface permit the user to define and delete secondary keys, as well as to request information on them. For access to records via the secondary keys, NK-ISAM has additional operands in the macros for read and pointer operations.

21.1 ISAM file structures

ISAM files consist not only of data blocks but also, for example, of blocks which contain management information for ISAM processing. An NK-ISAM file consists of data blocks, index blocks, a control block, free blocks and, possibly, overflow blocks, while a K-ISAM file consists of data blocks, index blocks and free blocks. Since the control block and the overflow blocks are created and managed by ISAM, there is no difference, at the user interface, in the processing of NK-ISAM and K-ISAM files.

The following sections describe the structural elements of an ISAM file, starting with the user's records, which are stored in data blocks and, with NK-ISAM, in some cases extend into so-called “overflow blocks”. In NK-ISAM, the data blocks are managed with the aid of the index blocks and the control block, in K-ISAM only with the aid of the index blocks.

21.1.1 File areas

Normal data blocks, index blocks, indexed data blocks, free blocks, the control block, and overflow blocks constitute the used area of an ISAM file. The unused file area, by contrast, is that part of a file which the user can release at any time (via the SPACE operand of the FILE macro or MODIFY-FILE-ATTRIBUTES command).

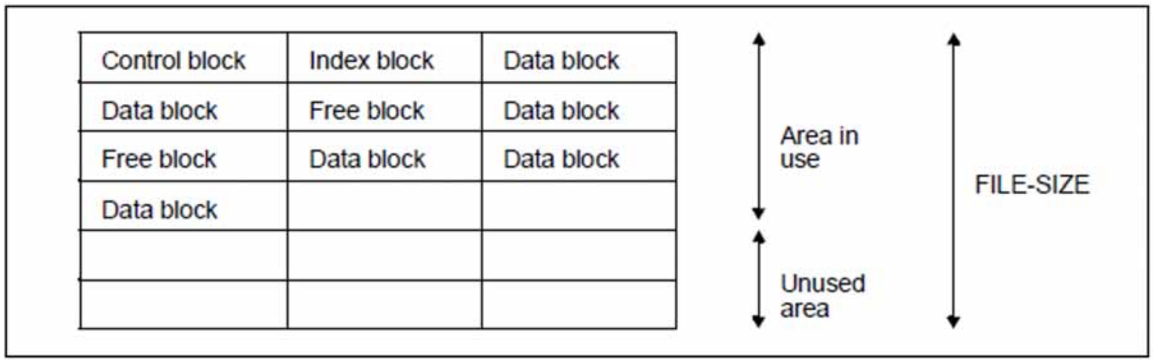


Figure 27: File areas of an ISAM file

Each field in this figure corresponds to one PAM page. Empty fields are PAM pages which have not yet been used by ISAM.

FILE-SIZE	Total size of the ISAM file (here: 18 PAM pages).
Area in use	All blocks which have been used at least once for the file, even if they are now free (here: HIGH-US-PA = 10).
Unused area	Blocks which have not yet been used by ISAM and can be released at any time (here: 8 PAM pages).

21.1.2 Record without a secondary key

ISAM records are characterized by an index area, which is in the same position, and has the same structure, in each record of a file. The index area always contains a record key which identifies the record. In addition, the index area may contain flags which permit searching for records on the basis of certain characteristics. The total index area (key + flags) may not exceed 255 bytes in length.

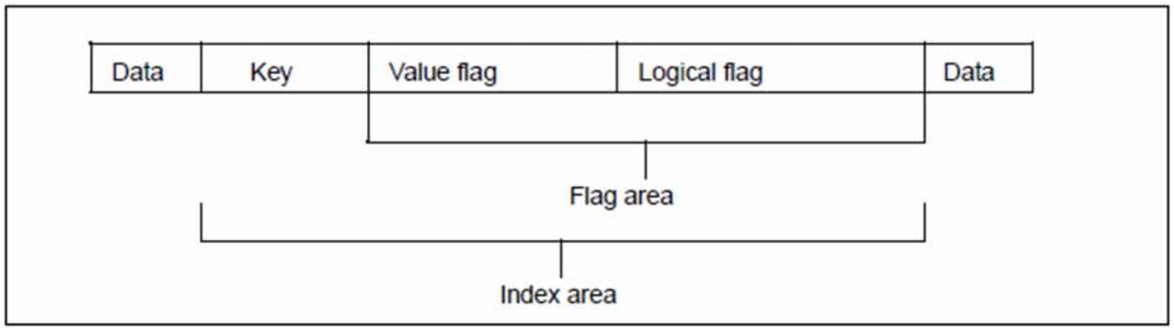


Figure 28: Structure of the ISAM index area

ISAM keys

Each record in an ISAM file must have a key with a length of at least one byte. The position of this ISAM key in the record is defined for all records of the file in FILE/FCB (operand KEYPOS) or in the ADD-FILE-LINK command (operand KEY-POSITION). The user must take the record format (see "ISAM record formats" (Record with secondary keys (NK-ISAM))) into account when defining the key position. In files with variable-length records, for example, the ISAM key cannot begin before position 5 (KEYPOS >= 5) because 4 bytes are needed for the record length field and for control information.

The length of the ISAM key is defined via the KEYLEN operand in FILE/FCB or via the KEY-LENGTH operand in ADD-FILE-LINK. When defining the key length, the user must remember that the maximum index length must not exceed 255 bytes (key + flags).

In K-ISAM, the key length may affect the file size and processing performance. Since the complete key is placed in the index block, the key length will determine how many index entries will fit into one index block.

With NK-ISAM, DMS uses key compression to optimize the number of index entries in each index block, so that longer keys are, on average, compressed to a length of 3-5 bytes and each index block can hold, on average, 160 entries. The number of index entries in an index block affects, in turn, the total number of index blocks needed, and thus the number of index levels in the file and the performance when accessing a record.

Duplicate ISAM keys

The principle of ISAM file processing is based on unique keys. If DUPEKY=YES is specified in FILE/FCB or if DUPLICATE-KEY is specified in the ADD-FILE-LINK command, the user can create files which may contain duplicate keys.

During write operations a record with a key which already exists in the file is always written after the last existing record with the same key. The records with identical keys are thus stored in the file in the chronological order in which they were written.

NK-ISAM adds a time stamp to records with keys which already exist in the file. Via the time stamp the records can be kept in the correct order, i.e. corresponding to the time at which they were written.

Sequential read operations always return the records in the order in which they were written. For "reverse" reading, the records are returned in the opposite sequence.

Non-sequential reading with the record key always returns the first record of a group of records with the same keys. The remaining records of the group must then be read sequentially.

If, during shared-update processing of NK-ISAM files, a record lock is set, then this lock affects all records with the same key.

> i An ISAM file with a large number of records with duplicate keys may lead to a considerable drop in performance. In the case of NK-ISAM files with very long records, it should be noted that the addition of time stamps can result in the creation of overflow blocks.

Flags in the ISAM index

In addition to the ISAM key, value flags or logical flags can be used to search for records. These flags follow the key in the ISAM index area of each record, the value flags preceding the logical flags if both types are used. In NK-ISAM, the flags are not included in the index entry, but are evaluated sequentially within NK-ISAM. For the user, this means that there will be a performance drop if flagged records are used, but processing of such records is otherwise the same as in K-ISAM.

> i Flag processing in NK-ISAM is considerably less efficient than in K-ISAM.

In K-ISAM, the flags are included in the index entries and are propagated as a component of the ISAM index – through all index levels of the file. However, inclusion of the flags in the index entry is carried out differently for value and logical flags.

The position of the value flag is determined by the position and length of the preceding key (KEYPOS + KEYLEN) and its length is defined by means of VALLEN in the FILE and FCB macros or with the VALUE-FLAG-LENGTH operand in the ADD-FILE command. For the logical flag, the position is determined by the sum of the key position, the key length and

the length of the value flag. The length is determined by the LOGLEN operand in the FILE and FCB macros or by the LOGICAL-FLAG-LENGTH operand in the ADD-FILE-LINK command.

When searching for a record with the aid of a value or logical flag, the contents of the flag area of the records being checked are compared with the specified value or with a logical bit mask. DMS returns the first record which fulfills the search condition.

In the case of K-ISAM, the value information of the index entries is created on the basis of what the user specifies for VALPROP in the FILE and FCB macros or for PROPAGATE-VALUE-FLAG in the ADD-FILE-LINK command. If the function MAX (MIN) is specified, each index entry contains the maximum (minimum) value flag in the block of the next lower level (data or index block) to which it points. Whether the propagation of the maximum or minimum value flag serves any purpose depends on how the records are subsequently to be retrieved. The usefulness of a value flag increases the more evenly the value flags rise or fall as the ISAM key ascends.

In the logical flags, binary attributes of the object described by the record can be encoded in individual bits. All logical flags of a data (or index) block are ORed together and passed on to the (next-higher) index entry. When searching for records with certain attributes, DMS can decide, immediately after checking the index entry, whether or not the block that this entry points to contains records with these attributes.

If the value for VALLEN (or VALUE-FLAG-LENGTH) or LOGLEN (or LOGICAL-FLAG-LENGTH) is not zero, DMS assumes that this is a flagged file. If the file already exists (i.e. the OPEN type is INPUT, INOUT or EXTEND), the catalog values for VALPROP (or PROPAGATE-VALUE-FLAG) and/or VALLEN (or VALUE-FLAG-LENGTH) are used. LOGLEN (or LOGICAL-FLAG-LENGTH) must always be specified if the file was created with a flag area. The catalog values cannot be modified by means of a FILE or FCB macro or ADD-FILE-LINK command.

Example: value flags

Let us assume that the “year of birth” is to be used as the value flag in a personnel file and that VALPROP=MIN was specified in the catalog entry. This would mean that the lowest birth year of the personnel records contained in each block would be transferred to the index block.

- Suitable query: which staff members were born before 1950 ? – Thanks to the value flag, entire blocks can be immediately skipped when searching.
- Unsuitable query: which staff members were born after 1950 ? – In this case, all records of a block must be checked. A file with VALPROP=MAX would be more appropriate here.

ISAM record formats

ISAM files may have fixed-length or variable-length records, i.e. records with format F or V. This format is specified in the RECFORM operand of the FCB and FILE macros or with the RECORD-FORMAT operand of the ADD-FILE-LINK command.

If RECFORM=V is specified, the user must remember, when defining the key position and the maximum record length, that the data is preceded by a 4-byte field. For fixed-length records (RECFORM=F), the user does not need to worry about this 4-byte field when defining the key position, although ISAM still. internally, places a 4-byte field in front of each record.

The table below indicates the effects of the record format on the record length defined by means of a macro or command.

Operand in FILE / FCB	**Operand in ADD-FILE-LINK**	**Meaning of RECSIZE or RECORD-SIZE**
RECFORM=F	RECORD-FORMAT=*FIXED	Specifies, in bytes, the length valid for all records
RECFORM=V	RECORD-FORMAT=*VARIABLE	Specifies the maximum length of a record; RECSIZE=BLKSIZE or RECORD-SIZE= BUFFER-LENGTH, as appropriate, are then used as default values

Table 48: Effect of the record format on the record length for ISAM

If RECSIZE approaches BLKSIZE (implicitly or explicitly), overflow blocks may be created in NK-ISAM (see section “Overflow block (NK-ISAM)”) and this may lead to an increase in the number of I/O operations.

If, for access to an existing file, the value specified for RECSIZE or RECORD-SIZE is too small, this may affect read operations: if DMS reads a longer record, it returns only the number of bytes specified in RECSIZE to the receive field defined by the use and then aborts the read operation with an error message. The user can intercept this with a suitable error routine in his/her program.

21.1.3 Record with secondary keys (NK-ISAM)

In addition to the ISAM key (primary key), the records of NK-ISAM files may contain up to 30 secondary keys via which the user can locate the desired records.
In contrast to conventional ISAM files, a file with secondary keys may contain neither value flags nor logical flags. However, the secondary keys mean that these extra features of the ISAM key are no longer required in any case.
The length of the primary key must not exceed 255 bytes. The length of each secondary key must not exceed 127 bytes.
The secondary keys may overlap each other and may overlap the primary key. A record with secondary keys could, therefore, have the following structure:

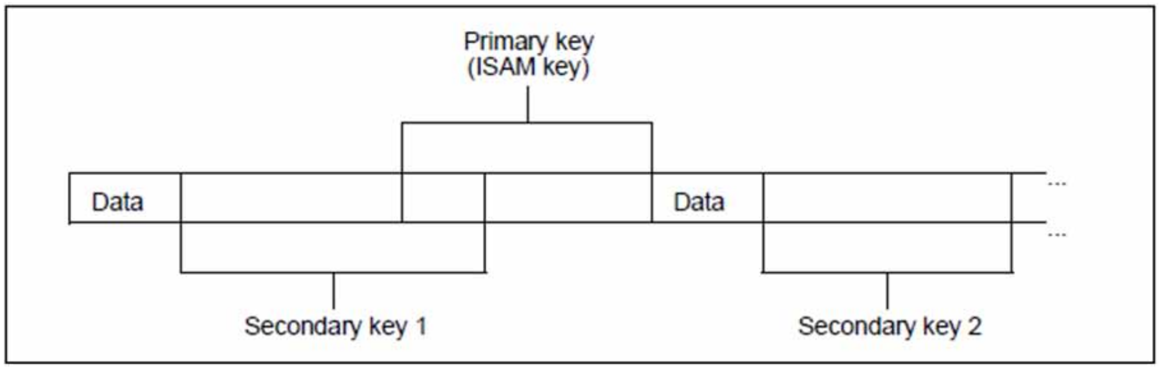


Figure 29: ISAM record with secondary keys

Primary key (ISAM key)

Each record in an ISAM file must have a primary key with a length of at least one byte. The position of the primary key within the record is defined for all records of the file with the KEYPOS operand of the FILE and FCB macros or via the KEY-POSITION operand of the ADD-FILE-LINK command. The user must take the record format (see “ISAM recordformats”) into account when defining the key position. In files with variablelength records, for example, the ISAM key must not start before position 5 (KEYPOS >= 5 or KEY-POSITION >= 5), since the first four bytes of each record are used for the record length field and control information.

The key length is defined via the KEYLEN operand in the FILE/FCB macro or via the KEY-LENGTH operand in the ADD-FILE-LINK command. The maximum permissible key length is 255 bytes. Logical flags and value flags are not permitted.

By means of key compression, DMS optimizes the number of index entries in each index block, so that longer keys are, on average, compressed to a length of 3-5 bytes and each index block can hold, on average, about 160 entries.

The number of index entries in an index block affects, in turn, the total number of index blocks needed, and thus the number of index levels in a file and the performance when accessing a record.

Secondary keys

In addition to the primary key, a record may contain up to 30 secondary keys. Just like the primary key, each secondary key has the same length and the same position in each record of the file.

In contrast to the primary key, the attributes of a secondary key (position, length, whether duplicate keys are permitted) are not kept in the catalog entry for the file, but in its control block. For this reason, they cannot be displayed by means of the FSTAT macro or the SHOW-FILE-ATTRIBUTES command. In order to enable the user to obtain information on the secondary keys of a file, ISAM therefore now provides the SHOWAIX macro and the SHOW-INDEX-ATTRIBUTES command.

A further difference between the primary key and a secondary key is that the secondary key is identified by a name. This is necessary because there may be several secondary keys defined for one file (whereas there is only ever one primary key).

Secondary keys can be defined only for an existing file, i.e. for a file which has been opened at least once with OUTPUT or OUTIN. The file may be empty, but users are advised, for reasons of performance, not to set up secondary keys for a new file until it has been filled with records.

The user can define a secondary key for a file via the CREAIX macro or the CREATE-ALTERNATE-INDEX command.

The position of the secondary key is specified via the KEYPOS operand of the macro or the KEY-POSITION operand of the command. Just as for the primary key, the 4 bytes for the record length field and control information must be taken into account in the case of files with variable-length records (see “Primary key (ISAM key)”).

The length of the secondary key is specified via the KEYLEN operand of the macro or the KEY-LENGTH operand of the command. A secondary key may be up to 127 bytes long. Both the number of secondary keys and their lengths affect the file size and the performance: since the entire secondary key is placed (without compression) in the secondary index block, the key length determines how many index entries will fit into one secondary index block.

The KEYNAME operand of the macro and the KEY-NAME operand of the command assign a name to the secondary key. The user can then specify this name in the macros GET, GETR, GETKY and SETL to refer to the secondary key to be used (in a manner similar to that for the primary key) for read and pointer operations. This name can also be specified in the DELAIX macro or the DELETE-ALTERNATE-INDEX command to identify a secondary key which is to be deleted. The user can obtain information about all secondary keys defined for a file with the aid of the SHOWAIX macro or the SHOW-INDEX-ATTRI-BUTES command.

Duplicate secondary keys

In files with secondary keys, the values of the primary key must be unique. In contrast, there may be duplicate secondary keys unless the user precludes this by specifying
DUPEKY=NO in the CREAIX macro or DUPLICATE-KEY=*NO in the CREATE-ALTERNATE-INDEX command.

Internally, DMS creates one or more secondary index blocks for each secondary key defined for a file. In this index, it creates an entry for each value of this secondary key which occurs in the file. This entry points to the primary key of the corresponding record. (For further details, see “Secondary index block (NK-ISAM)” (Primary index block (ISAM index block) and secondary index block)). If a given secondary key value occurs in several records, then the associated entry in the secondary index block also contains several pointers to the primary key values of these records. These pointers are stored in the order in which they are created, i.e. in the order in which they were written to or modified in the secondary index block. (For reasons of internal management, DMS appends a time stamp to each such pointer to a primary key when it is created.)

Sequential read operations via a secondary key thus return the records in the order in which the pointers to the associated primary key values were created, i.e. in the order in which they were written to or modified in the secondary index block. For “reverse read” operations, the records are returned in the reverse of this order.

Non-sequential read operations via a secondary key always return the first record of the group with identical secondary key values. The remaining records of the group must then be read sequentially.

Flags in the ISAM index

Neither value flags not logical flags are permitted in files with secondary keys.

ISAM record formats

With respect to record formats, the notes outlined for records without secondary keys (see "Record without a secondary key") also apply to records with secondary keys.

If secondary keys are subsequently defined for a non-empty file with RECFORM=V or RECORD-FORMAT=*VARIABLE, it must be remembered, when specifying the key position and key length, that the shortest record in the file must contain the entire secondary key being defined.

21.1.4 ISAM data block

The records of an ISAM file are stored in data blocks, sorted according to the order of their ISAM keys (primary keys). Each of these blocks may consist of 1-16 contiguous PAM pages; a data block consisting of more then one PAM page is also called a multiblock.

The length of a data block is defined via the BLKSIZE operand in the FILE/FCB macro or the BUFFER-LENGTH operand of the ADD-FILE-LINK command. Since ISAM files are disk files, only a blocking factor “n” for standard blocks (1 <= n <= 16) can be specified, i.e. an absolute length cannot be specified. When selecting a block size, due attention must be paid to its compatibility with the record length to ensure that optimum use is made of the storage space and that overflow blocks are avoided. Section section “Overflow block (NK-ISAM)” contains tables which show this relationship.

Block splitting

If, in the case of non-sequential insertion of a record (STORE, INSRT), there is not enough space in the appropriate block to allow insertion, block splitting is necessary: the old block is split and the resulting halves are transferred to new (empty) blocks. The old block still belongs to the file and is marked as a free block.

Example: block splitting

File structure

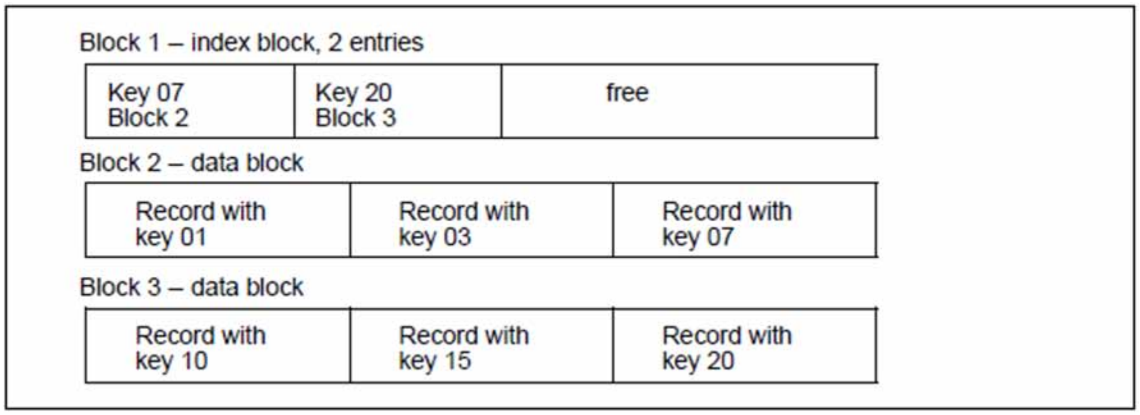


Figure 30: Example of an ISAM file structure before a record is inserted

The user specifies that a record with key 05 is to be inserted. Since there is no space in block 2, in which this record is to be stored, block 2 is split.

The file now has the following structure:

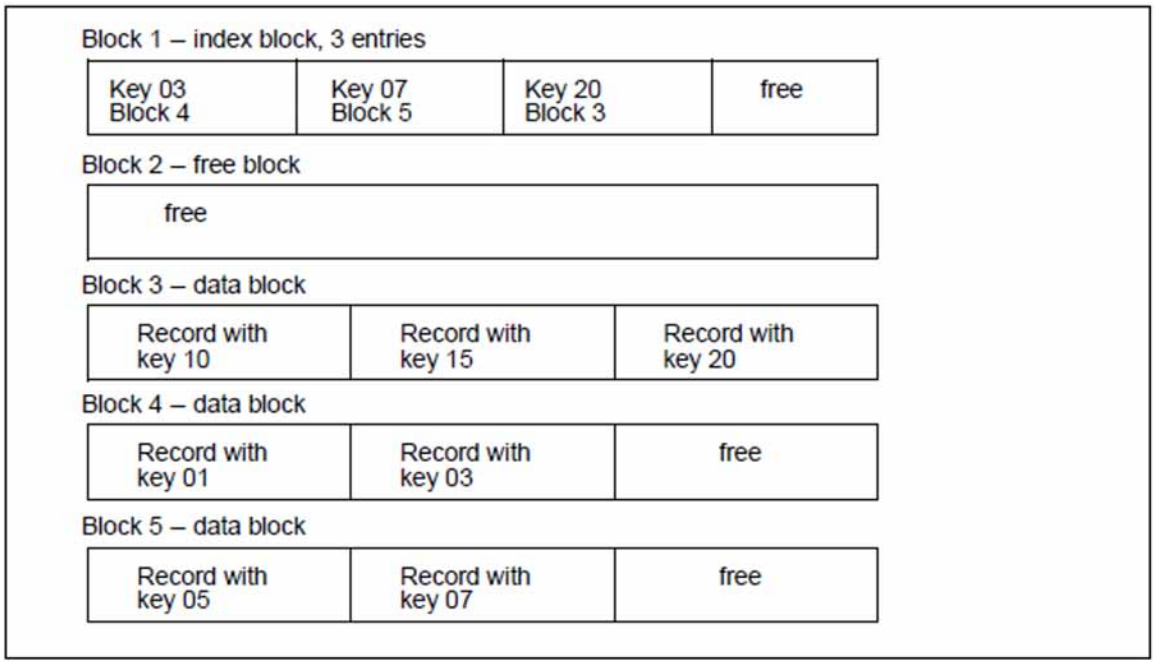


Figure 31: Example of an ISAM file structure after a record has been inserted

21.1.5 Overflow block (NK-ISAM)

Overflow blocks are extensions of data blocks. They are created if records longer than the usable length of a data block are written. This occurs, for example, if the user uses records with the same keys and, when defining the record and block lengths, forgets that NK-ISAM internally appends an 8-byte time stamp to such records. Overflow blocks also result when converting K-ISAM files into NK-ISAM files if the maximum record length was utilized in the K-ISAM file. The data which “overflows” is always placed at the beginning of an overflow block, and the end of this block contains a pointer to the associated data block. The data area of the overflow block must not contain any part of the ISAM key (primary key), of the flags or of a secondary key.

Maximum record length / Usable block length

No overflow blocks will be created if the following rules are observed:

- the maximum permissible record length is determined by the BLKSIZE or BUFFER-LENGTH specification: n * 2048, where 1 <= n <= 16 (for BLKSIZE=(STD,n) or BUFFER-LENGTH=*STD(SIZE=n))
- the usable block size is the block size minus the space needed for management information and the time stamp
- as ISAM places a 4-byte record length field in front of each record, the usable block size for records with RECFORM=F or RECORD-FORMAT=*FIXED is reduced by a further 4 bytes

This results in the following formula for the usable block size:

```
RECSIZE <= n * 2032 - 16 - T - F
```

n	blocking factor in BLKSIZE = (STD,n), where 1 <= n <= 16 or BUFFER-LENGTH=*STD(SIZE=n)
T	time stamp • T = 8 for files with duplicate keys • T = 0 for files without duplicate keys
F	record format • F = 0 bei RECFORM = V or RECORD-FORMAT=*VARIABLE • F = 4 bei RECFORM = F or RECORD-FORMAT=*FIXED

As each data block contains at least one record, a further 16 bytes for the management information must be taken into account when defining the record length. The table below shows the relationship between the maximum permissible record length and the usable block size for NK-ISAM files without overflow blocks for ISAM files with standard attributes (record format V, no duplicate keys).

Blocking factor n (BLKSIZE=(STD,n))	Maximum record length = BLKSIZE (in Byte)	Usable block length (in Byte)
1	2048	2016
2	4096	4048
3	6144	6080

4	8192	8112
5	10240	10144
6	12288	12176
7	14336	14208
8	16384	16240
9	18432	18272
10	20480	20304
11	22528	22336
12	24576	24368
13	26624	26400
14	28672	28432
15	30720	30464
16	32768	32496

Table 49: Usable block size for NK-ISAM files

Maximum key position and length

As neither the ISAM key nor the flags may extend into an overflow block, there are certain maximum values for key position (KEYPOS), key length (KEYLEN), value flag length (VALLEN) and logical flag length (LOGLEN). These values depend on the blocking factor and the record format. The permissible maximum values can be calculated from the usable block size (see table 49) with the aid of the following formulas:

ISAM key length and position:

```
    KEYPOS <= SL - F - KEYLEN - VALLEN - LOGLEN + 1

    KEYLEN <= SL - F - KEYPOS - VALLEN - LOGLEN + 1
```

KEYPOS	key position
KEYLEN	Key length
VALLEN	value flag (length)
LOGLEN	logical flag (length)
SL	Usable block length

F	contingent on the record format: • F = 0 for RECFORM=V • F = 4 for RECFORM=F

Example

File attributes:	BLKSIZE = (STD,1)
	KEYLEN = 12

The values for KEYPOS are as follows:

```
RECFORM = V => KEYPOS <= 2016 - 12 + 1 = 2005
RECFORM = F => KEYPOS <= 2016 - 4 - 12 + 1 = 2001
```

21.1.6 NK4 format

In order to support NK4 volumes, a new format called the NK4-ISAM block format has been introduced:

A logical block consists of a sequence of 4K blocks, which correspond to the transport unit for NK4 volumes. Each 4K block begins with a block control field which is made up of a general part comprising 12 bytes and an ISAM-specific part of 4 bytes.

A data block consists of a sequence of at least one and at most eight 4K blocks. The size of the logical data block is controlled by a specification in the FCB or FILE macro or in the ADD-FILE-LINK command. The layout of an NK4-ISAM block corresponds to that of an NK2-ISAM block. The addressing of records and the management of free space are handled via the block trailer.

Maximum record length / Usable block length

The total length of an NK-4K data block cannot be fully used for the user's records, since 16 bytes are used for the block control field by the system for each 4K page of the block, and a further 16 bytes per block are reserved for the block trailer. The effectively usable length of an NK-4K block of a file with block size (STD,n), where n is even, is thus as follows:

- The maximum permissible record length is determined by the specified block size
  (in the BUFFER-LENGTH operand of the ADD-FILE-LINK command or the BLKSIZE operand of the FILE macro): n * 2048, n = 2, 4, 6, 8
- The usable block size is the block size minus the space needed for management information and the time stamp.
  (16 bytes are used for the block control field by the system for each 4K page of the logical block, and a further 16 bytes per block are reserved for the block trailer.)
- Since ISAM places a 4-byte record length field in front of each record, the usable block size for records with RECFORM=F is reduced by a further 4 bytes.
  (Each data block must contain at least one record.)

Since the maximum record length corresponds to the logical block size given above rather than the effective block length (so as to ensure compatibility with the “K” formats), the length of a record may exceed the effective block length. An overflow block associated with the data block is created in such cases. The part of the record that could not be accommodated in the data block is stored in it (the size of the overflow block is 4K). The position and length of a key are subject to the same restrictions as for NK2-ISAM format, i.e. the key may not be fully or partially located in the overflow block in either NK4-ISAM or NK2-ISAM format.

Index data blocks consist of a minimum of one and a maximum of eight 4K blocks, with a block size that corresponds to that of the data block.

Index blocks are made up of exactly one 4K block. The index information is structured at two levels: in sections and entries, with the length of a section adjusted to the new index block size of 4K.

The control block is the first block of an ISAM file. It is made up of exactly one 4K block. It has the same layout as in the 2K format. The space in excess of 2K is not used.

Free blocks resulting from data or index data blocks are made up of a minimum of one and a maximum of eight 4K blocks. Free blocks obtained from index or overflow blocks consist of exactly one 4K block.

An NK4-ISAM file must be created with an even blocking factor. This can be done as follows:

- automatically, if the file is located on an NK4 disk (NK4 PVS)

- by an explicit specification from the user (in the operand BLOCK-CONTROL-INFO= WITHIN-DATA-4K-BLOCK of the ADD-FILE-LINK command or the operand BLKCTRL=DATA4K of the FILE/FCB macro)

For more information on the buffering of NK4-ISAM files in ISAM pools, see section “NK4-ISAM files in ISAM pools”.

21.1.7 Primary index block (ISAM index block) and secondary index block

Primary index block

From each of the data blocks described in section "ISAM data block", DMS takes the highest key value and places it, together with the logical block number of the data block, in a primary index block. In NK-ISAM, a "separator" is formed from this key; in K-ISAM, any existing value and/or logical flags are also included in the index entry.

Each primary index block consists of precisely one PAM page. If one primary index block is too small to hold the pointers to all existing data blocks, not only a second primary index block at the same level is created, but also a primary index block at a second level and the entries in this block point to the primary index blocks at the first level. This results in an index tree.

Primary index tree

The tree structure of an ISAM file is formed by the primary index and data blocks. The data blocks form the lowest level, which is called level 0. All higher levels contain only primary index blocks. The highest level always consists of a single primary index block, which is called the root of the index tree and is the entry point for all accesses to the records of the ISAM file.

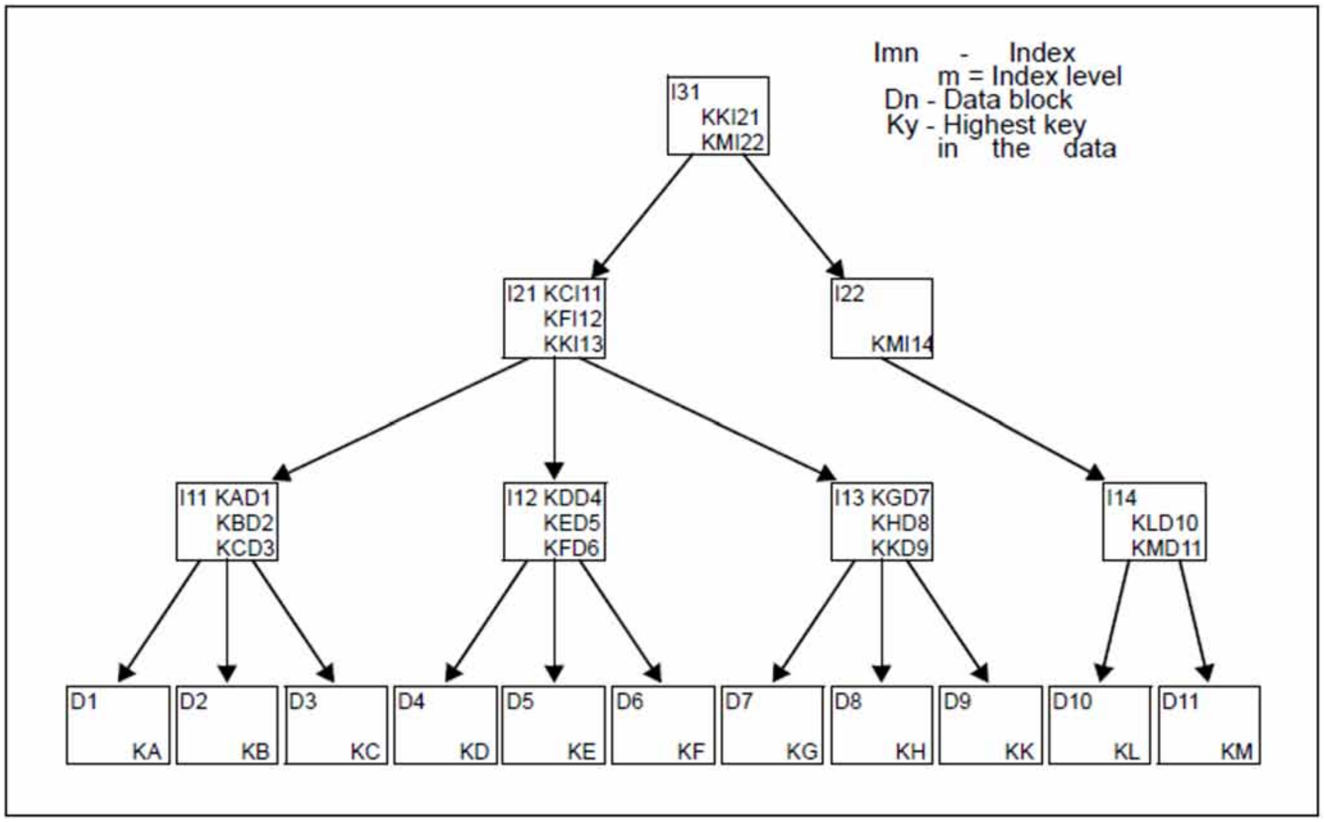


Figure 32: Primary index tree

The file shown in this example has three primary index levels (I1, I2, I3) and the data level (D). The root of the primary index tree is primary index block I31. The letters Ky represent the highest key in each data block. These keys are placed, together with pointers to the associated data blocks, in the primary index blocks of level I1. The highest key in each primary index block I1n is placed, this time with a pointer to the associated primary index block, in a primary index block of level I2. Since one primary index block is still not enough to accommodate all index entries at this level, there has to be a third index level (I3), which consists of a single primary index block, namely the root of the tree.

Secondary index block (NK-ISAM)

For each secondary key defined for a file, DMS creates a secondary index consisting of one or more secondary index blocks. In this index, it creates an entry for each value of this secondary key which occurs in the file. This entry contains the primary key values of the associated record as a pointer. Entries with the same secondary key value are combined to form one record. If such a record would exceed the boundaries of a secondary index block, another record is created for the same secondary key value.

An entry in a secondary index consists of:

- the secondary key value with the largest time stamp contained in the record. If the record involved is the last record with the secondary key value, it contains the value X'FF...FF' instead of the time stamp.
- one or more entries: each entry consists of the primary key value of the record involved and a time stamp. If duplicate keys are not permitted for the secondary key (operand DUPEKY=NO in the CREAIX macro or DUPLICATE-KEY=*NO in the CREATE-ALTERNATE-INDEX command), each entry contains only one pointer, and this pointer's time stamp contains the maximum possible value (X'FF...FF').

A record in a secondary index block for a given secondary key value thus has the following structure:

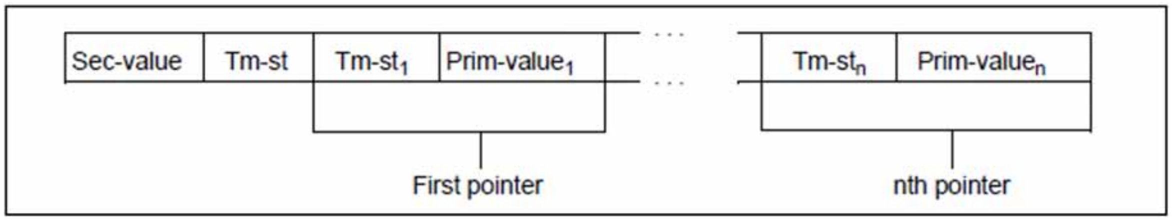


Figure 33: Structure of an entry in a secondary index block

where:

Sec-value	Secondary key value for which this record was created.
Tm-St	Time stamp of the record.
Tm-St$_i$	i=1,...,n: time stamp of entry i.
Prim-value$_i$	i=1,...,n: primary key value of the record to which pointer i points

Access to records in an ISAM file

Access via the primary key (ISAM key)

If an ISAM file is processed randomly (i.e. not sequentially), DMS must be able to retrieve any record requested by the user program. It starts the search for a record with the specified key at the highest primary index level – in the root of the primary index tree – and is then guided downwards through all existing primary index levels to the appropriate data block. In NK-ISAM, the search within a primary index block is speeded up by grouping the primary index entries into “sections”, which means that the search is executed in a series of jumps.

In the example shown in figure 32, let us assume the user wants the record with the key KI. The primary index entries in I31 show that the search must be continued in primary index block I21: the key KI is lower than KK.
The entries in primary index block I21 point to primary index block I13 for KI: KF < KI < KK.
The entries in primary index block I21 point to primary index block I13 for KI: KH < KI < KK.
The desired record can now be found quickly.

In both NK-ISAM and K-ISAM, the access paths are implemented in the form of a tree structure. However, the primary index block at the highest level in K-ISAM contains a 36-byte header with management information, while this information is kept in the control block in NK-ISAM.

The use of key compression in the primary index blocks means that the length of a primary index entry in NK-ISAM is virtually independent of the actual key length. The number of index entries and index levels thus depends only on the number of data blocks.

In K-ISAM, the number of primary index blocks and primary index levels of an ISAM file depends not only on the number of data blocks in the file, but also on the length of the ISAM key or, if value and/or logical flags are used, on the length of the ISAM primary index: no key compression is used; if the primary index area is 255 bytes long only 4 index entries will fit into one primary index block.

Access via a secondary key (NK-ISAM)

If the user requests a record with the aid of a secondary key, DMS first searches the associated secondary index blocks for an record with this value. If such a record exists, DMS uses the entries in the record to determine the primary key value of the desired record. If there are several entries in the record (which happens in the case of duplicate secondary keys), DMS always uses the first pointer. With this primary key value, DMS then searches for the desired record as described in the preceding section.

Searching for a record with the aid of a secondary key is thus less efficient than retrieving the record via its primary key: approximately twice the number of I/O operations and twice the CPU time are required.

21.1.8 Free blocks

Free blocks do not contain data. However, they have already been used at least once by ISAM and continue to be managed by ISAM. They result from block splitting operations or from deletion of records from the file.

i Free blocks must not be confused with “unused blocks”: the free blocks belong to the “area in use” of an ISAM file and the space they occupy cannot be released by means of FILE (operand SPACE) or MODIFY-FILE-ATTRIBUTES; “unused blocks” form the unused part of the ISAM file and the space they occupy can be released at any time.

If new data, index or overflow blocks are needed, any existing free blocks are used before the unused blocks are used.

21.1.9 Control block (NK-ISAM)

The first block of an NK-ISAM file (LBN=1) is always the control block. It contains management information, e.g. about the file size, the area in use, the free blocks, etc. The control block should not be confused with the root of the index tree, the highest-level index block which forms the entry point for searching via record keys.

21.1.10 Calculating the size of an ISAM file

The size of a file can be calculated in the same way for both NK-ISAM and K-ISAM. First, the number of data blocks is calculated from the record length, the usable block size and the number of records.

Usable block size:

```
    RECSIZE <= (n * 2048 (100 - PAD) / 100)

        ==> usable block size = n * 2048 (100 - PAD) / 100

    (n * 2048 (100 - PAD) / 100) <= RECSIZE <= n * 2048

        ==> usable block size = RECSIZE
```

n	blocking factor
PAD	PAD value (see PAD operand of the FILE macro)
RECSIZE	Record length

Size of the data section:

```
    Number of records per data block = (Usable block size) / RECSIZE

    Number of data blocks = (Records per file / Records per data block)
```

For NK-ISAM, the number of overflow blocks must also be estimated. This is dependent on the record length:

Overflow blocks:

```
    Number of overflow blocks = i * Number of data blocks
```

i = 0	for RECSIZE <= n * 2032 - 16
i = 1	for RECSIZE > n * 2032 - 16

In sequentially created NK-ISAM files, each primary index block can hold an average of about 200 primary index entries. The size of an NK-ISAM file can thus be estimated as follows:

Size of file:

```
    Number of PAM pages = Number of data blocks * (BLKSIZE + i + 0.005)
```

i = 1 / i= 0 takes account of overflow blocks (see above).

For NK-ISAM files, the total number of primary index blocks in a file can be considered to be approximately the same as the number of primary index blocks in the lowest primary index level. The following tables simplify the estimation of the sizes of NK-ISAM files.

For NK-ISAM files created sequentially, each primary index block holds about 200 primary index entries (due to key compression); this results in the values shown in the following table:

Number of data blocks	Number of primary index levels
0 - 200	1
200 - 40000	2
40000 - 8000000	3

Table 50: Calculation of primary index levels for sequentially created NK-ISAM files

For files created non-sequentially, the number of primary index levels increases faster than for sequentially created files, due to the block splitting involved. If there is more than one primary index level, each primary index block contains an average of about 160 index entries.

Number of data blocks	Number of primary index levels
0 - 200	1
200 - 32000	2
32000 - 5120000	3

Table 51: Calculation of primary index levels for non-sequentially created NK-ISAM files

For a file with secondary keys, the number of secondary index blocks can be estimated with the aid of the following formula. This formula takes into account the fact that an attempt is made to fill the secondary index blocks to a level of 75%. This percentage is only an average value. In some cases, this figure cannot be achieved.

Secondary index blocks for secondary key i:

$$\text{Number of PAM pages} = \text{Number of data blocks} * (\text{P-KEYLEN} + 8 + h_i * (\text{S-KEYLEN}_i + 8)) / 1512$$

PAM pages$_i$	PAM pages for the secondary index blocks for secondary key i.
P-KEYLEN	Length of the primary key of the file.
h_i	Relative frequency with which different values occur in the file for secondary key i: if, for example, there are on average 10 records with the same value in secondary key i, then $h_i = 0.1$
S-KEYLEN$_i$	Length of secondary key i.

The total number of secondary index blocks for all secondary keys in the file is then the sum of the values determined for all defined secondary keys:

Secondary index blocks for a file with n secondary keys:

```
Number of PAM pages = Number of PAM pages + . . . + Number of PAM pages n
```

K-ISAM

For K-ISAM files, the number of primary index entries per index block depends on the key length:

```
Number of index entries per primary index block = 1024 / (KEYLEN + 4)
```

This formula takes into account the fact that a primary index block cannot be more than 50% full. The total size of a K-ISAM file is then calculated as follows:

Size of file:

```
Number of PAM pages = Number of data blocks * (BLKSIZE + ((KEYLEN + 4) /
1024))
```

KEYLEN: Key length.

21.2 ISAM pools

Particularly the non-sequential processing of ISAM files tends to produce a high I/O rate. NK-ISAM reduces this I/O rate and enhances the performance significantly by processing the files in appropriately sized ISAM pools which are used to buffer file blocks.

i K-ISAM does not use such pools. When "ISAM" is used in the description below, NK-ISAM is always meant, never K-ISAM.

ISAM pools can be created and managed both explicitly by the user using special macros and implicitly by DMS.

- Pools opened implicitly by DMS are called **standard ISAM pools** (see section “Standard ISAM pools”).
- Pools created by the user are called **user ISAM pools** (see section “User ISAM pools”).

In addition, two areas of application are distinguished for pools:

- A **task-local** ISAM pool (SCOPE=*TASK) is stored in the class 5 storage of the owner task. No other tasks may access this storage - or the ISAM pool. A task-local ISAM pool is consequently only suitable for ISAM files which are opened in SHARUPD=NO mode.
- A **cross-task** ISAM pool (SCOPE=*HOST-SYSTEM) is stored in a privileged data space which all tasks in the system can access. Details on this enhanced concept are provided in section “ISAM pools in data spaces”.A cross-task pool is essential for processing ISAM files in SHARUPD=YES mode. It would be possible to edit ISAM files opened in SHARUPD=NO mode in cross-task pools, too. However, for performance reasons (e.g. superfluous serialization) this is not recommended.

SCOPE=*USER-GROUP and SCOPE=*USER-ID are only accepted for reasons of compatibility. Internally, however, they are mapped to SCOPE=*HOST-SYSTEM (cross-task pool).

The commands and macros for managing task-local ISAM pools are also available to RFA (see the "RFA" manual [6 (Related publications)]).

An overview of the ISAM pool macros can be found in section “User ISAM pools”.

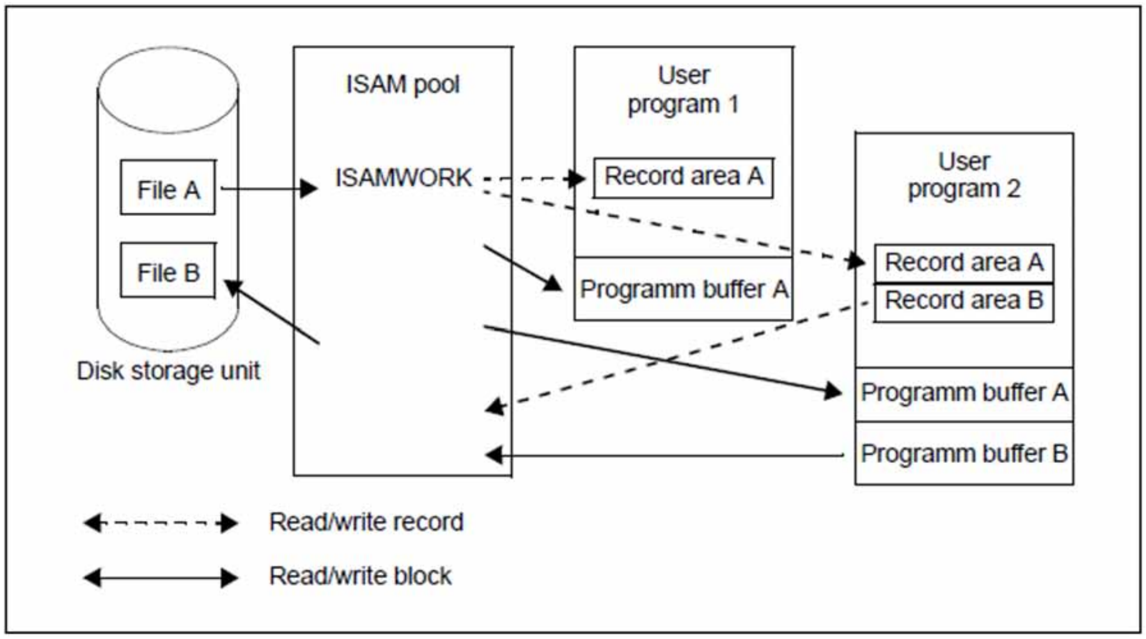


Figure 34: ISAM pool

File A is being read by user programs PROG A and PROG B, and file B is being created by PROG B. The ISAMWORK ISAM pool is a cross-task pool.

21.2.1 Functions of the ISAM pool

ISAM pools are used as buffers for one or more NK-ISAM files. In addition to the buffers required for the individual PAM pages, they contain management data which describes, amongst other things, the files being buffered, the jobs connected to the pool, and the accesses to the individual buffers.

If, during ISAM processing, a block of an ISAM file is needed, the ISAM pool in which this file is buffered is searched first. If the block is not found in the pool, it has to be read from the disk. If all buffers of the pool are full, one of the blocks which is already in the buffer is overwritten; it is selected according to the following criteria: data blocks and overflow blocks are overwritten before index blocks; within a block type, the LRU (Least Recently Used) block is overwritten.

The handling of updated blocks can be defined for ISAM pools (and also for individual files) by using operands. Depending on the selected processing attributes, updated blocks are written back to disk either immediately or only when the buffer is needed to store another block. (For “immediate write”, see "WROUT function").

The following table provides an overview of macros and commands used to process tasklocal ISAM pools and to request information about these.

Macro	Command	Brief description
ADDPLNK	ADD-ISAM-POOL-LINK	Assign a pool link name to an ISAM pool (task-specific).
CREPOOL	CREATE-ISAM-POOL	Create a new ISAM pool or set up a link between the calling task and an existing ISAM pool.
DELPOOL	DELETE-ISAM-POOL	Remove the link between the calling task and an ISAM pool. If the caller is the last or only task with a link to the ISAM pool, the ISAM pool itself is deleted as well. This macro can also be used to remove all links of the calling task to ISAM pools.
REMPLNK	REMOVE-ISAM-POOL-LINK	Cancel the assignment of a pool link name to an ISAM pool. In this case (as in DELPOOL/DELETE-ISAM-POOL), a special parameter may be specified to remove all (task-specific) assignments of pool link names to ISAM pools by means of a single call.
SHOPOOL	SHOW-ISAM-POOL-ATTRIBUTES	Return information on an ISAM pool to which the caller is connected at the time of the call. It is also possible to display information on all ISAM pools currently connected to the calling task.
SHOPLNK	SHOW-ISAM-POOL-LINK	Show the assignments of pool link names to ISAM pools.

Table 52: ISAM pool macros and commands

21.2.2 Standard ISAM pools

If an ISAM file is opened without the user having assigned it to a specific ISAM pool in the FILE/FCB macro (POOLLNK operand) or in the ADD-FILE-LINK command (POOL-LINK operand), DMS uses a system standard ISAM pool for processing the file: in the case of an OPEN with SHARUPD=NO, a task-local standard ISAM pool is allocated, and in the case of an OPEN with SHARUPD=YES a cross-task standard ISAM pool.

A **task-local** ISAM pool is characterized by the fact that it can be used by only one task. When an ISAM file is opened for the first time with SHARUPD=NO, the DMS creates the ISAM pool $TASK01. This pool is also used for further ISAM file processing in the same task, provided there is sufficient space for each file. Otherwise the pool $TASK02 is created when the next ISAM file is opened, and so on.

Up to 16 ISAM pools ($TASKn, 01 <= n <= 16) can be created for each job. Systems support defines the size of the pools using the ISAM parameter LCLDFPS.

A **cross-task** ISAM pool can be used by all tasks in the system. The first time an ISAM file is opened with SHARUPD=YES, DMS creates such a pool and as a rule only uses it for this file. If sufficient space is available, a separate pool is created for each further file which is opened with SHARUPD=YES.

In contrast to task-local pools, cross-task pools do not have a name. They are implicitly linked to the file they contain. Systems support defines their size using the ISAM parameter GLBPS and their maximum number indirectly using the ISAM parameter MAXDSBN. This parameter, which can be modified during ongoing operation using the MODIFY-ISAM-CACHING command, specifies how many data spaces DMS may use to accommodate cross-task ISAM pools. The maximum number of pools which may be generated is calculated from the fixed maximum size of a data space (2 GB) and the size of the pool.

i The concept of cross-task standard ISAM pools $SYS01 through $SYS16 which are stored in memory pools has not existed since BS2000/OSD-BC V6.0B.

One job may simultaneously process several ISAM files which can be allocated to different standard pools.

When an ISAM file which was processed using a standard pool is closed, DMS initially checks whether the job is still linked to this pool via other files. If this is not the case, it releases the resources reserved by this pool.

The pages of a cross-task ISAM pool in a data space which contain user data are not output in memory dumps. Only the management data of such pools which is required for diagnostics appears in any memory dumps which may need to be created.

21.2.3 User ISAM pools

A user can create and manage **task-local**ISAM pools with the aid of macros or commands.

Such a pool will, however, only be used for file processing if a pool link name is assigned to it by means of the ADDPLNK macro or ADD-POOL-LINK command and the POOLLNK or POOL-LINK operand is used to set up a connection between the ISAM pool and the file.

Pool link name – the link between file and ISAM pool

The ADDPLNK macro and the ADD-POOL-LINK command serve to assign an ISAM pool a pool link name in a job and to enter the name in the job's pool table. This pool link name then has to be entered in the TFT for each file that is to be processed in this pool (using the POOLLNK or POOL-LINK operand). In addition, NK-ISAM processing must be set by means of BLKCTRL=DATA or BLOCK-CONTROL-INFO=WITHIN-DATA-BLOCK.

When a file is opened, DMS checks whether a pool link name has been entered in the FCB or TFT, and (via the pool table) whether an ISAM pool exists to which the pool link name refers. If both checks are positive and if the pool and file attributes are compatible, the file is processed via this ISAM pool.

Comments on cross-task user ISAM pools

User ISAM pools are always **task-local**.

Commands and macros for cross-task user ISAM pools are still accepted for reasons of compatibility, but no longer result in such a pool being generated. Instead, DMS uses a standard ISAM pool in which the size specified beforehand by the user is not fallen below and the assignment of files to pools is retained.

The following diagram shows the actions which must be initiated by a job working with a user ISAM pool at the program interface.

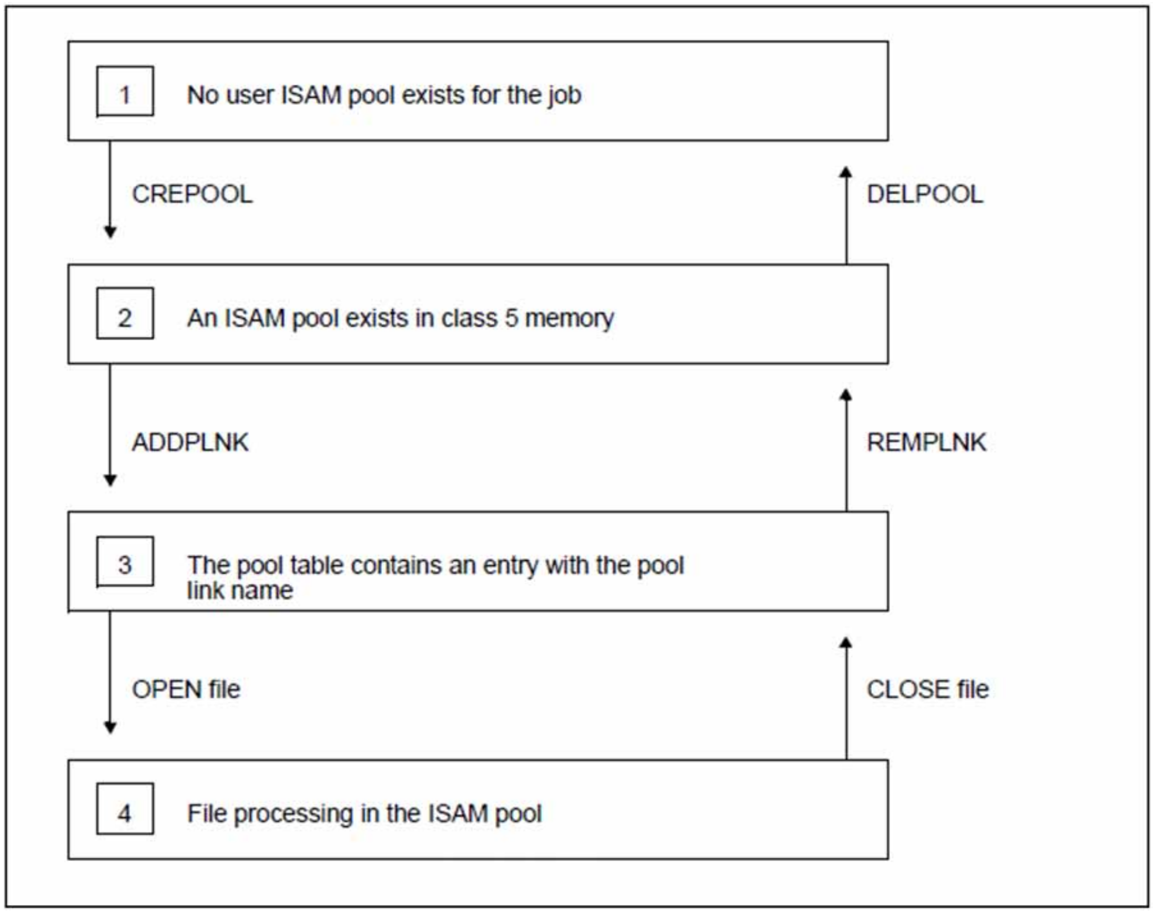

1 No user ISAM pool exists for the job
CREPOOL
DELPOOL
2 An ISAM pool exists in class 5 memory
ADDPLNK
REMPLNK
3 The pool table contains an entry with the pool link name
OPEN file
CLOSE file
4 File processing in the ISAM pool

Figure 35: Actions for file processing with ISAM pools (macros)

The following table shows the macros available to the user.

Macro	**Function**	**Action see figure 35**
CREPOOL	Create an ISAM pool. for a cross-task pool, the system checks if the pool already exists.	1 -> 2
ADDPLNK	Define the pool link name: this name is placed in the pool table.	2 -> 3

FILE/FCB
LINK=name
POOLLNK=name
BLKCTRL=DATA
	At OPEN time:
	set up the link between the ISAM pool and the file.
		3 -> 4
		(OPEN)
		4 -> 3
		(CLOSE)

REMPLNK	Delete the pool link name entry from the pool table.	3-> 2
DELPOOL	“Delete” the ISAM pool: release space in class 5 memory or clear the link to the cross-task pool if other jobs are still using the pool.	2 -> 1

Table 53: Overview of the ISAM pool macros

The following diagram shows the actions which must be initiated by a job working with a user ISAM pool at the command interface:

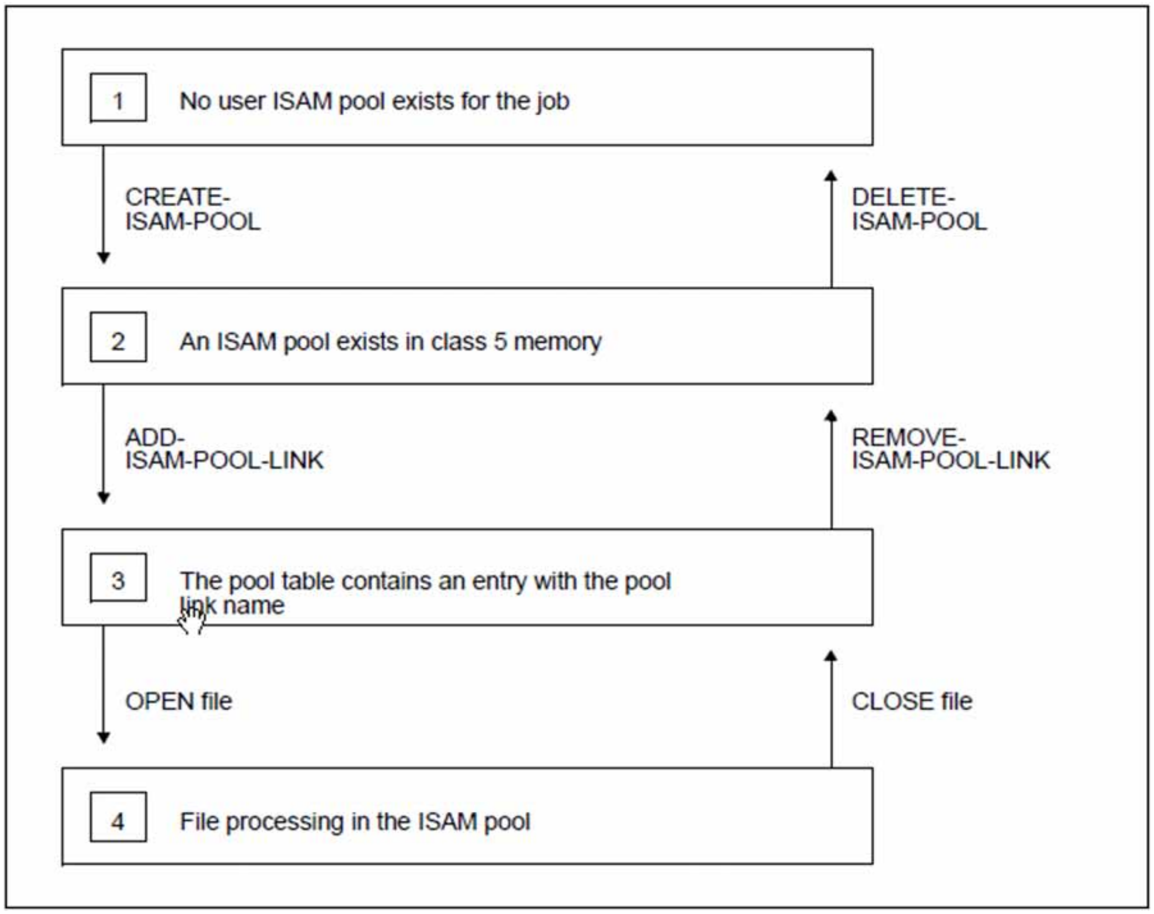

1
No user ISAM pool exists for the job
CREATE-
ISAM-POOL
DELETE-
ISAM-POOL
2
An ISAM pool exists in class 5 memory
ADD-
ISAM-POOL-LINK
REMOVE-
ISAM-POOL-LINK
3
The pool table contains an entry with the pool link name
OPEN file
CLOSE file
4
File processing in the ISAM pool

Figure 36: Actions for file processing with ISAM pools (commands)

The following table shows the commands available to the user.

Command	**Function**	**Action see figure 36**
CREATE-ISAM-POOL	Create an ISAM pool. for a cross-task pool, the system checks if the pool already exists.	1 -> 2
ADD-ISAM-POOL-LINK	Define the pool link name: this name is placed in the pool table.	2 -> 3
ADD-FILE-LINK LINK-NAME=name POOL-LINK=name BLOCK-CONTR-INFO= *WITHIN-DATA-BLOCK	At OPEN time: set up the link between the ISAM pool and the file.	3 -> 4 (OPEN) 4 -> 3 (CLOSE)
REMOVE-ISAM-POOL-LINK	Delete the pool link name entry from the pool table.	3-> 2

DELETE-ISAM-POOL	“Delete” the ISAM pool: release space in class 5 memory or clear the link to the cross-task pool if other jobs are still using the pool.	2 -> 1

Table 54: Overview of the ISAM pool commands

21.2.4 NK4-ISAM files in ISAM pools

To buffer NK4 files, ISAM pools are now also created in units of 4K blocks and not in units of 2K blocks as before. If both NK2 and NK4 files are buffered in an ISAM pool, the ISAM pool will consist of two extents. One to accept 2K blocks and one for 4K blocks. When the ISAM pool is created, the first extent is made available unformatted and as soon as the first file that is buffered via the pool is opened, the extent is created as required for the file (2K or 4K). A second extent is created if a file with some other block control attribute is to be opened via the same ISAM pool. The size of the second extent is also determined by the value specified for the block size.

NK-2K files and NK-4K files should not be processed via the same ISAM pool; otherwise, optimizing the pool size will require double the amount of space, since separate 2K and 4K extents will need to be created in the specified size.

21.2.5 Calculating the size of an ISAM pool

The size of an ISAM pool has an effect on the performance of file processing. When optimizing/matching the pool size to the file processing to be performed, the following must be noted:

- How many files are to be processed in the ISAM pool?
- How are they to be processed: sequentially or non-sequentially?
- How large are the index and data areas and/or the data blocks?
- How many jobs may access the pool?

Calculating the size of a task-specific ISAM pool for a single file is described below. If several files are to be processed in the pool, this calculation should be repeated for each file and the results added together. If the ISAM pool is to be accessible to several jobs, the space required for each job should be calculated and these results added together. Files processed by several files at the same time (SHARUPD=YES) constitute a special case: here, the space required for the index area needs to be provided only once.

See section “Calculating the size of an ISAM file”, for details of how to calculate the sizes of the index and data sections of ISAM files.

Sequential file processing

For purely sequential file processing, the pool size has virtually no effect on performance: blocks which have been processed are not needed again, and the ISAM pool can be kept very small. It should be capable of simultaneously accommodating: the control block, the necessary index blocks, and two data blocks together with any required overflow blocks. The pool size can be calculated (approximately) with the formula:

Pool size:

```
    SIZE = 1.1 * (1 + i + 2m)
```

i = number of index levels in the file.
m = number of PAM pages per data block, including the overflow block; (m = n+1, n as specified in BLKSIZE=(STD,n)).

The factor 1.1 takes into account the fact that about 10% of the pool space is used for management data.

The total number of index blocks in an ISAM file is approximately the same as the number of index blocks in the lowest index level. This means, for non-sequential processing of an ISAM file, that the optimum pool size can be found by first calculating the space required for the index and then, since the pool must also hold a number of data blocks, rounding this up.

Number of index blocks in level 1:

```
    number = LASTPG / (n * 160)
```

LASTPG = the number of PAM pages in the file
n = the blocking factor (BLKSIZE=(STD,n))

The value 160 is the average number of index entries in an index block.

Example: pool size

For file ISAM.B: LASTPG = 18270; ISAM.B was defined with BLKSIZE=(STD,2) and contains no overflow blocks.

```
    LASTPG / n = 18270 / 2 = 9135
```

The file contains approximately 9135 data blocks.

```
    9135 / 160 > 57
```

The lowest index level occupies 57 index blocks. The optimum pool size for this file would be between 60 and 70 PAM pages: it would then accommodate all of the index and several data blocks.

Small files

Small files can in many cases be held completely in the ISAM pool.

Pool size for small ISAM files:

```
    SIZE  1.1 * LASTPG
```

LASTPG is the highest PAM page used by ISAM, and thus indicates the actual size of the file.

Index blocks in the ISAM pool

Ideally the ISAM pool should be able to accommodate all the index blocks. In addition, space for a data block should be provided for every job accessing the file. The space requirements can be calculated as follows:

Space requirements:

```
    SIZE >= 1.1 * (LASTPG / (160 * BLKSIZE) + #TASK * BLKSIZE)
```

#TASK is the number of jobs accessing the file concurrently and BLKSIZE is the blocking factor.

Minimum pool size

Minimum pool size:

$\mathtt{SIZE}_{\mathtt{MIN}}$ `= 1,1 * ((#INDEX -1 + BLKSIZE) * #TASK +1)`

In addition to the block size and the number of concurrent jobs, the number of index levels (#INDEX) in a file must also be taken into account. #INDEX can be estimated as follows:

Number of index levels:

```
    #INDEX = [ ln (LASTPG / BLKSIZE) / ln(160) ]
```

“ln” is the natural logarithm, while “[“ and “]” mean that the value of the entire expression should be rounded up to the next higher integer.

Pool-specific limit values

The number of files (#FILE) and the number of jobs with concurrent access (#TASK) are limited by the SIZE specification. They must comply with the following rule:

Limit values:

```
    #FILE + #TASK <= SIZE * 1.4
```

The actual size value used here is the internal, rounded value, not the value specified by the user.

Example

Pool size defined in a CREPOOL macro or CREATE-ISAM-POOL: SIZE = 32. The sum of the number of open files and the number of connected jobs must not exceed 44.

21.2.6 ISAM pools in data spaces

The creation of buffer areas for NK-ISAM files (ISAM pools) which are opened with SHARUPD=YES is automated and optimized. The explicit configuration of ISAM pools via the relevant program or command interfaces can thus be dispensed with.

ISAM pools are created on a file-specific basis the first time a file is opened. When a file is opened, a suitable section of a data space is initialized as an ISAM pool. In the course of this the size is determined automatically: A data space with a total size of max. 2 GB is split into equal-sized sections of 1 MB (or a 2^n multiple of this).

An ISAM pool consists of at least one such section. Depending on the file size an ISAM pool can also consist of several adjacent sections of this data space. The pool size is estimated using the formula on "Calculating the size of an ISAM pool".

It is also possible to define a size explicitly using the ADD-FILE-LINK ...,POOL-SIZE= command or via the relevant FILE program interface. However, this specification is only evaluated for ISAM pools in data spaces, in other words for file processing with SHARUPD=YES.

i Commands and macros for cross-task user ISAM pools are still accepted for reasons of compatibility, but no longer result in such a pool being generated. Instead, DMS uses a standard ISAM pool in which the size specified beforehand by the user is not fallen below and the assignment of files to pools is retained.

The system administrator can use the SHOW-ISAM-CACHING command to obtain information on ISAM files buffered in privileged data spaces.

21.3 File processing

The file and processing attributes are defined in the FCB, in a FILE macro or in a ADD-FILE-LINK command.

21.3.1 File attributes

> In order to make the following easier to read, reference is made exclusively to the operands of the FILE and FCB macros. The equivalent operands of the command are listed in a table at the end of this section.

Access method and file format

The access method is specified via the FCBTYPE operand, where ISAM is the default value. Depending on the value specified for the BLKCTRL operand, the file is processed with NK-ISAM or with K-ISAM.

The file format (BLKCTRL=DATA/PAMKEY/NO) refers to the internal structure of the file: for NK-ISAM, the block control information is kept in the PAM block itself
(BLKCTRL=DATA), not in a separate PAM key. K-ISAM uses this PAM key: BLKCTRL=PAMKEY. The format BLKCTRL=NO (= no block control field) is, for ISAM, equivalent to the entry BLKCTRL=DATA.

Structure of a PAM page

K-ISAM: the block control information is kept in a PAM key which precedes the PAM page, and the first 16 bytes of the PAM page are available for user data.

NK-ISAM: each PAM page begins with a 16-byte block control field, the last 4 bytes of which contain ISAM-specific information.
The usable length of a PAM page is thus 2048 – 16 = 2032 bytes.

NK2 files: each first PAM page of a logical block begins with a 16-byte block control field, the last 4 bytes of which contain ISAM-specific information.
The usable length of the first PAM page is thus 2048 – 16 = 2032 bytes.

NK4 files: each physical page (4096 bytes) begins with a 16-byte block control field, the last 4 bytes of which contain ISAM-specific information.
The usable length of each page is thus 4096 – 16 = 4080 bytes.

Record and block formats

Fixed-length and variable-length records, specified via the operand RECFORM=F/V of the FILE command, can be processed with ISAM. The record format affects the possible definitions of the record length and key position (see section “ISAM file structures”).

The record length is defined via the RECSIZE operand. For variable-length records, the user should remember that the first 4 bytes of the record are used for the record length field and a control field. Furthermore, RECSIZE then specifies the maximum permissible record length instead of the actual record length. If a file contains records which are longer than specified in RECSIZE, only the number of bytes specified in RECSIZE is returned by a read operation.

For fixed-length records, the entire record length specified in RECSIZE is available for user data. When specifying RECSIZE, however, the user should note that ISAM still places a 4-byte record length and control field at the beginning of each record, and that this may cause overflow blocks to be generated.

The maximum permissible record length is always the block size (BLKSIZE).

The blocking factor specified in the BLKSIZE operand defines the number of PAM pages which are to form a data block:
BLKSIZE=(STD,n), where 1 <= n <= 16.

The block size – and the record length – should always be selected such that no overflow blocks will be created, since this leads to a drop in performance.

Index structure

The ISAM index is formed from the ISAM key and the flags. It may be anywhere within the record, but must not extend into an overflow block. The maximum permissible index length is 255 bytes, and the minimum length 1 byte. Flags are evaluated with the aid of the GETFL macro.

The ISAM index begins with the key, whose starting position in the record is defined via the KEYPOS operand. The default value for KEYPOS depends on the record format: KEYPOS=5 for RECFORM=V, KEYPOS=1 for RECFORM=F. The key length is defined via KEYLEN; by default, DMS generates an 8-byte key.

The operands VALLEN and LOGLEN apply to the flags. The VALPROP operand has no function in NK-ISAM. It is provided only for reasons of compatibility, since it is used in K-ISAM to control inclusion of the value flag in the index entry.

The table below lists the above-mentioned operands of the FILE and FCB macros and their equivalent operands in the ADD-FILE-LINK command.

Operand in the FILE and FCB macros	Operand in the ADD-FILE-LINK command
FCBTYPE	ACCESS-METHOD
BLKCTRL=DATA/PAMKEY/NO	BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK / *PAMKEY / *NO
RECFORM=F/V	RECORD-FORMAT=*FIXED / *VARIABLE
RECSIZE	RECORD-SIZE
BLKSIZE	BUFFER-LENGTH
KEYPOS	KEY-POSITION
VALLEN	VALUE-FLAG-LENGTH
LOGLEN	LOGICAL-FLAG-LENGTH
VALPROP	PROPAGATE-VALUE-FLAG

Table 55: ISAM operands in FILE/FCB and ADD-FILE-LINK

21.3.2 Processing attributes

This section describes the processing attributes of ISAM files. These include shared-update processing and the opening of files, which are described separately in section “Sharedupdate processing” (Shared-update processing) and section “Opening an ISAM file”, respectively.

File link name and pool link name

The file link name links the file to the file control block in the program via the TFT (task file table) (see section “Access via the file link name”).

If an NK-ISAM file is to be processed in a user ISAM pool, a pool link name must be specified in POOLLNK operand of the ADDPLNK macro or in the POOL-LINK operand of the ADD-POOL-LINK command to connect the file to the desired pool. The pool can be created using the CREPOOL macro or CREATE-ISAM-POOL command. The pool link name is then defined by means of the ADDPLNK macro or the ADD-POOL-LINK command. If POOLLNK is not specified in either the FILE or the FCB macro or POOL-LINK in the ADD-FILE-LINK command, the related NK-ISAM file is processed in a standard pool. K-ISAM files cannot be processed in ISAM pools.

Duplicate keys

The user may specify in the FILE/FCB macro (operand DUPEKY) or in the ADD-FILE-LINK command (operand DUPLICAT-KEY) that several different records in his/her file may have the same key. In NK-ISAM, DMS adds a time stamp to these records; this is not done for K-ISAM.

Block usage: PAD value

During sequential creation of a file with PUT, the user can specify via the PAD operand in the FILE/FCB macro or via the PADDING-FACTOR operand in the ADD-FILE-LINK command how much space is to be left free in each data block for subsequent updating. The default value is PAD=15, i.e. 15% of the space in each data block is left free. If a PUT macro would result in less free space in an existing block, a new data block is created for the next record.

For NK-ISAM files, a new data block is requested as soon as this limit is exceeded; K-ISAM requests a new data block before the limit is exceeded.

For sequential creation with PUT, the size of a file increases if the PAD factor is increased. However, subsequent processing of the file (STORE/INSRT) can be optimized by selecting a suitable value for PAD: the free space left in the data blocks should be so large that block splitting does not occur when the file is subsequently extended. In order to select the correct PAD value, it is thus necessary to estimate how much the file will grow.

If ISAM files are created using STORE, the PAD value has no effect on how the blocks are filled: STORE writes records into a block until it is full. If a further record is then written, block splitting occurs. The blocks are usually only 50% full.

Even if the file is created by means of PUT, but not via an I/O area in the program, the PAD has no effect (the I/O area is defined in the FCB with IOAREAn; see “Program buffer = I/Oarea in the user program”). Each PUT macro initiates a write operation and DMS attempts to place the new record in the current data block. A new data block is created only when the current block is full.

Overlapped reading

If an ISAM file is to be processed primarily sequentially, the user can use OVERLAP=YES in the FILE macro or the READ-IN-ADVANCE operand in the ADD-FILE-LINK command to specify that, whenever a data block is read, the next one is also read, “just to be on the safe side”. This may reduce the number of I/O operations. Overlapped reading is performed for both “forward” (towards end-of-file) and “backward” (towards beginning-of-file) read operations.

WROUT function

The WROUT function (or WRITE-IMMEDIATE function) controls how often updated blocks are written back to disk. If WROUT is active, the file on the disk is always kept consistent with its copy in virtual memory.

If WROUT is not active, a block is written back to disk only when its buffer is needed for another block. This delay saves write operations in cases where multiple updates are executed in the same block.

In ISAM pools which are not task-specific, WROUT=YES is set by default: updated blocks are always written back to disk immediately. Delayed writing to disk is also possible. This must be specified explicitly by the user in the CREPOOL macro or CREATE-ISAM-POOL command, as well as in the FILE macro or ADD-FILE-LINK command.

If a file is processed in a task-specific user ISAM pool, the WROUT function can be activated both in the FILE/FCB macro for the file and in the CREPOOL macro for the pool. If commands are being used, the WRITE-IMMEDIATE function can be activated in both the ADD-FILE-LINK command for the file and in the CREATE-ISAM-POOL command for the pool. If these two specifications differ, the one which activates the WROUT function is used.

Program buffer = I/O area in the user program

If a program uses its own I/O area, this must be at least as large as a data block (= n * 2048 BLKSIZE, where 1 <= n <= 16), as specified in BLKSIZE=(STD,n). By default, the system generates an I/O area in class 5 memory.

The existence of an I/O area defined by the user via IOAREA1/2 in his/her program is particularly advantageous for the sequential processing of ISAM files, since it reduces the number of SVCs:

- sequential read operations (GET/GETR): in the first read operation, as many records as possible are transferred to the I/O area before the first record is returned to the program. For subsequent read operations, the records already in the I/O area are returned to the program. A new I/O operation is executed only when all of these records have been passed to the program.
- sequential write operations (PUT): during the sequential creation or extension of a file, the records are collected in the I/O area until it is full (taking the PAD value into account) or until sequential processing is terminated by a call for another operation. Care should be taken that sequences of PUT operations are not interrupted by other actions, since each such interruption causes the contents of the I/O area to be written as a new data block and this can lead to blocks with relatively small amounts of data.

For NK-ISAM in move mode, the user can dispense with an I/O area (IOAREA1=NO in the FCB). In this case, each action macro call leads to an SVC.

Operating modes: Move mode and locate mode

ISAM actions are normally executed in move mode. Some ISAM actions can be executed in locate mode, which is selected using the operand IOREG. Since it is then not necessary to transfer records from and to an I/O area, slight performance improvements can be achieved in this manner for sequential processing. Locate mode is supported by NK-ISAM only for reasons of compatibility. For further details, see chapter “Access methods”.

21.3.3 Index and data separation

K-ISAM files can be created with their index and data sections on different private disks. Devices, volumes and storage space can be assigned independently to the two parts of the file.

i The description below makes reference to the operands of the FILE macro. The corresponding operands of the commands are listed in a table for those users working with the DMS commands.

The operands DEVICE, VOLUME and SPACE in the FILE macro apply to the index section, while DDEVICE, DVOLUME and DSPACE apply to the data section. The devices, volumes and spaces are reserved in the same manner. However, the user must remember that specifications for SPACE apply to VOLUME and DEVICE, while specifications for DSPACE apply to DVOLUME and DDEVICE. Only the device types specified for DEVICE may be used in specifications for DDEVICE and there is no default value for DSPACE when a file is created.

If a FILE macro refers to a file which does not yet have storage space reserved, DDEVICE, DVOLUME and DSPACE must always be specified together. For files which already have some storage space, DSPACE may be specified without DDEVICE and DVOLUME. Only the storage space for the entire file may be released (not just for the index section or the data section). Once an ISAM file has been created with the index and data sections on different volumes, it is not possible to move both of these sections to the same volume.

Such separation of the data and index sections of an ISAM file is not possible on public volumes. For reasons of compatibility, NK-ISAM does not reject the operands DDEVICE, DSPACE and DVOLUME, but it ignores them during processing. Due to the optimized buffer management, it is no longer necessary to split the file into separate index and data sections when using NK-ISAM.

The table below contains a list of the equivalent command operands:

Operand in the macro	Operand in the CREATE-FILE and MODIFY-FILE-ATTRIBUTES commands
DEVICE	DEVICE-TYPE
VOLUME	VOLUME
SPACE	SPACE
DDEVICE	DATA-DEVICE-TYPE
DVOLUME	DATA-VOLUME
DSPACE	DATA-SPACE

Table 56: Operands in the macro and commands for index and data separation

21.4 Shared-update processing

ISAM files can be opened and updated concurrently by several jobs. Each such job must activate shared-update processing in the corresponding FILE/FCB macro or ADD-FILE LINK command (SHARUPD=YES or SHARED-UPDATE=*YES, as appropriate). Data consistency is assured by the use of internal locks.
The user can also explicitly lock records by means of the ISAM key. In the macros GET, GETRL, GETKY and GETR the lock is requested by means of the LOCK operand. If the desired lock cannot be imposed because the record or range is already locked by another job, the job requesting the lock is entered in a queue (unless the program contains a routine (EXLST exit PGLOCK) which is activated when a page-lock event occurs).

21.4.1 Locks

The locking of records is implemented differently in NK-ISAM (where key locks are used) and in K-ISAM (where block locks are used).

Key locks (NK-ISAM)

If a record is to be inserted into a file (STORE, INSRT) or a record with a specified key is to be deleted (ELIM), the record is locked by ISAM and unlocked again after processing. Even if the user has explicitly requested the record lock (external lock using LOCK), ISAM releases it after the write operation has been completed – except in the case of sequential write operations (PUT) via an I/O area. A range of keys is locked, the lower limit being a key less than or equal to the key of the records in the buffer. Its upper limit is “high value” (X'FF').

If a record is to be updated, the write operation is preceded by a read operation. The user must lock the record when he/she issues the read call (GET LOCK). The lock is released automatically when the record has been written back (PUTX) or deleted (ELIM with no key specified).

Read operations (GET, GETFL, GETKY, GETR) can be executed with or without a lock. Records which are not locked can be read and/or updated by other users. A lock requested for a read operation is not automatically reset by ISAM after the read operation. Even if a record is read with the aid of a secondary key, the lock applies to the primary key of the record.

For sequential reads with an I/O area, a key range is locked. The limits of this range are determined by the lowest and highest keys of the records currently in the I/O area.

Releasing locks

The user can explicitly clear a lock which exists in a file for his/her job (ISREQ). This may, for example, be necessary if he/she has locked a record for a read operation and this read operation is followed by no further ISAM actions for the file. Since each job may have only one lock active at any time, any ISAM operation which explicitly or implicitly requests a lock automatically clears any existing lock, thus avoiding the possibility of a deadlock situation.

Block lock (K-ISAM)

In K-ISAM, all locks are implemented in the form of block locks, i.e. the data block containing the record is locked for all other users.

21.4.2 OPEN modes for shared-update processing

The first user who opens an ISAM file may specify any possible combination of the values valid for OPEN and SHARUPD. The following table shows which combinations of OPEN and SHARUPD are permitted for user B if user A has already opened the file. If the file has already been opened by more than one user, the OPEN/SHARUPD combination specified by user B is compared with all other OPEN/SHARUPD combinations. Uer B may open the file only if the results of this comparison permit it.

			USER B									
	SHARUPD=		***YES**					***NO**				
		OPEN mode	INPUT	INOUT	EXTEND	OUTIN	OUTPUT	INPUT	INOUT	EXTEND	OUTIN	OUTPUT
USER A	***YES**	INPUT	X	X	X			X		X		
		INOUT	X	X	X					X		
		EXTEND	X	X								
		OUTIN	X	X								
		OUTPUT	X	X								
	***NO**	INPUT	X					X				
		INOUT										
		EXTEND										
		OUTIN										
		OUTPUT										

Table 57: ISAM: Permissible SHARUPD/OPEN combinations

X indicates that this OPEN mode is permitted for user B.

Note on the NK-ISAM access method

When files are accessed in SHARUPD=YES mode, it is possible that processing will result in a file < 32 GB growing to become a file >= 32 GB.

Here it is necessary to distinguish between two cases:

- Callers who are prepared for this situation (with the specification LARGE_FILE=*ALLOWED in the FCB macro or EXCEED-32GB=*ALLOWED in the ADD-FILE-LINK command)
- Unprepared callers (specification LARGE_FILE=*FORBIDDEN in the FCB macro or EXCEED-32GB=*FORBIDDEN in the ADD-FILE-LINK command).

On each SVC entry, the size of the NK-ISAM file in question is checked on the basis of the extent list present in the File Entry Table. If this check indicates that the file size is greater than 32 GB and if the caller has set the attribute LARGE_FILE=*FORBIDDEN in the FCB then processing is canceled. In this case, NK-ISAM issues the return code.

```
X'00000A23' FILE SIZE GREATER 32 GIGABYTES IS NOT ALLOWED.
```

(or the corresponding DMS message: `DMS0A23`)

The K-ISAM access method (BLKCTRL=PAMKEY) is not affected by this problem.

21.4.3 ISAM pools for shared-update processing (NK-ISAM)

Files which are to be opened for shared-update processing must be assigned to a hostspecific user ISAM pool (or one that is not task-specific) so that other users can access this pool and thus also the file. Before file processing is started, the user must create a hostspecific ISAM pool by means of the CREPOOL macro (operand SCOPE=HOST) or with the CREATE-ISAM-POOL command (operand SCOPE=HOST-SYSTEM) or link his/her job to an existing host-specific ISAM pool.

By means of the ADDPLNK macro or the ADD-POOL-LINK command, he/she must assign a pool link name to this pool and specify this link name in the POOLLNK operand of the FILE/FCB macro or in the POOL-LINK operand of the ADD-FILE-LINK command. Furthermore, he/she must also select NK-ISAM processing with the operand BLKCTRL=DATA of the FILE/FCB macro or with the operand BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK of the ADD-FILE-LINK command. After completion of file processing, he/she must delete the pool link name and the pool itself (by means of the REMPLNK macro (DELPOOL operand) or the REMOVE-ISAM-POOL/DELETE-ISAM-POOL command).

The following figures show which combinations of OPEN mode, SHARUPD and pool scope are permitted.

Example 1: Input file

A file is used as an input file: it can be assigned to any number of task-specific ISAM pools and to only one ISAM pool that is not task-specific.

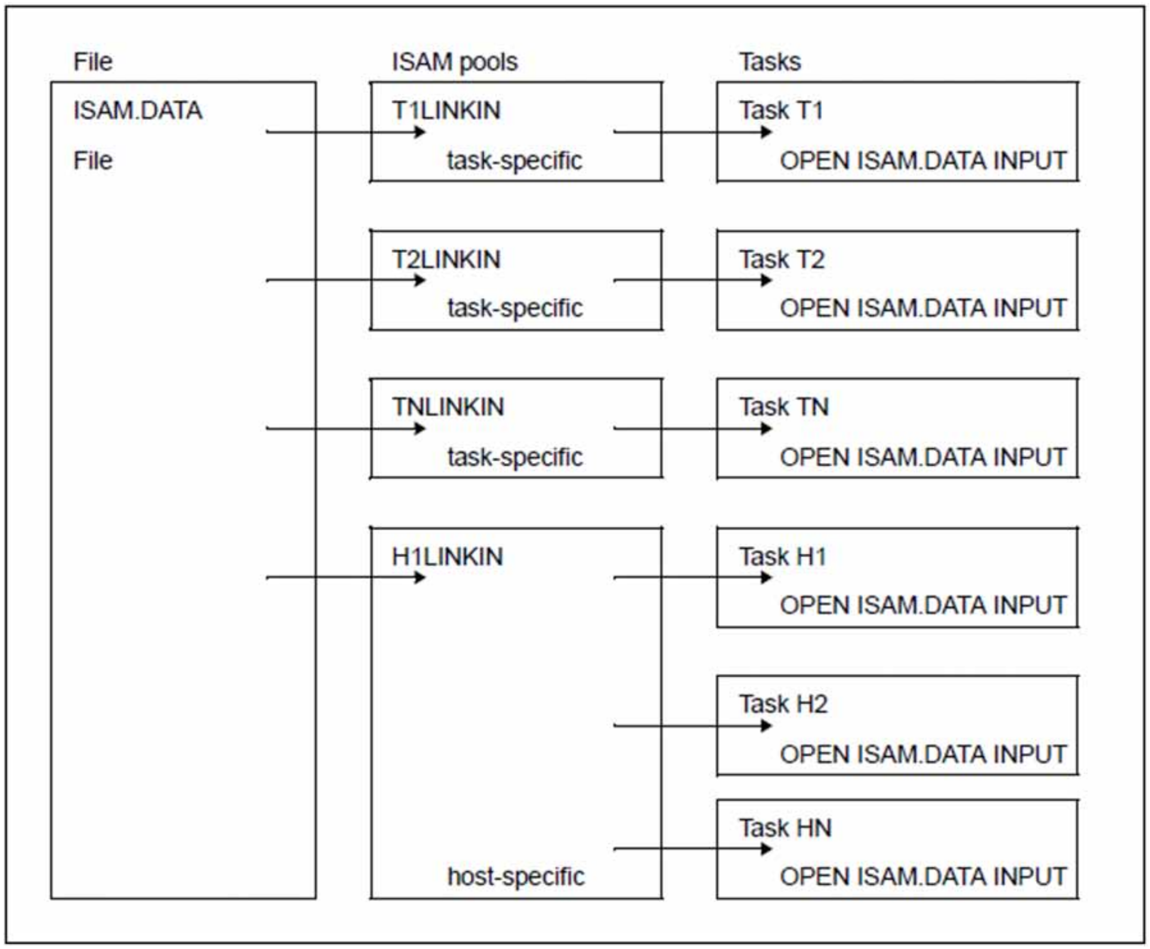


Figure 37: Input file in ISAM pools

Example 2: Output file

A file is to be opened concurrently by several jobs and updated by at least one of these jobs: it must be processed in an ISAM pool which is not task-specific, and must be opened with SHARUPD=YES in all participating jobs. If a user-specific ISAM pool is involved, all tasks must run under the same user ID.

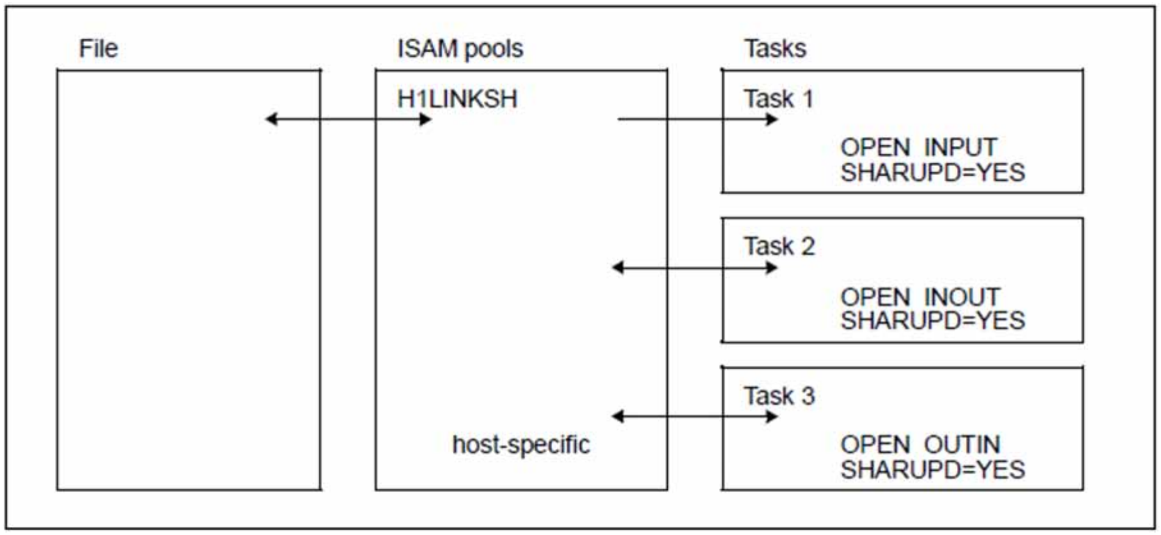


Figure 38: Shared-update processing in ISAM pools

21.5 Opening an ISAM file

Before a file can be processed, it must be opened using an OPEN macro. The following open modes are permitted for ISAM files: OUTPUT, OUTIN, EXTEND, INOUT, INPUT. At the same time, it is necessary to check whether the file has already been opened by another job, in which ISAM pool it is to be processed, whether it is to be processed in locate or move mode, etc. The table below provides an overview of the open modes.

OPEN mode	Brief description
OUTPUT	A new file is created sequentially and only the PUT macro may be used. If an ISAM file with the specified name already exists, it is overwritten and the catalog entry is updated.
OUTIN	As for OPEN OUTPUT, a new file is created and any existing file is overwritten. All ISAM actions are permitted.
EXTEND	An existing file is extended sequentially; as for OPEN OUTPUT only write operationswith PUT are permitted.
INOUT	An existing file is to be updated: as for OUTIN, all ISAM actions (such as finding, reading, updating, inserting and deleting records) are permitted.
INPUT	An existing file is to be read, i.e. only read operations are permitted.

Table 58: Open modes when opening an ISAM file

File locks

If a file is not opened for shared-update processing, the OPEN mode determines whether other jobs may work concurrently with this file. If a file is opened in any mode other than INPUT, it is locked for all other jobs; if it is opened in INPUT mode, other jobs may also open it with OPEN INPUT.

Storage space allocation

The minimum space allocation must make allowance for the fact that there is always at least one data block and one index block and, for NK-ISAM files, one control block. This means, for BLKSIZE=(STD,n), that the primary allocation for NK-ISAM files must always be at least n + 2 PAM pages, while n + 1 PAM pages are sufficient for a K-ISAM file. Since no secondary allocation can be made when a file is opened, an OPEN error will result if insufficient space is allocated. (For further details of the primary allocation, see section “Requesting storage space”.)

Operating modes

ISAM files are normally processed in move mode. File processing in locate mode is still supported for NK-ISAM, but only for reasons of compatibility.

Action macro	OPEN type				
	INPUT	OUTPUT	EXTEND	INOUT	OUTIN
GET	B	-	-	B	B
GETR	B	-	-	B	B
GETFL	B	-	-	B	B

GETKY	B	-	-	B	B
PUT	-	B	B	B	B
PUTX	-	-	-	B	B
INSRT	-	-	-	M	M
STORE	-	-	-	M	M
ELIM	-	-	-	x	x
SETL	x	-	-	x	x

Table 59: ISAM action macros and OPEN types

B:	Move mode or locate mode permitted.
M:	Move mode (locate mode only if a work area is supplied).
x:	Action macro may be used.
-	Action macro may not be used.

21.5.1 Transferring file attributes to the FCB

The contents of FCB fields for file attributes may differ from the values specified for the corresponding operands in the FILE macro. This is true, for example, for KEYPOS, KEYLEN, PAD and BLKSIZE. If “n” is the value in field KEYPOS and “m” is the value of KEYLEN in the catalog entry or in the TFT or the FCB macro, the following applies to files which are open:

- field KEYLEN in the P1 FCB contains m-1
- field KEYPOS in the P1 FCB contains, for RECFORM=F, (n+4)-1 = n+3

The PAD value is taken into account only when a file is created sequentially by means of the PUT macro. The macros INSRT and STORE use all the available space of each block, but block splitting may occur during creation of the file.

The contents of field BLKSIZE in the P1 FCB may also differ from the value of the BLKSIZE operand in the FILE or FCB macro or the catalog entry: except for input files, the BLKSIZE is recalculated, taking the PAD value into account (BLKSIZE minus PAD) and placed in FCB field BLKSIZE.

Example

> Catalog entry: BLKSIZE=(STD,3); PAD=15 (default value).
> The contents of BLKSIZE in the TU-FCB are calculated as follows: (3 * 2048) * (1.0 – 0.15); during file processing, the field BLKSIZE contains the value X'1467'.

21.5.2 DUMMY files

Dummy files can be used for test purposes, particularly for testing error routines in user programs. These dummy files are defined by *DUMMY in the FILE macro and ADD-FILE-LINK command. No actual I/O operations are executed when these files are processed. Any attempt to read such a file causes control to be passed to an error routine and write attempts are ignored, i.e. in each case a no-op (i.e. no operation) is executed.

The following table shows which events occur for the various ISAM read operations.

Read operation	Macro	Event	EXLST exit	Message
Sequential read	GET/GETR	End of file	EOFADDR	DMS0AAE
Read with key	GETKY	Key not found	NOFIND	DMS0AA8

Read with flags	GETFL
	LIMIT=KEY
	LIMIT=END
		Key not found
		End of file
			NOFIND
			EOFADDR
				DMS0AA8
				DMS0AAE

Table 60: ISAM read operations for dummy files

21.6 Using NK-ISAM

Using ISAM pools

For processing without shared-update, each user may decide whether his files are to be processed in user pools or in standard pools provided by the system.

If maximum performance is desired for large applications, user ISAM pools should always be used, since this is the only way of matching the pool size to the application's requirements.

Standard pools can be used if the user works only occasionally with ISAM and generally has small applications. Standard pools may also be suitable in the initial stages of the changeover from K-ISAM to NK-ISAM.

Defining the size of ISAM pools

For the user, the question of the optimum pools size applies only to the user pools. The possibility of loading a complete file into an ISAM pool, so that no I/O operations are necessary during file processing, can be utilized only for small files: the maximum pool size is 4096 PAM pages.

ISAM pools should be capable of holding all of the index blocks of a file, together with at least one data block for each job which accesses the file. When calculating the optimum pool size, the size of the real memory must also be taken into account since large ISAM pools without sufficient real memory will result in an increase in the paging rate. Furthermore, a distinction can be made between “active files” which are accessed frequently and files to which access occurs only occasionally: only “active files” occupy space in the ISAM pool.

Sequential file processing in ISAM pools

For sequentially processed files, the I/O rate can be reduced only slightly. In task-specific pools, for example, block switching is carried out by linear incrementation of the index entries, and access almost always affects the same index blocks. Another way of optimizing file processing is by using larger blocking factors. For sequential processing, therefore, large pools are not advisable.

Record length

Since each PAM page starts with a 16-byte control field, the space available in each block is reduced accordingly. To avoid the creation of overflow blocks, the record length should comply with the following rule:

```
RECSIZE <= (n * 2048) - (n * 16) - 16 - F - T
```

F	Record format F = 0/4 for RECFORM = V/F
T	Time stamp T = 0/8 for files without/with duplicate keys

(n * 2048) describes the block size; (n * 16) describes the block control field at the start of each PAM page.

Flag processing

As the flags are no longer included in the index, flag processing in NK-ISAM is implemented simply in the form of a sequential search through file sections. This can result in a serious drop in performance. In the GETFL macro, the range to be searched should be kept as small as possible by specifying appropriate values in the LIMIT operand.

Key position and index length

The ISAM index must not extend into an overflow block. The following rule thus applies to the index structure (taking due account of existing flags):

```
KEYPOS + KEYLEN + VALLEN + LOGLEN <= n * 2032 - 16 - F
```

n	Blocking factor in BLKSIZE=(STD,n), where 1 <= n <= 16
F	Record format • F = 0 for RECFORM = V • F = 4 for RECFORM = F

ISAM pools and BLKCTRL = DATA without NK-ISAM

If a user attempts to create ISAM pools or to process NK-ISAM files (BLKCTRL=DATA) although NK-ISAM is not loaded on his/her system, processing is aborted with an error message. For ISAM pool macros, a return code is placed in the standard header of the operand list. The FCB of the related file contains an error code in field ID1ECB.

ISAM files copied to tape

ISAM files can be saved to tape, but they cannot be processed as tape files with ISAM.

No information about the data format (BLKCTRL or BLOCK-CONTROL-INFO, as appropriate) is kept on tape. This information is lost for files copied to tape (with COPFILE or COPY-FILE) if the catalog entry for the file is deleted.

If the file is to be copied back to disk, the COPFILE macro must be preceded by a FILE macro with the operand STATE=FOREIGN or the COPY-FILE command must be preceded by an IMPORT-FILE command, as appropriate. The user must specify the correct value for the operand BLKCTRL or BLOCK-CONTROL-INFO, namely PAMKEY or DATA (macro) or *PAMKEY or *WITHIN-DATA-BLOCK (command) to match the actual data format.

If a K-ISAM file (BLKCTRL=PAMKEY) is inadvertently copied back to disk as an NK-ISAM file (BLKCTRL=DATA), the resulting disk file cannot be read since the first 16 bytes of each PAM page, which contain data in the format BLKCTRL=PAMKEY, are overwritten with management information.

OPEN errors when processing NK-ISAM files

- NK-ISAM file on tape: an invalid data format was specified in the FILE or FCB macro, or in the IMPORT-FILE command, when importing the tape file or writing it back to tape.
- ISAM pool is overloaded: if the ISAM pool in which an NK-ISAM file is to be opened is overloaded, OPEN processing is rejected with error message `DMS0D9B`.

21.7 K-ISAM/NK-ISAM conversion

- Migration
- Reverse migration
- Compatibility

21.7.1 Migration

In the migration from K-ISAM to NK-ISAM, the following three stages can be distinguished: conversion, modification of command procedures, modification of source code.

Conversion

K-ISAM files can be converted to NK-ISAM format by means of “logical copying”. “Logical copying” can be performed, for example, by the utility routine PERCON (see the “PERCON” manual“ [12 (Related publications)]). In such cases, it is not necessary to modify either the programs which process the files or the command sequences of the jobs in which the files are addressed.

Command procedures

By changing the command sequences in procedures, the performance advantages offered by the NK-ISAM interface with its ISAM pools can be utilized. The commands for ISAM pool management can be inserted at the appropriate positions in jobs whose performance is critical. In the FILE/FCB macros, the POOLLNK operand, and in the ADD-FILE-LINK command, the POOL-LINK operand must be added so that the files are linked to ISAM pools and can be processed in the pools. If NK-ISAM files are to be opened with SHARUPD=YES or SHARED-UPDATE=*YES, they must be processed in host-specific user ISAM pools. For this purpose, the job must be linked with the corresponding ISAM pool beforehand with CREATE-ISAM-POOL. The pool must be assigned a link name with ADD-ISAM-POOL-LINK and this link name specified in the POOL-LINK operand. Once file processing has been concluded, the links to the ISAM pool must be removed (REMOVE-ISAM-POOL, DELETE-ISAM-POOL) unless the user is working with default pools.

Source code modification

By means of changes to the source code of the processing programs (and possibly also to the file structure), ISAM processing can be optimized for NK-ISAM:

- match the record length to the BLKSIZE in order to avoid overflow blocks
- in flag applications, keep the range to be searched as small as possible (with GETFL ...,LIMIT=)
- ISAM pool macros and FILE/FCB macros must specify the POOLLNK operand for ISAM processing in ISAM pools.

21.7.2 Reverse migration

NK-ISAM files can be converted back to K-ISAM files with the aid of the utility routine PAMCONV. The utility routine PAMCONV is described in the “Utility Routines” manual [13 (Related publications)].

21.7.3 Compatibility

As the ISAM action macros are invariable as far as the format of the file to be processed is concerned, there are no conversion problems on this level. For example, the same macro can be applied to both NK-ISAM and K-ISAM files in a program if the FCB address is passed in a register. The sole limitation with respect to the user interface is that overflow blocks must not contain parts of the ISAM index. In the transition from NK-ISAM to K-ISAM, the user must of course remember that ISAM pool macros are supported only if NK-ISAM is loaded.

The following table summarizes the most important changes in NK-ISAM, together with references to their possible effects.

NK-ISAM	Effect
Flag processing	Performance deterioration when the GETFL macro is used
Index and data separation	Is ignored; secondary allocations are effective only for the data section
File format	Note for UPAM processing of ISAM files: usable area of data blocks is reduced; overflow blocks increase the I/O rate
ISAM pools	Possibly increased address space requirements;class 4 memory load is reduced for shared-update processing
I/O areas	IOAREA1/2 is replaced by the pools and is needed only in locate mode; advisable for sequential processing
PAD factor	PAD limit must be exceeded before a new data block is requested
Record lock	Replaces the old block lock
Checkpoint /restart	No checkpoints are written with host-specific or user-specific ISAM pools; a checkpoint file with NK-ISAM files can be used in a restart only if NK-ISAM is loaded.

Table 61: Incompatibilities when converting to NK-ISAM

21.8 “Crash resistance” of ISAM files

A file is “crash-resistant” if, after a system crash, it is still in a consistent state and the results of all operations successfully executed beforehand have not been corrupted.

After a system crash, write operations started beforehand will be either complete or not executed at all. In both cases, however, the file is still consistent as far as ISAM is concerned.

When the system crashes, files which are open cannot be closed normally, which means that the LASTPG pointer is no longer correct. A subsequent OPEN detects this and permits the user program to start an error routine (OPENC exit of the EXLST macro) to rectify the problem (VERIF macro). If no such error routine exists, the program is aborted with an error message to this effect (`DMS0DD1, DMS0D1A`). The last-page pointer of a file can be updated by means of the VERIF macro or the REPAIR-DISK-FILES command (see section “File recovery”). Crash-resistance as described above is guaranteed for WROUT=YES or WRITE-IMMEDIATE=*YES and when a file is processed with PUT macros with an I/O area in the program.

*Crash behavior with WROUT=NO or WRITE-IMMEDIATE=*NO*

If WROUT=NO or WRITE-IMMEDIATE=*NO is specified, changes to records which have not yet been written back to disk may be lost. File consistency can be restored by means of the VERIF macro (operand REPAIR=ABS) or the REPAIR-DISK-FILES command. However, the user data may still be inconsistent. It is the user's responsibility to restore the necessary data consistency.

PUT (sequential write) with an I/O area in the program

If an I/O area is used for PUT macro operations, output to disk is initiated only when the amount of space left in the buffer drops below the specified PAD value. If a system crash occurs before this, all records still in the buffer, but not yet written to disk, are lost. From the user's point of view, the data is inconsistent and/or incomplete, but ISAM regards the file as consistent.

22 SAM - Sequential Access Method

The access method SAM processes files sequentially. This includes searching for a record, updating the record, and writing it back to the file.

As SAM is a record-oriented access method, it carries out the blocking, deblocking and buffering of the records for the user. If two I/O areas are available in the user program, exchange buffer operation can be used. If only one I/O area is provided, no overlapped processing is possible.

SAM is virtually device-independent and permits the processing of files on disks and tapes; tape cartridges are essentially handled like normal magnetic tapes.

Block formats

The SAM access method is capable of processing files with different block formats (see section “Block formats for disk files”, and section “Block formats for tape files”). The BLKCTRL operand of the FILE and FCB macros and the BLOCK-CONTROL-INFO operand of the ADD-FILE-LINK command can be used to specify whether a K file or an NK file is to be processed:

- BLKCTRL=PAMKEY or BLOCK-CONTROL-INFO=*PAMKEY: A K-SAM file is to be processed.

  K-SAM files (Key SAM files) have the conventional "PAMKEY" block format: they are characterized by the fact that DMS management information for each PAM page is held in a separate PAM key located outside the page.
- BLKCTRL=DATA/NO or BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK/*NO specifies an NK-SAM file.

  NK-SAM files (non-key SAM files) have the block format “DATA” or “NO”: They do not contain separate PAM keys. With the block format “DATA”, the DMS management information is kept in a block control field within the PAM page. With SAM, block format “NO” only exists for tape files. No such block-specific management information is supported for the block format “NO”.

With respect to the functional scope, there is no difference between K-SAM and NK-SAM files. When planning files, however, the user must remember that part of each logical block in an NK-SAM file is needed for the block control field and is thus not available for user data (for further details, see section “Logical block in a K-SAM file” and section “Logical block in an NK-SAM file”).

SAM functions

SAM supports file processing in both move mode and locate mode. In move mode the system takes responsibility for moving the data records between buffers and the user input/output area. In locate mode the records are processed directly in the input/output buffer and the individual user is responsible for correctly addressing his/her data records. In locate mode it is possible to obtain direct access to a record using the retrieval address which DMS contains in the FCB when the file is created (see section “FCB retrieval address”).

The following action macros are available in SAM to control file processing:

Macro	Brief description
GET	Sequential read: the records are returned sequentially; SAM expects that the records will be required in the order in which they were written.

PUT	Sequential write operations: in move mode the logical routines of the access method carry out the blocking of the records; this means that output to the volume is delayed until the output buffer is full. The buffers are managed automatically by the system. In locate mode the user must deal with the blocking himself/herself.
PUTX	Write an updated record back to disk (only in locate mode for disk files).
RELSE	Closes a data block., i.e. for input files the next GET macro will read the next data block, and for output files the next PUT macro will write the contents of the buffer as a data block, and the next record will become the first record in the next data block. (This is necessary in locate mode if the following record does not fit into the current buffer.)
SETL	Position to a specific block within the file.
FEOV	For tape files: initiate a tape swap.

Table 62: Action macros for SAM file processing

If files are opened as output files (OPEN OUTPUT/EXTEND), SAM interprets each PUT or SETL macro as an end-of-file (EOF) indicator. The last PUT or SETL macro prior to a CLOSE macro for a file thus automatically indicates EOF to the system. If the user wishes to delete all records after a specific record in a file, he/she can use the SETL macro to position to the desired point in the file, and then issue a CLOSE macro to close the file.

For direct access, a retrieval address is made available to the user. The format of this retrieval address is described in detail in section “FCB retrieval address”. When a record is written, its retrieval address is made available in the FCB. If the user wishes, he/she can create a new file from this retrieval address data to serve as a basis for subsequent non-sequential processing of this file. The retrieval address is also made available in the FCB after execution of a GET macro. In this case too, therefore, even if the user did not create the file, he/she can still create a secondary file from the retrieval addresses in order to perform subsequent non-sequential processing of the file.

If, in move mode with two output buffers, physical end-of-volume is detected when a data block is written to a tape file, the other buffer (which contains only one record) is written to the other tape. A tape swap is not executed until this has been done.

If a file is to be created in locate mode (OPEN OUTPUT/EXTEND), the start address of the first record to be written is returned to the user, after OPEN, in the register specified in the operand IOREG in the FCB macro. After execution of the PUT macro, this register contains the address for the next higher record. In the case of a file with variable-length records (RECFORM=V), the user also always receives the number of free bytes in the current block: this information is passed in the register he/she specified in the VARBLD operand of the FCB macro.

22.1 Logical blocks for SAM files

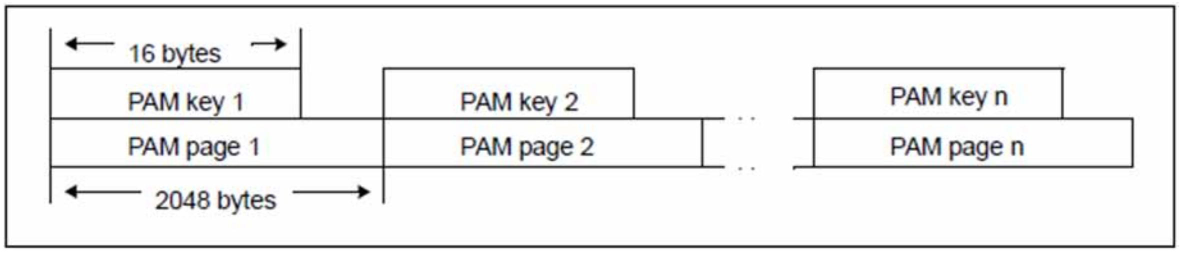
The format of the logical blocks is defined in the FILE and FCB macros via the BLKCTRL operand and in the ADD-FILE-LINK command via the BLOCK-CONTROL-INFO operand. SAM files can contain records with the format F, V or U.

The size of the record must not exceed the block size. The maximum block length for nonstandard blocks is 32768 bytes for SAM tape files.

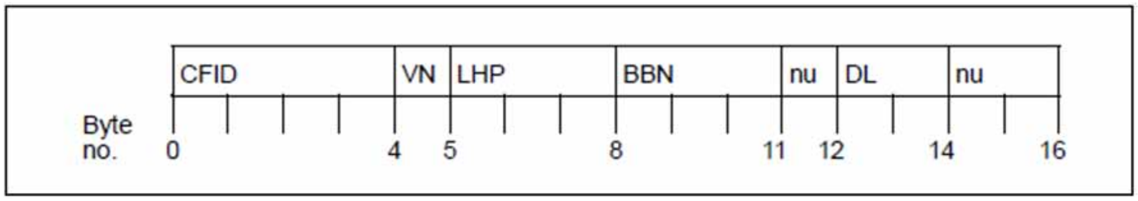
22.1.1 Logical block in a K-SAM file

Each PAM page of a conventional K-SAM file has a 16-byte PAM key, located outside the page, in which DMS management information is kept.

In a K-SAM file in which n PAM pages have been grouped together using either the operand BLKSIZE=(STD,n) in the FILE or FCB macro or the operand BUFFER-LENGTH=*STD(SIZE=n) in the ADD-FILE-LINK command to form a logical block, this block has the following structure:



All n * 2048 bytes of the logical block are available for user data, since the DMS management information is stored in the 16-byte PAM key. For SAM files, the PAM key has the following structure:



where:

CFID	Coded File IDentification: an identifier which enables DMS to determine, by comparing it with a corresponding CFID in the catalog entry, whether the data in the PAM page belongs to the file or whether the page is reserved (for this file) but does not yet contain data.
VN	Version number: This number permits a partial backup of a file by means of ARCHIVE or HSMS. This is a save run in which only those PAM pages of a file which have been modified since the last partial or incremental backup are saved.
LHP	Logical Half-Page number: this is used for consistency checking for files and for decompression when a file is recovered by ARCHIVE or HSMS from a backup copy.
BBN	Binary block number.
DL	Length of the usable data in the buffer.
nv	Bytes of the PAM key not used by SAM.

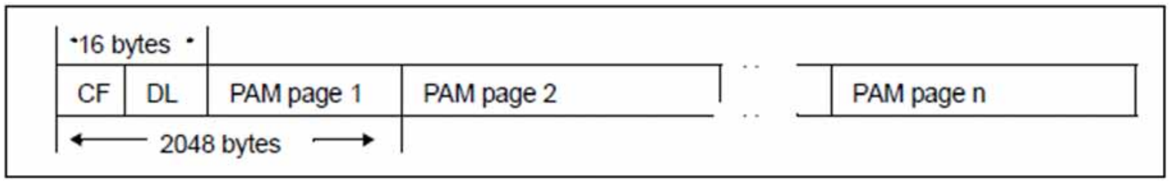
22.1.2 Logical block in an NK-SAM file

In contrast to conventional K-SAM files, NK-SAM files do not have separate PAM keys outside the PAM pages. Depending on what block format has been defined for an NK-SAM file, block-specific management information is either stored in the logical blocks themselves or simply discarded altogether.

Block format "DATA"

This block format is generated when a file is created with the operand BLKCTRL=DATA of the FILE macro or the operand BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK of the ADD-FILE-LINK command. The block-specific management data is stored in a special area of each logical block of the file, namely the 12-byte block control field. NK-SAM adds 4 bytes of information to this block control field indicating the length of usable data in the buffer.

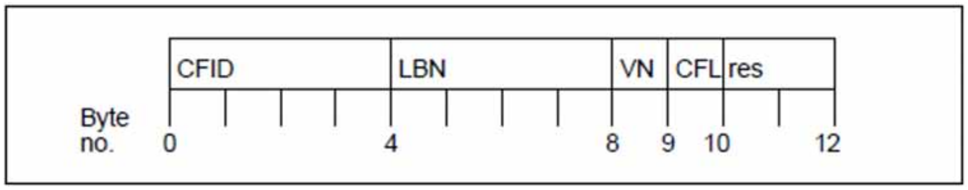
In an NK-SAM file in which n (where n <= 16) 2K blocks (standard blocks) have been grouped together using either the operand BLKSIZE=(STD,n) in the FILE or FCB macro or the operand BUFFER-LENGTH=*STD(SIZE=n) in the ADD-FILE-LINK command to form a logical block, this block has the following structure:



where:

CF	Block control field, 12 bytes.
DL	Length of the usable data in the buffer, 4 bytes.

The block control field has the following structure:



where:

CFID	Coded File IDentification: this has the same meaning as in the PAM key (see section “Logical block in a K-SAM file”).
LBN	Logical Block Number: number of the standard block which contains the block control field. (Standard blocks are numbered sequentially within a file.)
VN	Version number: this has the same meaning as in the PAM key.
CFL	Control FLag: the control flag flags gaps within a file. This information is used primarily when copying a file from or to a tape.
res	Bytes in the block control field which are reserved for future use.

The 12 bytes of a logical block used for the block control field and the following 4 bytes used for the SAM-specific length field are not available for storing user data. This fact must be taken into consideration when planning blocking factors and record lengths for an NK-SAM file.

The length of the part of a logical block consisting of n standard blocks available for user data (the maximum permissible record length) is thus always calculated as follows, regardless of the record format used:

Usable block size

```
    RECSIZE <= n * 2048 - 16
```

`RECSIZE`	permissible record length in the NK-SAM file
n	blocking factor from BLKSIZE=(STD,n), 1 <= n <= 16

When the buffer offset is used (operand BUFOFF=16 in the FCB and FILE macros or operand BLOCK-OFFSET=16 in the ADD-FILE-LINK command), files with the block format “DATA” can be used for data swapping. These files can be imported into V9.5.

Block format "NO"

This block format is generated when a file is created with the operand BLKCTRL=NO of the FILE macro or the operand BLOCK-CONTROL-INFO=*NO of the ADD-FILE-LINK command. SAM supports this block format *only for tape files*. No block-specific management information is generated. The user can make use of the full length of the logical block for his/her own data.

A tape file created by another system with block-specific management information can be imported with a special buffer offset (operand BUFOFF / BLKCTRL=NO in the FCB and FILE macros or operand BLOCK-OFFSET / BLOCK-CONTROL-INFO=*NO in the ADD-FILE-LINK command).

22.2 K-SAM/NK-SAM conversion

Converting K-SAM files into NK-SAM files

Prerequisites

K-SAM files can be converted into NK-SAM files using either of the utility routines PAMCONV (see the “Utility Routines” manual [13 (Related publications)]) or PERCON (see the “PERCON” manual [12 (Related publications)]). Conversion is possible only if the maximum record length in the K-SAM file is not greater than the usable block length of the NK-SAM file minus 16 bytes. For a K-SAM file with BLKSIZE=(STD,n) or BUFFER-LENGTH=*STD(SIZE=n), the following condition must therefore be fulfilled:

Maximum record length:

$$\texttt{RECSIZE}_{\texttt{K-SAM}} \texttt{<= (n * 2048 - 16)}$$

$\texttt{RECSIZE}_{\texttt{K-SAM}}$	maximum record length in the K-SAM file
n	blocking factor from BLKSIZE=(STD,n), 1 <= n <= 16

Optimizing block usage

After conversion, K-SAM files which fulfill the above condition with regard to the record length but which use all the available bytes in the logical block in K-SAM format for user data will require considerably (up to 100%) more storage space. The reason for this is that fewer records will fit into a logical block because of the reduced usable block length in NK-SAM format. This reduced effective block length must be taken into account if all the available bytes of a logical block are to be used in the NK-SAM format for user data (see "Logical block in an NK-SAM file"):

Usable block length:

$$\texttt{NBLN}_{\texttt{K-SAM}} \texttt{<= (n * 2048 - 16)}$$

$\texttt{NBL}_{\texttt{NK-SAM}}$	usable block length available for user data in an NK-SAM file
n	blocking factor from BLKSIZE=(STD,n), 1 <= n <= 16

Space requirements

When a K-SAM file is converted to NK-SAM format, the space required for a file with variable-length records increases by an average of 1%.

The increased space requirements for a file with fixed-length records when converted from K-SAM to NK-SAM format can be calculated using the following formula:

Space required for an NK-SAM file:

$$\#\text{PAM-pages}_{NKSAM} = (n * 2048 / (n*2048 - 16)) * \#\text{PAM-pages}_{K\text{-}SAM}$$

$\#\text{PAM pages}_{NK\text{-}SAM}$	space required for the NK-SAM file in PAM pages
$\#\text{PAM pages}_{K\text{-}SAM}$	space required for the K-SAM file in PAM pages
n	blocking factor from BLKSIZE=(STD,n), 1 <= n <= 16

Converting NK-SAM files into K-SAM files

NK-SAM files can be converted into K-SAM files with the aid of the PAMCONV utility routine (see the “Utility Routines” manual [13 (Related publications)]) or the software product PERCON (see the “PERCON” manual [12 (Related publications)]).

Compatibility

NK-SAM offers the user the same functional scope, with identical interfaces, as K-SAM.

However, when converting from K-SAM to NK-SAM, it may be necessary to modify programs which use their own algorithms for determining the retrieval addresses for records, since the blocking of the file to be processed may change during conversion. Due to the reduced usable block length, it may happen that each block contains one record less and that the file size increases. In such cases, the calculation algorithm for retrieval addresses must be modified.

Applications which use the retrieval address made available by DMS in the FCB (see section “FCB retrieval address”) require no such modifications.

22.3 Opening a SAM file (OPEN modes)

The following table provides an overview of the OPEN modes.

OPEN mode	Brief description
INPUT	Read the file towards the end of file; the file must exist.
OUTPUT	Create a new file sequentially or overwrite an existing file.
EXTEND	Add records to the end of an existing file.
UPDATE	Only for disk files in locate mode: update records; the record to be updated must first be retrieved by means of GET; the record length may not be modified during processing; the updated record is written back to the disk by means of PUTX.
REVERSE	Read the file sequentially toward the end of file; the file must exist;Tape files which extend over more than one volume can be read only section by section using the VSEQ operand (FILE macro) or using the START-POSITION operand (ADD-FILE-LINK command); no automatic tape swap.

Table 63: Open modes when opening a SAM file

The table below provides information on which action macros are permitted in conjunction with the above-mentioned open modes.

Action macro	OPEN type				
	INPUT	OUTPUT	EXTEND	UPDATE	REVERSE
GET	x			x	x
PUT		x	x		
PUTX				x	
RELSE	x	x	x	x	x
SETL	x	x	x	x	x

Table 64: SAM action macros and OPEN modes

An application processing SAM node files must set the SAM_NODE_FILE_ENABLE indicator in the file control block when it opens the file, to indicate that it knows the special aspects of processing SAM node files (see note in the section “FCB retrieval address”).

Primary and secondary allocations (disk files)

When a SAM file is created or overwritten (OPEN OUTPUT) or extended (OPEN EXTEND), both the primary and secondary allocations must be at least the same as the block size.

If a SAM file for which RECFORM=F or RECFORM=V is specified is to be created or extended in move mode (OPEN OUTPUT/EXTEND) and is to have more than one record per data block, then the following conditions apply:

- The primary allocation must be at least twice as long as the data block (primary allocation >= 2 * BLKSIZE). Otherwise control will be passed to the EXLST exit NOSPACE (insufficient storage space).

- The secondary allocation defined by SAM is implemented as of the first record of the last block that can still be written in the assigned area. If the secondary allocation cannot be implemented, control passes to the EXLST exit NOSPACE (provided it is available in the program). This is giving the user the opportunity to write the remaining records of the last block.

Effects of the LABEL operand (tape files)

The table below indicates the effect of specifying the LABEL operand of the FILE command or the LABEL-TYPE operand of the ADD-FILE-LINK command when a SAM file is opened.

FILE macro: Operand LABEL=	**ADD-FILE-LINK command: Operand LABEL-TYPE=**	**Effect**
NO / NSTD	*NO / *NON-STD	Specification of the BUFOFF operand will lead to an OPEN error with CODE=ISO /OWN and BLKSIZE=STD.
(STD,0)	*STD(DIN-REVISION-NUMBER=0)	Specification of the BUFOFF operand and CODE=ISO/OWN will lead to an OPEN error.
(STD,1)	*STD(DIN-REVISION-NUMBER=1)	Specification of the BUFOFF operand and the entry CODE=ISO/OWN will lead to an OPEN error.
(STD,2)	*STD(DIN-REVISION-NUMBER=2)	Standard blocks are converted into nonstandard blocks (BLKCTRL=DATA/NO) and V-records are converted to D-records; RECSIZE > 9999 together with RECFORM=V will lead to an OPEN error.
(STD,3)	*STD(DIN-REVISION-NUMBER=3)	Standard blocks are converted into nonstandard blocks (BLKCTRL=DATA/NO) and V-records are converted to D-records; RECSIZE > 9999 together with RECFORM=V will lead to an OPEN error.
STD or no LABEL operand specified	*STD	Specification of the BUFOFF operand will lead to an OPEN error.

The combination of the specifications for the operands CODE, RECSIZE and RECFORM (FILE macro) or CODE, RECORD-SIZE and RECORD-FORMAT (ADD-FILE-LINK command) result in the following values for LABEL if the file to be created is the first file on a tape:

CODE	**Block format**	**REFORM**	**LABEL value (implicit)**
EBCDIC	PAMKEY	-	(STD,1)

EBCDIC	DATA/NO	I	(STD,2)
EBCDIC	DATA/NO	F/V	(STD,3), V records --> D records
ISO/OWN	PAMKEY	-	OPEN error
ISO/OWN	DATA/NO	I	(STD,2)
ISO/OWN	DATA/NO	F	(STD,3)
ISO/OWN	DATA/NO	V	V records --> D records (STD,3) OPEN error if BLKSIZE > 9999

Table 66: Effects of the CODE, BLKSIZE and RECFORM specifications on the LABEL operand

The combination BUFOFF and RECFORM=U or BLKCTRL=DATA in the FILE macro is not permitted. The combination RECORD-FORMAT=*UNDEFINED and BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK is also not permitted.

In the case of LABEL=(STD,1) or LABEL-TYPE=*STD(DIN-REVISION=1), it is not possible to write a file with CODE=EBCDIC, with nonstandard blocks and D-records. However, it is possible to read such a file.

An OPEN error will occur if the standard label level requested via the LABEL operand does not match that specified in the VOL1 label.

If LABEL=STD or LABEL-TYPE=*STD is specified or the operand is omitted, the label level is determined as shown in table 66. If the resulting label level is higher than that in the VOL1 label, the label level defined in the VOL1 label is used. If the resulting label level is lower than that in the VOL1 label, or if the VOL1 label specifies LABEL=(STD, 0), and if CODE=ISO/OWN is specified, an OPEN error will result.

22.4 Record formats for SAM files

The SAM access method permits the record formats: F (fixed record length), V (variable record length) and U (undefined record length).

When format U is used, SAM reads or writes only one record per data block (buffer). For example, the definition RECFORM=U in combination with BLKSIZE=STD and BLKCTRL=PAMKEY (is equivalent to RECORD-FORMAT=*UNDEFINED in combination with BUFFER-LENGTH=*STD and BLOCK-CONTROL-INFO in the ADD-FILE-LINK command) and a current record length of 48 bytes would mean that 2000 bytes in each PAM page would be “wasted”.

If the full capacity of a standard block is not used (e.g. after the action macro RELSE, SETL, FEOV or CLOSE), the remaining bytes in the block are unchanged, which means that their contents are undefined.

The size of the record must not exceed the block size.

For further information about record formats, see section “Record formats”.

22.5 FCB retrieval address

When creating a SAM file, DMS places a retrieval address in the FCB. This address can be used for direct access during subsequent processing. It consists of a block number and a record number. The block number always refers to a logical data block (not to a PAM page), the record number indicates the position of the record within the indicated block. For multivolume files, it should be noted that the block number is maintained only within each volume.

In the 31-bit TU FCB, i.e. for XS programming, the retrieval address is split into two fields, each one word long: the field ID1BLK# contains the block number within the file and the field ID1REC# contains the record number within this block. Both fields are incremented automatically by the system. Whenever a data transfer operation is initiated, the record counter is automatically reset.

For non-XS programming, the retrieval address is kept in field ID1RPTR of the 24-bit TU FCB in the form “bbbbbbrr”, where

- “bbbbbb” is the number of the data block in the file and
- “rr” is the number of the record within the block.

The record counter is not automatically incremented by the system; the user must do this in his program if he/she wants to use the retrieval address. However, the record counter is automatically reset whenever a data transfer operation is initiated. For tape files, it should be noted that the retrieval address for non-XS programming is returned only for files with standard blocks, which means that SETL R cannot be used for files with nonstandard blocks.

XS interface: 31-bit-TU-FCB		**Non-XS interface: 24-bit-TU-FCB**	
ID1BLK# (1 word)	ID1REC# (1 word)	ID1RPTR (1 word)	
		Byte 1-3	Byte 4
Block number	Record number	Block number	Record number

Table 67: Structure of the retrieval address

The first record in a file thus has the following retrieval address:

- in the 31-bit TU FCB00000001 in field ID1BLK# and 00000001 in field ID1REC#
- in the 24-bit TU FCB00000101 in field ID1RPTR

The table below indicates the values of the retrieval address after the action macros SETL B and SETL E as a function of the open mode.

OPEN mode	**SETL B**			**SETL E**		
	ID1BLK#	ID1REC#	ID1RPTR	ID1BLK#	ID1REC#	ID1RPTR
INPUT, UPDATE	-	-	-	max.	1	max. 1
OUTPUT	1	0	1 0	error	error	error
EXTEND	1	0	1 0	error	error	error
REVERSE	-	-	-	-	-	-

Table 68: Values of the retrieval addresses after SETL B and SETL E as a function of the OPEN mode

-	Field contents unchanged.

max. Highest block number.

Field IDRPTR contains both the block and record counters.

The table below describes the manner in which the action macros supplies a value for the retrieval address.

Macro	Value supplied for the retrieval address
GET	If a data transfer is necessary for the specified record, the block counter contains the logical block number and the record counter is reset. For XS programming, the record counter is updated by each action macro.
PUT	If the PUT macro results in a data transfer for the specified record, the block counter is updated and the record counter is reset. The block counter thus contains the number of the new data block which is to receive the record. For XS programming, the record counter is updated by each action macro.
RELSE	If a file is to be created or extended (OPEN OUTPUT/EXTEND), the block counter contains the number of the data block which is to receive the following record and the record counter is set to zero.
FEOV	After a tape swap, the block and record counters are reset for the new tape by the system.

Table 69: SAM action macros and retrieval addresse

Example

The file attributes are defined:

- using the FILE macro:
  BLKCTRL=PAMKEY,
  BLKSIZE=(STD,2),
  RECFORM=F,
  RECSIZE=512

or

- using the ADD-FILE-LINK command:
  BLOCK-CONTROL-INFO=*PAMKEY,
  BUFFER-LENGTH=*STD(SIZE=2),
  RECORD-FORMAT=*FIXED,
  RECORD-SIZE=512

The retrieval address is:

	Retrieval address		
	31-bit-TU-FCB		24-bit-TU-FCB
	ID1BLK#	ID1REC#	IDRPTR
for record 10	00000002	00000002	00000202
for record 20	00000003	00000004	00000304

Information on processing SAM node files

If a data block is closed with RELSE during a write process on a SAM node file before it has been filled completely, the retrieval addresses are only valid from that moment on for as long as the file is open.
CLOSE and re-OPEN results in a different distribution of the data sets of the node file on the SAM blocks, which are transferred to the access method for processing; this means the previous retrieval addresses do not match the file anymore!

The same is true for processing with OPEN UPDATE. If a SAM node file was opened and enhanced in UPDATE mode, the retrieval addresses become invalid as soon as the file is closed and opened again.

This behavior of SAM node files (Net-Storage) is incompatible with the processing of SAM files on conventional public space, where the block and record structure remains unchanged even after CLOSE and OPEN. Because of this, the application must set the indicator SAM_NODE_FILE_ENABLE in the file control block (FCB) before it opens the file, to indicate that it is capable of processing SAM node files correctly.

23 UPAM - User Primary Access Method

UPAM is the primary, block-oriented access method in BS2000 for random access to disk files. Read or write access to any block of a file is possible at any time.

The access method UPAM is capable of processing files with different block formats (see section “Block formats for disk files”, and section “Block formats for tape files”):

- K-PAM files (Key PAM files) have the block format “PAMKEY”: They are characterized by the fact that DMS management information for each PAM page is kept in a separate PAM key located outside the page.
- NK-PAM files (non-key PAM files), have the block format “DATA” or “NO”. With the block format “DATA”, the DMS management information is held in a block control field within the data block. No such block-specific management information is stored for the block format “NO”. NK-PAM files are classified into NK2-PAM files and NK4-PAM files. For more information on these files, see section “Block formats for disk files”.

By way of the BLKCTRL operand in the macros FILE and FCB, the user can select whether a K file or an NK file is to be processed: BLKCTRL=PAMKEY defines a K-PAM file, BLKCTRL=DATA or BLKCTRL=NO specifies an NK-PAM file.

At command level, the user can use the BLOCK-CONTROL-INFO operand of the ADD-FILE-LINK command to specify whether a K or an NK file is to be processed: BLOCK-CONTROL-INFO=*PAMKEY defines a K-PAM file, BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK or BLOCK CONTROL-INFOR=*NO defines an NK-PAM file.

If the block size is specified as BLKSIZE=(STD,n) or BUFFER-LENGTH=*STD(SIZE=n), and n is even, an NK4-PAM file is created; if n is odd, an NK2-PAM file is created. NK4 disks, by contrast, can only be used for an NK4-PAM file (see section “Disk formats”).

The basis for file processing is a standard block; record structures are not supported. The (logical) data block for K-PAM files (block format BLKCTRL=PAMKEY) and NK-PAM files (block format BLKCTRL=DATA or NO) consists of one or more 2048-byte standard blocks (BLKSIZE=(STD,n)).

In both cases, multiple 2048-byte standard blocks can be combined to form a single data block (logical block) by specifying BLKSIZE=(STD,n) (with n > 1) in the FCB/FILE macro or BUFFER-LENGTH=*STD(SIZE=n) in the ADD-FILE-LINK command.

PAM files of the type node file always have the block format BLKCTRL=NO.

In the case of a K-PAM file, each standard block within the logical data block can be addressed in the program. For an NK-PAM file, it is only possible to address a complete data block (logical block) from within a program; separate processing of the individual 2048-byte blocks which make up the data block is not possible.

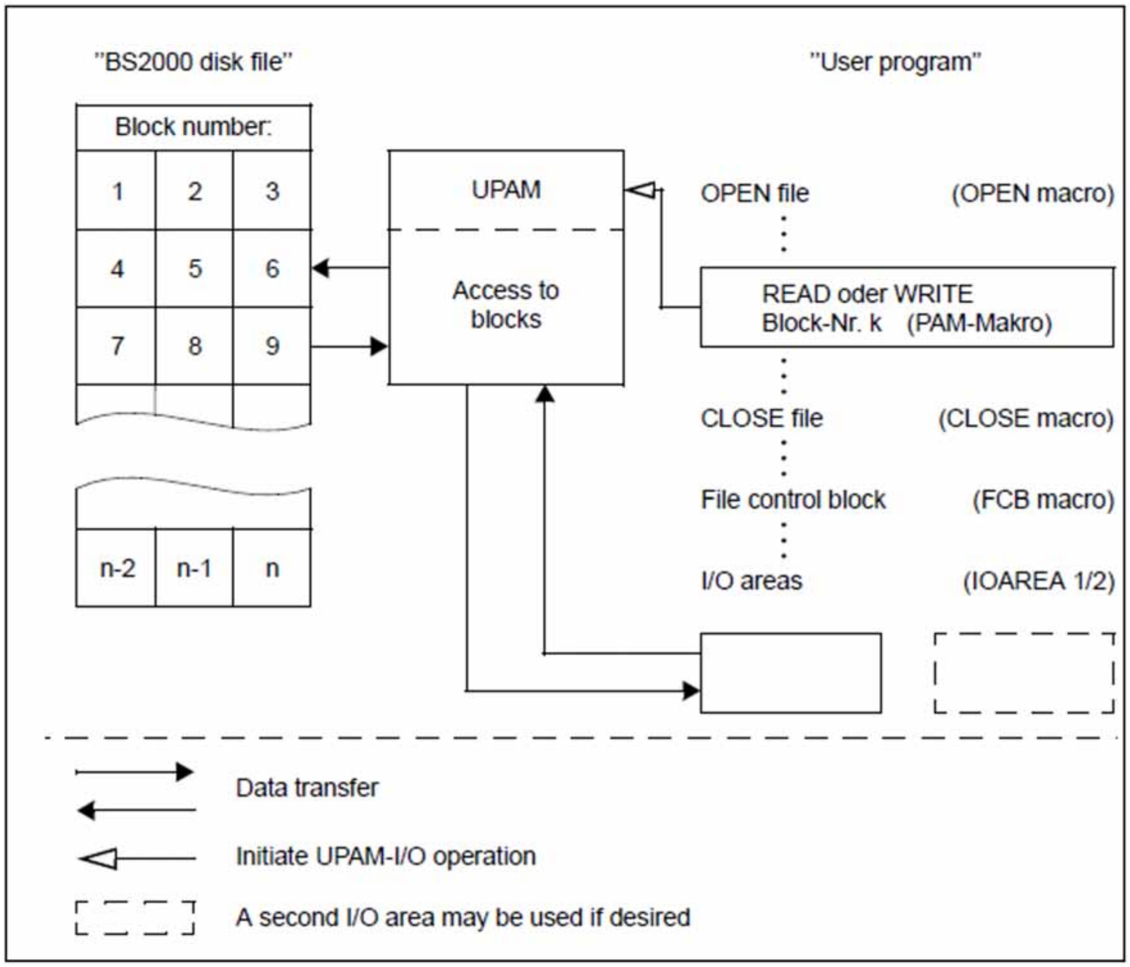


Figure 39: Principles of UPAM operation

23.1 Data blocks

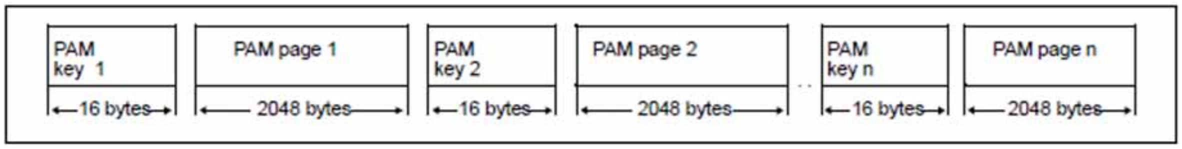
- Data block in a K-PAM file
- Logical block in an NK-PAM file

23.1.1 Data block in a K-PAM file

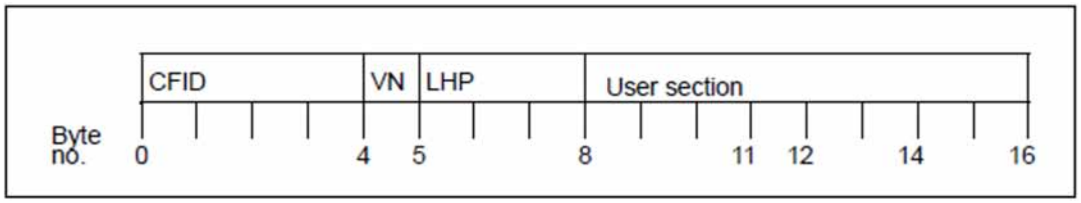
Each PAM page of a conventional K-PAM file has a 16-byte PAM key, located outside the page, in which DMS management information is kept.

Specifying the operand BLKSIZE=(STD,n) in the FILE or FCB macro or the operand BUFFER-LENGTH=*STD(n) in the ADD-FILE-LINK command enables several PAM pages to be combined to form one data block. However, even here, file processing with UPAM is based on the PAM page, i.e. it is always individual PAM pages that are read or written.

In a K-PAM file in which n PAM pages have been grouped together by means of the operand BLKSIZE=(STD,n) or BUFFER-LENGTH=*STD(n) to form one data block, this block has the following structure:



All n * 2048 bytes of the data block are available for user data, since the DMS management information is stored in the 16-byte PAM key. The PAM key has the following structure:



where:

CFID	Coded File IDentification: the encrypted name of the file to which the PAM page belongs (or last belonged). By comparing this with a corresponding CFID in the catalog entry for the current file, DMS can determine whether the data in the PAM page belongs to the file or whether the page is reserved (for this file) but does not yet contain data.
VN	Version number: the version number of a PAM page is updated or evaluated by DMS every time the file is created, modified, set up or deleted. It permits, among other things, partial backup of a file by means of ARCHIVE or HSMS. This is a save run in which only those PAM pages of a file which have been modified since the last partial or incremental backup are saved.
LHP	Logical Half-Page number: this is used for consistency checking for files and for decompression when a file is recovered by ARCHIVE or HSMS from a backup copy.
User section	This part of the PAM key is not used by UPAM and is available to the user. However, when processing SAM and ISAM files with UPAM, it should be noted that SAM and ISAM use this field.

i To ensure that it is possible to convert K-PAM files to NK-PAM files, UPAM users are advised not to use the PAM key.

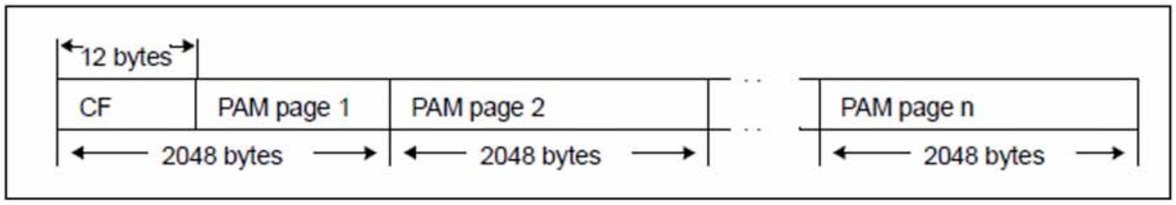
23.1.2 Logical block in an NK-PAM file

In contrast to K-PAM files, NK-PAM files do not have separate PAM keys outside the PAM pages. Depending on the block format selected for an NK-PAM file, the block-specific management information is either kept within the logical blocks or does not exist:

Block format "DATA"

This block format is generated when a file is created with the operand BLKCTRL=DATA of the FILE macro or the operand BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK of the ADD-FILE-LINK command. The block-specific management information is kept in a part of the logical block of the file, namely in the 12-byte block control field.

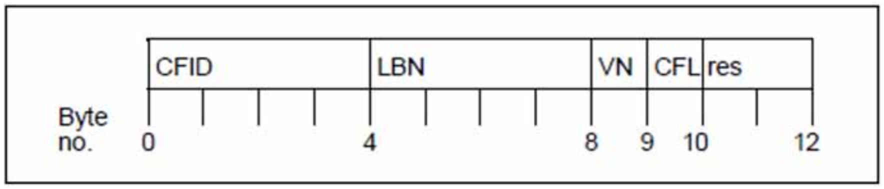
In an NK-PAM file in which n 2-Kbyte blocks (standard blocks) have been grouped together using either the operand BLKSIZE=(STD,n) in the FILE or FCB macro or the operand BUFFER-LENGTH=*STD(n) in the ADD-FILE-LINK command to form a logical block, this block has the following structure:



where:

CF Block control field, 12 bytes.

The block control field has the following structure:



where:

CFID	Coded File IDentification: this has the same meaning as in the PAM key (see "Data block in a K-PAM file").
LBN	Logical Block Number: number of the standard block which contains the block control field. (Standard blocks are numbered sequentially within a file.)
VN	Version number: this has the same meaning as in the PAM key.
CFL	Control FLag: the control flag flags gaps within a file. This information is used primarily when copying a file from or to a tape.
res	Bytes in the block control field which are reserved for future use.

The 12 bytes of a logical block used for the block control field are thus not available for storing user data.

The length of the part of a logical block that can be used for user data, consisting of n standard blocks, is therefore:

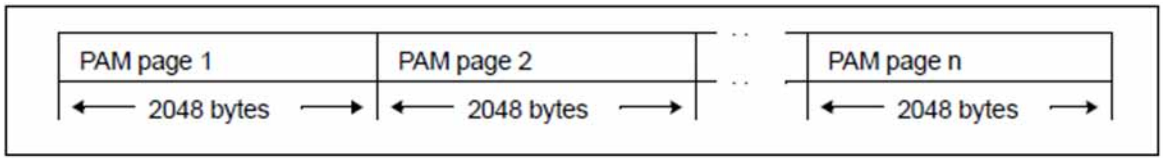
Usable block length:

```
    Maximum data length = n * 2048 – 12
```

Block format "NO"

This block format is generated when a file is created with the operand BLKCTRL=NO of the FILE macro or the operand BLOCK-CONTROL-INFO=*NO of the ADD-FILE-LINK command. No block-specific management information is generated. The entire logical block is available for storing user data.

In an NK-PAM file in which n 2K blocks (standard blocks) have been grouped together using either the operand BLKSIZE=(STD,n) in the FILE or FCB macro or the operand BUFFER-LENGTH=*STD(n) in the ADD-FILE-LINK command to form a logical block, this block has the following structure:



Since the entire logical block is available for user data, the usable block length is n * 2048 bytes (for a logical block which consists of n standard blocks).

Block format “NO” is particularly suitable for files which are to contain an uninterrupted stream of data, extending beyond logical block boundaries (e.g. load modules). System functions which need block-specific management information from the PAM key or from the block control field (e.g. partial backups using ARCHIVE or HSMS) cannot be used for such files.

23.2 UPAM file formats

UPAM is a block-oriented access method. The basic processing unit is the 2-Kbyte standard block for K-PAM files and the logical block for NK-PAM files, its size being determined by the BLKSIZE operand in the FCB and FILE macro or the BUFFER-LENGTH operand in the ADD-FILE-LINK command. UPAM can read in or output up to 16 2-Kbyte standard blocks at the same time. This number is defined with the LEN operand in the PAM macro. If LEN=(STD,n) or LEN=n*2048 is specified, then n <= 16.

The following point applies to K-PAM files:

If the value specified for the LEN operand in the PAM macro is not an integer multiple of 2048, it is rounded up to the next higher integer multiple of 2048.
In the case of a write operation, the remainder of the last PAM page to be written in the file is undefined. If this write operation took place at the end of the file and this file is then closed, the position of the last valid byte on this PAM page is stored in the catalog entry's last-byte pointer.
For a read operation, the remainder of the last PAM page to be read is not passed to the buffer.

The following point applies to NK-PAM files:

If the value specified for the LEN operand in the PAM macro is not an integer multiple of the size of a logical block, it is rounded up to the next higher integer multiple of the logical block size.
In the case of a write operation, the remainder of the logical block in the file is undefined. If this write operation took place at the end of the file and this file is then closed, the position of the last valid byte of the last logical block is stored in the catalog entry's lastbyte pointer.
For a read operation, the remainder of the last logical block to be read is not passed to the buffer and the remainder of the buffer contents is undefined.

A last-byte pointer with the value 0 means that the last logical block of the file is completely valid (for exceptions see the REPAIR-DISK-FILE command and VERIF macro).

A last-byte pointer with the value 0 means that the last logical block of the file is completely valid (for exceptions see the REPAIR-DISK-FILE command and VERIF macro).

The SHOW-FILE-ATTRIBUTES command outputs the value of the last-byte pointer in the allocation part as LAST-BYTE or S variables if the corresponding valid bit has been set in the catalog entry of the file. The last-byte pointer points to the last valid byte of the last logical block in a file.

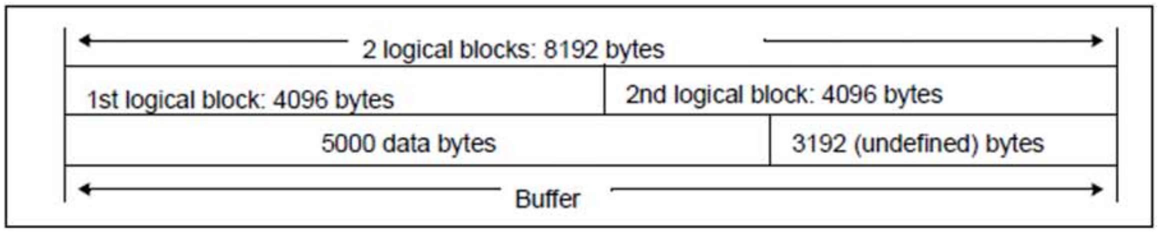
i In case of problems, the customer service may be able to reset the last-byte pointer.

Prerequisite for the usage of the last-byte pointers for PAM files on public space is the setting of the LBP_REQUIRED indicator in the file control block (FCB) for OPEN. LBP_REQUIRED makes the LBP in the FCP accessible during OPEN and saves the updated value in the catalog entry during CLOSE.

With PAM node files, a last-byte pointer is always supplied, irrespective of the LBP_REQUIRED flag. However, it is to be noted that the LBP in the CE may be outdated, as the file might have been changed by another system. Therefore, the current valid lastbyte pointer is identified and made accessible to the application in the file control block (FCB) during OPEN.

Example

The operands WRT and LEN=5000 are specified in a PAM call for a file with BLKCTRL=NO and BLKSIZE=(STD,2) (which is equivalent to BLOCK-CONTROL-INFO=*NO and BUFFER-LENGTH=*STD(SIZE=2) in the ADD-FILE-LINK command). 5000 bytes are transferred from the buffer, and the remainder up to the next highest integer multiple of the logical block size (8192 bytes) is undefined.


2 logical blocks: 8192 bytes
1st logical block: 4096 bytes
2nd logical block: 4096 bytes
5000 data bytes
3192 (undefined) bytes
Buffer

If the write operation in the example took place at the end of the file and the file was then closed, the last-byte pointer has the value 904. Consequently 904 bytes were written in the 4th and at the same time last logical block of the file. The last-byte pointer, like the last-page pointer (here LPP=8), is noted in the catalog entry when the file is closed and is used by the application to ascertain the precise end of file after the file has been opened again.

23.2.1 Converting K-PAM files into NK-PAM files

With the aid of the COPY-FILE command or the COPFILE macro, K-PAM files can be converted into NK-PAM files with the block format “NO” (see the example below) subject to the condition that the user section of the PAM key is not used i.e. has the value XL8'00'. The software product PERCON (see the “PERCON” manual [12 (Related publications)]) can be used in order to convert a K-PAM file into an NK-PAM file with the block format “DATA”.

Example: Converting a K-PAM file into an NK-PAM file (block format “NO”) using the COPY-FILE command

```
/ADD-FILE-LINK LINK-NAME=DMCOPY22,FILE-NAME=nk-pam-datei,
                 BLOCK-CONTROL-INFO=*NO
/COPY-FILE FROM-FILE=k-pam-datei,TO-FILE=nk-pam-datei,
                 BLOCK-CONTROL-INFO=*IGNORE-ATTRIBUTE
```

K-PAM files can also be converted to NK files with the block format "NO" using the PAMCONV utility routine or by saving and restoring with HSMS/ARCHIVE.

If existing applications do not require the PAM key, it is recommended to convert to NK-PAM files on NK disks to benefit from the better performance and higher utilization of disk space.

23.2.2 Converting NK-PAM files into K-PAM files

NK-PAM files can be converted into K-PAM files with the aid of the PAMCONV utility routine (see the “Utility Routines” manual [13 (Related publications)]) or the software product PERCON (see the “PERCON” manual [12 (Related publications)]).

23.2.3 Compatibility of file formats

NK-PAM files offer the user the same functional scope for processing as K-PAM files.

However, NK-PAM files do not have a field comparable with the user section of the PAM key. Applications which use this part of the PAM key in K-PAM files must therefore be modified if they are to process NK-PAM files.

NK-PAM files with the block format “DATA” are subject to the following restriction with regard to *parallel processing*: When processing a PAM file with BLKCTRL=DATA, the IOAREAs may not be used in parallel for several I/O operations, as this would mean that the contents of the block control field would be undefined.

Example

- Illegal approach:

  ```
  PAM WRT,FCB1,LOC=BUFFER

  PAM WRT,FCB2,LOC=BUFFER              (Parallel I/Os for BUFFER)

  PAM WT,FCB1

  PAM WT,FCB2
  ```

- Permitted approach:

  ```
  PAM WRTWT,FCB1,LOC=BUFFER            (Consecutive I/Os for BUFFER)

  PAM WRTWT,FCB2,LOC=BUFFER
  ```

When working with NK-PAM files with the block format “DATA”, users should bear in mind that the usable block length is less than that available with K-PAM files: a logical block of n standard blocks provides only n * 2048 – 12 bytes for user data.

NK-PAM files with the block format “NO” do not contain a block control field, which means that they do not contain any information corresponding to that held in the system section of the PAM key in a K-PAM file. As a result, partial backup of such files using ARCHIVE or HSMS is not possible. Furthermore, when restoring such files with the aid of the REPAIR-DISK-FILE command or VERIF macro, the files must be processed (without gap testing) up to the end of the file, rounded down to a multiple of the block size. The practical effect of this is that the last-page pointer and possibly the last-byte pointer are set to the end of the file.

In an NK-PAM file with the block format “NO” – as in K-PAM files – the entire length of each logical block is available for user data. In contrast to K-PAM files, however, only a complete logical block can be accessed during file processing; it is not possible to access each separate 2-Kbyte standard block of a logical block.

23.3 Opening a PAM file (OPEN modes)

The following table provides an overview of the OPEN modes.

Open mode	Brief description
INPUT	Read blocks from an existing file.
OUTIN	Create a new file and, if required, read blocks from this file.
INOUT	Read blocks from an existing file and, if required, add or exchange blocks.

Table 70: Open modes when opening a UPAM file

The table below provides information on which functions of the PAM macro are permitted in conjunction with the above-mentioned open modes.

PAM macro functions	OPEN mode		
	INPUT	OUTIN	INOUT
RD, RDWT, RDEQU, LRD, LRDWT	X	X	X
WRT, WRTWT, WRTWU	-	X	X
WT, CHK, SYNC	X	X	X
LOCK, UNLOCK, SETL	X	X	X
SETLPP	-	X	X

Table 71: UPAM functions and permissible open modes

23.4 UPAM processing of disk files

The following UPAM functions are available for disk files:

Creation of disk files

Users themselves must program access to records (e.g. sequential access or associative access using hashing techniques).

Reading and transfer of SAM and ISAM files

(OPEN INPUT) reading of SAM and ISAM files and their transfer to other volumes (e.g. from disk to tape): the file attributes are defined in the FCB (e.g. BLKSIZE, RECSIZE, RECFORM) or in the ADD-FILE-LINK command (BUFFER-LENGTH, RECORD-SIZE, RECORD-FORMAT). This enables the user to program access to records.

UPAM cannot open SAM or ISAM files in UPDATE mode.

Because of the complex relationships between index blocks and data blocks, ISAM files cannot be effectively processed using UPAM. UPAM can, however, be used for the block-oriented transfer of ISAM files to tape.

Chained I/O

Up to 255 logically consecutive PAM pages of a file can be input or output by means of a single PAM macro. The maximum value depends on the disk attributes (see SHOW-MASTER-CATALOG-ENTRY, I/O length).

PAM macros in list form

Up to 255 PAM macros (not necessarily all referring to the same file) can be handled with a single UPAM I/O request, i.e. only one SVC is required. The chaining of PAM macros (just like chained I/O) serves to optimize the runtime performance of user programs.

Eventing mechanism

The user job is notified when a UPAM I/O operation terminates and a contingency process starts.

DRV

For files with DRV (Dual Recording by Volume; see the “DRV” manual [14 (Related publications)]), user information is output on the current status (e.g. loss of a copy of a disk). This information is requested by UPAM when an I/O operation is performed, and stored in the FCB (field ID1DRVST). However, this field is not updated unless the DRV status is modified.

Shared-update processing (multi-user mode)

A number of parallel jobs can process a PAM file concurrently.

Disk file without a PAM key

A file without a PAM key is defined by means of the operands BLKCTRL=NO / DATA in the FILE/FCB macro or with the operands BLOCK-CONTROL-INFO=*NO / *WITHIN-DATA-BLOCK in the ADD-FILE-LINK command. The following points should be kept in mind:

- The file must have standard blocks (BLKSIZE=(STD,n) or BUFFER-LENGTH=*STD(SIZE=n))
- If the file is not an ISAM file (FCBTYPE=SAM/PAM or ACCESS-METHOD=*SAM/*UPAM), the secondary allocation must be at least as large as the defined block size (BLKSIZE or BUFFER-LENGTH).

The following applies to the UPAM access method:

- If SHARUPD=*WEAK is specified, a file can be opened in read mode even if it has been opened in write mode by another system in the multi-system environment. However, a file extension (secondary allocation) initiated by the other system at a point in time *after* the file is opened locally will be ignored. Any file sections generated at this later time cannot be read by the local user.
- Illegal combinations result in an OPEN error.
- If a PAM file is to be processed simultaneously by a number of different jobs, the operand SHARUPD=YES or SHARED-UPDATE=*YES must be specified for each job in the FCB macro or in the ADD-FILE-LINK command, respectively.

  In UPAM, there is no implicit locking/unlocking of PAM pages. The user must explicitly request and release every lock himself/herself. The only exception is the CLOSE macro: before the file is closed, any PAM pages still locked in the file by this job are unlocked.

- Locking a PAM page does not prevent other jobs from reading it or writing to it. It merely prevents it from being locked once more by another job. Therefore, all jobs participating in shared-update mode should observe the following processing sequence: `Sperren (LOCK) -> Read(RD) -> Write(WRT) -> (UNLOCK)`

  The entries in parentheses indicate the operands of the PAM macro.

  The user should not attempt to request other resources for exclusive use while he/she has PAM pages locked, since this could lead to a deadlock situation (EXLST exit DLOCK).
- One job must not have more than 255 PAM pages locked at the same time.
- The same PAM page can only be locked once within a given UPAM operand list chain.
- UPAM examines the FCB field ID1TOUT (value of the PAMTOUT operand) whenever a lock is requested. Thus the user program can amend this value as and when required, as long as the file remains open. If several locks are requested when PAM macros are chained together, the PAMTOUT value in the FCB referenced by the first lock request applies to all the locks requested in this “chain”.
- If a lock is not available immediately, the job is queued for the length of time specified in the ID1TOUT field. If the requested lock is still not available after this time, the DLOCK or PGLOCK exit of the EXLST macro is taken, depending upon whether or not there are PAM pages locked by the job when the timeout occurs.
- Any LOCK or UNLOCK operation applied to a file previously opened with SHARUPD=*NO or SHARUPD=*WEAK is treated as a no-op; all that happens is that the file pointer is updated.
- If a PGLOCK is encountered, the program can continue normally, e.g. make a new attempt to lock the page after a waiting time. A DLOCK means that a multiple lock has at least been attempted, if not imposed, causing the program to become unstable. As long as it is in this unstable state, any attempt to lock a PAM page before all current locks are released will lead to abnormal termination of the program. Only when all current locks have been released, either in or after the routine for the DLOCK exit, will the program become stable again and be able to request locks in the usual way.

- UPAM can also be used to access SAM node files. For write access (OPEN OUTIN), the additional bit SAM_NODE_FILE_ENABLE must be set in the FCB. By default, UPAM, when reading from SAM node files, recieves SAM blocks of the same structure as from public space. Likewise, UPAM has to write data blocks with the SAM structure.It is, however, possible to work without converting the data from UFD to the SAM block format: The so-called RAW access enables a direct access to the data of the node file. The RAW access is activated by setting the UPAM_RAW_ACCESS indicator in the FCB before OPEN processing. Dynamic switching between block-based processing and RAW processing is not possible.
  If data are written to somewhere other than the end of the file, it is important in both cases (block-oriented with conversion or in RAW mode without conversion) to ensure that the writing (user) data are of the exact same length as the data to be overwritten. Disregarding this may lead to loss of data. It is also important to note that, when accessing data in RAW mode, UPAM addresses the data in multiples of 2 kB and therefore generally without considering the start and end of the record as well as the record length. The application is responsible for the correct processing of the data.

23.4.1 Chained I/O

Chained I/O permits the simultaneous input/output of up to 255 logically consecutive PAM pages using a single PAM macro (not to be confused with the chaining of PAM macros in list form with the CHAIN operand). This reduces the number of I/O operations (and interrupts) and thus reduces processing time. On the downside, however, it increases main memory requirements and the paging rate.

UPAM uses chained I/O if the LEN operand of the PAM macro contains a value greater than STD or greater than 2048.

End-of-file (EOF) processing

If the end-of-file condition is encountered during a write operation, a secondary allocation is performed and the specified number of PAM pages are appended to the file.

If the end-of-file condition is detected during a read operation, UPAM merely transfers the PAM pages belonging to the file into the buffer.

UPAM notifies the calling job of end-of-file processing as follows:

- *If eventing is not used:*
  The user job receives control at the EXLST exit USERERR with error code X'0922' in field ID1ECB of the FCB. The number of PAM pages transferred is contained in field ID1NBPP of the FCB. The value X'00' in this field means that all the PAM pages to be read are located outside the file. A value greater than X'00' means that the user job must perform a wait operation, unless this was implicitly included in the read operation (i.e. in a RDWT operation).
- *If eventing is used:*
  If all the PAM pages to be read are located outside the file, UPAM passes control to the user job at the EXLST exit USERERR with error code X'0922' in field ID1ECB of the FCB. If at least one of the PAM pages to be read belongs to the file, UPAM either resumes the basic task or initiates a contingency process (see section “TU eventing: event-driven processing”). Field IDECBNPA of the FECB (File Event Control Block, see "TU eventing: event-driven processing") now contains a value indicating the number of PAM pages transferred.

X'00'	All PAM pages to be read were transferred to the buffer.
X'0n'	n = number of PAM pages belonging to the file that were transferred to the buffer.

Locking and unlocking of PAM pages

Only the first in a series of PAM pages to be locked/unlocked needs to be specified in a PAM macro; the number of pages to be locked/unlocked is derived from the value of the LEN operand. It should, however, be noted that after a lock or unlock operation, the file pointer points to the last PAM page that was locked/unlocked. This page may lie outside the file (see “End-of-file processing”, previous page).

A LOCK or UNLOCK operation applied to a file opened as SHARUPD=*NO or SHARUPD=*WEAK is a no-op: the only action taken is to update the pointer to indicate the last page processed, and the LEN operand is evaluated.

Processing of PAM keys

There are two possible ways of processing PAM keys:

- The user reads/writes each individual key of a series of PAM pages: MKEY=YES operand in the PAM macro; the KEYFLD operand must contain the address of a sufficiently large area.

- The user reads/writes only the first key of a series of PAM pages. During writing, succeeding blocks are assigned the same key as the first, with just the logical block number being incremented by 1 each time.

Notes

As blocks are not transferred to the user buffer until an explicit action macro is issued, this causes a delay. Consequently asynchronous I/O operations have to be terminated by means of the WT action macro. For TU eventing, the SOLSIG macro should be used (synchronously or asynchronously).

Any unsuccessful branch to the UPAM routines causes control to be passed to the routine specified in the EXIT operand of the FCB macro, or to the corresponding EXLST routine. An indicator is stored in the FCB.

Whenever UPAM induces a program termination, it supplies registers 0, 1 and 15 with the following contents, which can be easily identified and evaluated in memory dumps:

Register	Contents
0	Address at which the termination occurred
1	Address of the element in the UPAM operand list chain in which the error was detected.
15	UPAM error code

If register 1 contains an invalid address when the PAM macro is issued for the first time, this address will be contained in register 0 when the dump is taken, and register 1 will contain zeros (i.e. the error occurred before the first element in the operand list chain was located).

UPAM uses the following EXLST exits:

EXLST exit	Brief description
EXLST exit	Brief description
ERRADDR	Hardware fault or abnormal I/O termination.
USERERR	Invalid use of a macro in the program or attempt to read a PAM page not belonging to the file (EOF).
EOFADDR	Attempt to read a dummy file.
PGLOCK	Not all of the requested locks become available within the specified time, and the job currently has no blocks locked.
DLOCK	The request to set up a lock is rejected and the job already has locks set.

PAM pages which have been allocated to a file but have not yet been written by the file owner are identified by an internal file name code assigned by the system (CFID = Coded File ID, bytes 0-3 of the PAM key or block control field), which does not correspond to the current file name. The name comparison must be performed by the user. Thereby making sure (see also the KEYFLD operand of the PAM macro) that at OPEN time the current CFID is written to the first word of field ID1KEY1 in the FCB.

The following points apply solely to the processing of K-PAM files (BLKCTRL=PAMKEY or BLOCK-CONTROL-INFO=*PAMKEY):

- After execution of a RDWT, LRDWT or RDEQU operation, the CFID of the block just read is located in the first word of FCB field ID1KEY2.
- After execution of one of the operations WRT, WRTWT, WRTWU or WT, the CFID of the PAM page concerned is located in the first word of FCB field ID1KEY1. As the entry generated by OPEN will be overwritten, it should be saved prior to processing so that subsequent comparisons can be made.
- After execution of an LRD or RD operation, the contents of fields ID1KEY1 and ID1KEY2 are not changed.
- With event-driven processing, the CFID of the relevant PAM page is in the first word of FCB field ID1KEY1 after completion of an I/O operation.

The fields ID1LWB (PARMOD=24) and ID1LWBPT (PARMOD=31) in the FCB are used to store the address of the last block on which UPAM successfully performed an I/O operation. The leftmost byte of field ID1LWB is used as an indicator for an outstanding WT operation.

If the last UPAM operation on the file was a successful WT, then the indicator byte in ID1LWB is set to X'00' and the three least significant bytes of ID1LWB/ID1LWBPT contain the address of the block affected by the WT operation. This occurs regardless of whether or not the operation which initiated the WT was successfully completed.

If the last UPAM operation on that file did not initiate a WT, the indicator byte in ID1LWB is set to X'FF'. The contents of the remaining three bytes in ID1LWB or ID1LWBPT are of no significance.

23.5 UPAM processing of tape files

The following UPAM functions are available for tape files:

Creation of tape files

UPAM creates tape files that do not extend over more than one tape. The user is responsible for programming access to logical records in these files.

Reading of SAM files with standard blocks

The file attributes are defined in the FCB (e.g. BLKSIZE, RECSIZE, REC-FORM) or by the BUFFER-LENGTH, RECORD-SIZE, RECORD-FORMAT operands in the ADD-FILE-LINK command by OPEN processing (see section “Sequence of OPEN processing”). This enables the user to program access to logical records.

Chained I/O

Not possible for tape files.

Eventing mechanism

The user job is notified when a UPAM I/O operation terminates and a contingency process starts.

Notes

The following points apply to the use of UPAM with tape files:

- The file must reside on a single tape.
- Every read/write operation processes exactly one physical block.
- In the case of a file with the format BLKCTRL=PAMKEY or BLOCK-CONTROL-INFO=*PAMKEY (explicit or implicit), this must be a 2064-byte standard block: the first 16 bytes contain the PAM key and the remaining 2048 bytes contain the user data. Only STD or 2048 may be specified for the LEN operand of the PAM macro.
- In the case of a file with the format BLKCTRL=DATA / NO (which is equivalent to BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK / *NO in the ADD-FILE-LINK command), BLKSIZE (or BUFFER-LENGTH) can be specified in bytes, in which case the value must be a multiple of 2048. The LEN value in the PAM macro must not be greater than the BLKSIZE value. Blocks of the length specified by LEN and containing either only user data (with BLKCTRL=NO) or a 12-byte block control field plus the (LEN – 12) bytes of user data (with BLKCTRL=DATA) are written to the tape. These blocks can be read by means of the UPAM function RD, in which case the value for LEN_{RD} in the read call must lie within the range:

  $\text{LEN}_{\text{WR}} <= \text{LEN}_{\text{RD}} <= \text{BLKSIZE}$

  (LEN_{WR}: value for LEN when writing to the file)

  In this context, the following special feature applies to files with FCBTYPE=ISAM or ACCESS-METHOD=*ISAM: regardless of the value specified for BLKSIZE, UPAM always works with a block size of 2048 bytes. This is because, for ISAM, each 2K block has a block control field and thus constitutes a separate entity. In this case, the LEN value must not be more than 2048.
- The following errors may arise in the context of a UPAM access to a tape file with BLKCTRL=PAMKEY and blocks which deviate from standard blocks (2064 bytes):
  - hardware error (error code = 927)
  - the data may be stored in the buffer incorrectly
  - the PAM key may be corrupted.

- Tape files may not be accessed by several jobs at once, or by one job several times (this is due directly to the properties of the magnetic tapes). This means:
  - a tape file must not be opened with SHARUPD=YES/WEAK
  - a tape file which is already open cannot be opened again, even when both OPEN operations are in INPUT mode
  - a tape file must not be opened with an FCB operand whose PAMREQS value is greater than one.
- It is possible to read a tape file with UPAM using random access. However, the time outlay involved can be considerable.
- It is possible to write PAM blocks as of a specific point in an existing tape file. The last newly written block automatically becomes the last block in the file, even if the existing file contained more blocks. The file can then only be read up to this block. If a block nearer the start of the file is then read and the file is then closed, the most recently read block becomes the last block in the tape file.

Programming notes

Whenever UPAM induces a program termination, it supplies registers 0, 1 and 15 with the following contents, which can be easily identified and evaluated in memory dumps:

Register	Contents
0	Address at which the termination occurred
1	Address of the element in the UPAM operand list chain in which the error was detected
15	UPAM error code

If register 1 contains an invalid address when the PAM macro is called for the first time, this address will be contained in register 0 when the dump is taken. Register 1 will then contain the value 0. I.e. the error occurred before the first element in the operand list chain was located.

UPAM uses the following EXLST exits:

EXLST exit	Brief description
ERRADDR	Hardware fault or abnormal I/O termination.
USERERR	Invalid use of a macro in the program or attempt to read a PAM page not belonging to the file (EOF).

Fields ID1LWB (PARMOD=24) and ID1LWBPT (PARMOD=31) in the FCB are used to store the address of the last block on which a WT operation was carried out successfully by UPAM. The leftmost byte of field ID1LWB is used as an indicator byte.

If the last UPAM operation on the file was a successful WT, then the indicator byte in ID1LWB is set to X'00' and the three least significant bytes of ID1LWB/ID1LWBPT contain the address of the block affected by the WT operation. This occurs regardless of whether or not the operation which initiated the WT was successfully completed.

If the last UPAM operation on that file did not initiate a WT, the indicator byte in ID1LWB is set to X'FF'. The contents of the remaining three bytes in ID1LWB or ID1LWBPT are of no significance.

UPAM treats magnetic tape cartridges just like normal magnetic tapes.

23.6 Chaining PAM macros in list form

PAM macros which are to be chained, and which do not necessarily have to refer to the same file, must be generated in list form by means of the operand MF=L and then stored in a constant area; chaining is implemented by specifying the CHAIN operand.

All the macros (except the last) have the following format:

	Operation	**Operands**
element$_n$	PAM	fcbadr,operation,...,MF=L,CHAIN=element$_{n+1}$

In the last element of the chain, the CHAIN operand is omitted.

A chain of PAM macros in list form is called by means of a PAM macro with the following format:

	Operation	**Operands**
	PAM	MF=(E,element$_1$)

Only one SVC instruction is required for each chain, i.e. by chaining UPAM requests the user avoids the overhead of multiple SVC processing. The elements of a chain do not necessarily have to refer to the same file.

Example of coding

```
START
        LDBASE 10
        USING *,10
        .
        .
        PAM MF=(E,ELEM1)
        .
        .
        .
        TERM
*CONSTANT AREA
ELEM1   PAM     ......,MF=L,CHAIN=ELEM2
ELEM2   PAM     ......,MF=L,CHAIN=ELEM3
ELEM3   PAM     ......,MF=L
        .
        .
        .
        END
```

If execution is successful, the user regains control at the statement following the PAM macro which requested processing of an operand list chain.

All operations are carried out in exactly the same order in which the PAM macro lists appear within the chain – with one exception: when the first operation to request a lock is encountered, the rest of the chain is examined and all operations requesting locks are registered. If all the requested locks cannot be imposed within the specified time, the chain is terminated at the operation which requested the first lock.

If an action requested in the chain is not performed successfully, none of the following actions in the chain is executed (including locks). Control is passed to the appropriate EXLST exit; register 1 points to the FCB of the file in which the error occurred. The error code (ID1ECB) and the sense byte (ID1XITB) are set in this FCB in the normal way. The ID1CHERR field in the FCB is set to the address of the element in the operand list chain that caused the error.

A check on an unfinished I/O operation also causes control to be transferred from the operand list chain to the user program at the specified address. The remaining requests in the chain (including locks) are not executed, and field ID1CHERR in the FCB is set to the address of the operand list element which contains the CHK operation. It is therefore not advisable to use check operations in operand list chains.

The user must ensure that the lock, read, write, wait, check and unlock operations are applied appropriately within a chain. Buffers and key fields are utilized by UPAM according to the request. A check is made to verify that a buffer exists and that access is authorized, but there is no guarantee that a buffer or a key field filled by an operation in a chain will not be overwritten by a later operation involving this chain.

The format of the PAM operand list can be described by means of a dummy program section (DSECT) generated with the IDPPL macro.

In all cases where a chain of UPAM operand lists cannot be processed in full (e.g. I/O operation failed, error detected, lock not accepted, EOF condition encountered, or CHK operation attempted on currently executing I/O request), the address of the first unexecuted chain entry is moved to the ID1CHERR field of the FCB. The user can assume that all preceding entries were executed correctly.

UPAM does not report errors via FCB-EXIT and EXLST

- if the UPAM SVC is executed and neither register 1 nor any chain operand contains a valid address (e.g. the address is not aligned on a word boundary or refers to a field which is not entirely within the user's virtual address space and is not large enough to accommodate a UPAM macro operand list, etc.)
- if the FCB address in the UPAM macro operand list is either missing or invalid.

In both these cases, there is no FCB to which the error could be reported, and therefore UPAM aborts the program.

A UPAM operand list chain is validated in full before any function is initiated. If a CHAIN address or FCB address is found to be invalid, the job is aborted before any chain element is executed.

Where the eventing mechanism is used, the user program is likewise aborted if the I/O operation is completed and there is no event item available to which the event can be reported.

23.7 TU eventing: event-driven processing

The event-driven processing described below and the macros mentioned are described in detail in the "Executive Macros" manual [2 (Related publications)].

Eventing is used by UPAM to report the completion of an I/O request to a job. The job can

- be continued in parallel with the UPAM I/O operation and, when the expected event occurs (in this case, termination of the requested I/O operation), proceed with a contingency process (asynchronous processing).
- wait for termination of the requested I/O operation and then proceed (synchronous processing; this is, of course, equally feasible without eventing).

Upon completion of an I/O operation, UPAM sends a message to the associated event item (using the POSSIG macro). Sooner or later this message encounters the request issued by the user (by means of the SOLSIG macro). When both request and message are present (for the same event item), a contingency process is started or the basic task resumed.

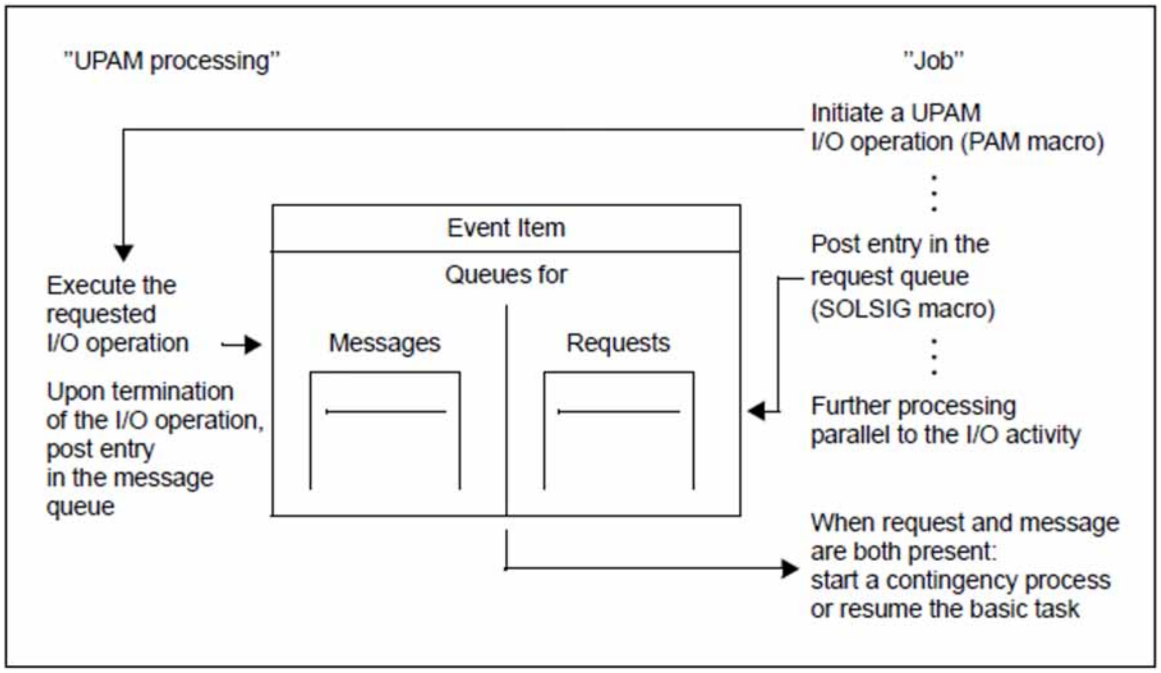

"UPAM processing"
"Job"
Initiate a UPAM I/O operation (PAM macro)
Event Item
Queues for
Messages
Requests
Execute the requested I/O operation
Upon termination of the I/O operation, post entry in the message queue
Post entry in the request queue (SOLSIG macro)
Further processing parallel to the I/O activity
When request and message are both present: start a contingency process or resume the basic task

Figure 40: Coordination of user job and UPAM processing

Basic task

The system must be informed of the event items and contingency definitions that are to be used (ENAEI, ENACO macros).

For each I/O operation, the system must be supplied with the address of a file event control block (FECB). I/O operations running in parallel must refer to different FECBs. The maximum number of parallel I/O operations is defined via the FCB operand PAMREQS; for tape files, only PAMREQS=1 may be specified.

The number of contingency definitions depends on whether different procedures are required after execution of I/O operations. If, for instance, the same procedure is to be used in all cases, the contingency definition needs to be coded once only.

A 14-byte FECB (File Event Control Block) must be set up for each event item.

Until the first I/O operation with an FECB is terminated, the FECB must not be used for other I/O operations.

The address of the associated FECB must be specified for each UPAM I/O request (FECB operand in the PAM macro). No wait operations may be requested either explicitly or implicitly by the PAM macro. The instruction sequence read (RD) -> write (WRT) -> wait (WT) for the same block would thus yield an undefined result. The user must wait for the event (I/O termination) before issuing the WRITE call.

After each UPAM I/O request, precisely one request must be issued to the associated event item (SOLSIG macro). The request may also be used to specify whether the basic task is to continue in parallel with the I/O operation or is to wait for it to terminate.

At the start of a contingency process, it is passed the following information via registers 2 and 3:

For PARMOD=31: the I/O operation is initiated using the 31-bit operand list; the required information is transferred in registers 2, 3 and 4:

Register	Information
2	Contains the event information code.
3	The two rightmost bytes contain a POST-CODE supplied by the user at the start of the I/O operation, and the leftmost byte contains an identifier indicating a UPAM event (X'10').
4	Contains the address of the operand list for the operation which has just been completed.

Table 72: Register values for TU eventing (31-bit operand list with UPAM)

For PARMOD=24: the I/O operation is initiated using the 24-bit operand list; the required information is held in registers 2 and 3 (as in earlier versions of BS2000):

Register	Information
2	Contains the event information code.
3	The three rightmost bytes contain the address of the operand list for the operation just terminated, and the leftmost byte contains the value X'10'.

Table 73: Register values for TU eventing (24-bit operand list with UPAM)

If a SOLSIG macro generated with PARMOD=24 or a 24-bit contingency definition is addressed via an operand list created with PARMOD=31, the secondary return code (leftmost byte in register 15) indicates that the sending and receiving lengths are not consistent.

Format of the file event control block (FECB)

The FECB must be aligned on a word boundary. It can be given a symbolic name by means of the IDECB macro.

Meaning of field	Field length (in bytes)	Field name
Internal ID of event item	4	CBEVID
Address of FCB	4	CBP1LNK

Standard device byte	1	CBSDB
Sense bytes	3 x 1	CBSB1, CBSB2, CBSB3
Executive flag byte	1	CBEFB
Number of PAM pages transferred	1	CBNPA

Table 74: Format of the file event control block (FECB)

Executive flag byte

A UPAM I/O operation can be terminated in a number of different ways (EFB = Executive flag byte; see FECB):

Normal I/O termination	AMB=X'80'
I/O operation led to exception condition	AMB=X'C0'
Unrecoverable error (e.g. hardware fault)	AMB=X'A0'

In a contingency process, the user can program appropriate responses to the various ways in which an I/O operation can terminate.

23.8 Multi-user mode

A UPAM file can be created or edited using the access modes below

- UPAM
- FASTPAM (see chapter “FASTPAM – Fast Primary Access Method” (FASTPAM - Fast Primary Access Method))
- DIV (see chapter “DIV – Data In Virtual” (DIV - Data In Virtual))

Note, however, that FASTPAM and DIV can process UPAM files with BLKCTRL=NO only.

Whether a file may be processed by more than one user in parallel depends on the operand values specified for SHARUPD, MODE, LOCKENV and LOCVIEW when the file is opened. The table below shows permissible combinations of opening a file in parallel:

Permissible combinations of opening a file in parallel

			USER B								
	SHARUPD=		***YES**			***NO**			***WEAK**		
		OPEN-Modus	INPUT	INOUT	OUTIN	INPUT	INOUT	OUTIN	INPUT	INOUT	OUTIN
USER A	***YES**	INPUT	X	O		X			X		
		INOUT	O	O					X		
		OUTIN	O	O					X		
	***NO**	INPUT	X			X			X		
		INOUT							X		
		OUTIN							X		
	***WEAK**	INPUT	X	X		X	X		X	X	
		INOUT							X		
		OUTIN							X		

Table 75: UPAM: Permissible SHARUPD/OPEN combinations

X: OPEN erlaubt

O: OPEN permitted only

- if the same block-oriented access method is used by all (either UPAM/FASTPAM or DIV)
- **and** the same value is used by all for the LOCKENV operand (either LOCKENV=*HOST or LOCKENV=*XCS)
- **and** all are running on the same host or, with LOCKENV=*XCS, in an XCS network

Notes

- A file can be opened with SHARUPD=*WEAK for concurrent reading and writing.

  Exception:
  The conditions applying to reading with UPAM/FASTPAM-SHARUPD=*WEAK also apply to reading with DIV-SHARUPD=*WEAK after specifying OPEN LOCVIEW=*NONE.
- Users opening with DIV-SHARUPD=*YES are not compatible with users opening with UPAM/FASTPAM-SHARUPD=*YES.
- Read operations are always compatible irrespective of the access method, the values specified for SHARUPD and LOCKENV and the host.
- Illegal combinations result in an OPEN error.
- Any attempt to open a tape file with SHARUPD=YES or WEAK will also result in an OPEN error.
- If the secondary allocation for a non-key PAM or SAM disk file is smaller than BLKSIZE, UPAM will also reject any attempt to open the file with FCBTYPE=PAM and report an OPEN error. Exceptions: secondary allocation=0 or OPEN mode INPUT.
- If BLKSIZE is not a multiple of 2048 for a non-key tape file, UPAM will also reject any attempt to open the file using FCBTYPE=PAM with an OPEN error.
- When files are accessed in SHARUPD=YES mode it is possible that processing will result in a file < 32 GB growing to become a file >= 32 GB.

  Here it is necessary to distinguish between two cases:
  - Callers who are prepared for this situation
    (with the specification LARGE_FILE=*ALLOWED in the FCB macro or
    EXCEED-32GB=*ALLOWED in the ADD-FILE-LINK command)
  - Unprepared callers
    (specification LARGE_FILE=*FORBIDDEN in the FCB macro or
    EXCEED-32GB=*FORBIDDEN in the ADD-FILE-LINK command).

  The size of the file in question is checked after every allocator call. If this check indicates that the file size is >= 32 GB and the attribute LARGE_FILE=*FORBIDDEN is set in the associated FCB then processing is canceled. In this case UPAM issues the following return code (or the corresponding DMS message `DMS09AD`):

  ```
  X'000009AD' FILE SIZE GREATER 32 GIGABYTES IS NOT ALLOWED.
  ```
- The last-byte pointer specifies the byte-precise logical end of file of a PAM file. It is included in the file's catalog entry when the file is closed. (For details see section“Opening a PAM file (OPEN modes)”.) In the case of shared-update processing, the user or application must ensure that the job which writes last to the file's last logical block also finally closes the file. Otherwise the LBP is not assigned the correct value, which can lead to inconsistencies when copying and also when saving and restoring.

24 Appendix

The Appendix contains the following tables and summaries

- Label formats ("Label formats")
  - Volume header labels
  - User volume header labels (UVL1 to UVL9)
  - File header labels (HDR1 to HDR9)
  - User file header labels (UHL)
  - End-of-volume labels (EOV1 to EOV9)
  - End-of-file labels (EOF1 to EOF9)
  - User file trailer labels (UTL)
- Processing of label fields ("Processing of label fields")
  - Requirements for a sending system
  - Requirements for a receiving system

24.1 Label formats

- Volume header labels
- User volume header labels (UVL1 through UVL9)
- File header labels (HDR1 through HDR9)
- User file header labels (UHL)
- End-of-volume labels (EOV1 through EOV9)
- End-of-file labels (EOF1 through EOF9)
- User file trailer labels (UTL)

24.1.1 Volume header labels

Each volume contains at least one volume header label (VOL1) and at most nine. Volume header labels VOL2 through VOL9 are optional.

First volume header label (VOL1)

The first volume header label identifies the volume, the owner, the access conditions, the implementation, and the edition of the appropriate standard.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	VOL
4	Label number	1	1
5 to 10	Volume identifier	6	“a” characters. Permanently assigned by the owner to identify the volume.
11	Volume access indicator	1	“a” characters. Indicates restrictions governing access to the data on this volume. A space indicates that there are no access restrictions for this volume. Any other “a” character means that there are special restrictions governing access to this volume. BS2000: • Space or “0” : unrestricted access. • “1” : access restricted to owner.
12 to 24	Reserved for future standardization	13	Spaces.
25 to 37	System code	13	“a” characters. Identifies the implementation that recorded the volume header label.
38 to 51	Owner identifier	14	“a” characters. Identifies the owner of the volume. For BS2000:
38 to 41		4	Spaces.
42 to 49		8	“a” characters: user ID.
50 to 51		2	Spaces.
52 to 79	Reserved for future standardization	28	Spaces.

80	Label standard version	1	Indicates to which version of the standard the labels and data formats on the volume conform. • 4 signifies: DIN 66029-4 (September 1987 edition) • 3 signifies: DIN 66029-3 (May 1979 edition) • 2 signifies:n DIN 66029-2 (June 1976 edition) • 1 signifies: DIN 66029-1 (August 1972 edition)

Other volume header labels (VOL2 till VOL9)

The other volume header labels are optional. They are not generated by Fujitsu Technology Solutions implementations.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	VOL
4	Label number	1	Digits 2 through 9
5 to 80	Reserved for the implementation	76	There are no conventions or restrictions with regard to the recording and contents of this field.

24.1.2 User volume header labels (UVL1 through UVL9)

The user file header labels are optional.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	UVL
4	Label number	1	Digits 1 through 9
5 to 80	Reserved for the installation	76	There are no conventions or restrictions with regard to the recording and contents of this field.

BS2000 supplies the user with the user volume header labels.

24.1.3 File header labels (HDR1 through HDR9)

Each file or file section contains at least two file header labels (HDR1 and HDR2) and at most nine. File header labels HDR3 through HDR9 are optional.

First file header label (HDR1)

The first file header label identifies a file section, describes its location within the file set and defines certain attributes of the file section.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	HDR
4	Label number	1	1
5 to 21	File name	17	“a” characters. Identifies the file.
22 to 27	File set identifier	6	“a” characters. Identifies the file set to which this file belongs.
28 to 31	File section number	4	“n” characters. Identifies the file section. The number of the first file section in a file is 0001. This number is incremented by one for every subsequent file section in this file.
32 to 35	File sequence number	4	“n” characters. Identifies the file in the file set. The file sequence number of the first file in a file set is 0001. This number is incremented by one for each subsequentfile in this file set. This field must contain the same number in all the labels of a specific file, regardless of whether the file is contained on one or more volumes.
36 to 39	Generation numbers	4	“n” characters. Distinguishes the successive extensions/updates of the file from 0001 through 9999.
40 to 41	Version number	2	“n” characters. Distinguishes the successive repetitions of a generation.
42 to 47	Creation date	6	Spaces or “n” characters. Specifies the creation date of the file section. A space indicates the 20th century; the digit 0 indicates the 21st century, followed by two “n” characters (00 - 99) indicating the year within the century, followed by three “n” characters (001 - 366) indicating the day of the year. The value 00000 in the last five places indicates that the creation date is irrelevant. The value 00000 in the last five places indicates that the creation date is irrelevant.
48 to 53	Expiration date	6	Spaces or “n” characters. Specifies the earliest date on which the file section may be deleted. The format is the same as for the “creation date” field (positions 42 - 47). The value 00000 in the last five places indicates that the expiration date is irrelevant and that the file section is obsolete.

54	File access indicator	1	“a” characters. Indicates restrictions with regard to access to data in this file. A space indicates that there are no restrictions with regard to accessing the file. Any other “a” character indicates that there are special restrictions governing access to the file. BS2000: with “1” or “3” tape or file owner has access authorization.
55 to 60	Block count	6	000000
61 to 73	System code	13	“a” characters. identifies the implementation responsible for creating the labels.
61 to 65		5	BS2000: Spaces.
66 to 73		8	BS2000: Spaces.
74 to 80	Reserved for future standardization	7	Spaces.

Second file header label (HDR2)

The second file header label describes certain attributes of the file and of the implementation.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	HDR
4	Label number	1	2
5	record format	1	F, D or S, and V and U (not supported according to DIN); specifies the format of the records in the file. • F: all the records in the file have a fixed record length. • D: all the records in the file have a variable record length and the number of characters (as a decimal) is specified in the record itself. • S: all the records are segmented. • U: all the records have an undefined length (not supported according to DIN). • V: all the records are variable in length and the number of characters (in binary form) is specified in the record itself (not supported according to DIN).
6 to 10	Block length	5	“n” characters. Specifies the maximum number of characters per block in the file. BS2000: With standard identifier 1 the following contents are possible:
6 to 7	Standard blocks		“80”: standard block identifier.
8 to 10			“n” characters. Specifies the number of standard blocks.
11 to 15	Record length	5	“n” characters. Identifies the record length in conjunction with the record format (position 5). • With record format F this field contains the actual record length. • With record formats D and V this field contains the maximum length, including the record length word (RLW). • With record format S this field contains the maximum record length, excluding the segment control words (SCW). If this field contains 00000 with record format S, it means that the record length may be greater than 99999. • With record format U, this field contains the maximum number of characters that may be contained in one record.

16 to 50	Reserved for the implementation	35	There are no conventions or restrictions with regard to the recording and contents of this field.
16	Recording density	1	BS2000 assignments up to support of DIN level 4: 0 = 200 Bpi 1 = 556 Bpi 2 = 800 Bpi 3 = 1600 Bpi 4 = 6250 Bpi
17	Data position	1	Indicator in the case of track swapping: 0 = no 1 = yes
18 to 34	Job step	17	ID assigned by task management
35 to 36	Recording density for tape cartridges	2	2 spaces = not compressed “P” and space = compressed
47 to 50	File name code	4	Used only up to DIN 66029-1 if positions 6 and 7 contain standard blocks.
51 to 52	Buffer displacement	2	“n” characters. Specifies the length (in characters) of an additional field that is placed at the beginning of each block.
53 to 80	Reserved for future standardization	28	Spaces.

Third file header label (HDR3)

The HDR3 label contains the complete file name, the passwords and the access mode relevant for the file owner.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	HDR
4	Label number	1	3
5 to 12	Owner identifier	8	Identifies the owner of the file (user ID).
13 to 56	File name	44	The first 44 characters of the name of the file or file generationto which the file belongs.
57 to 60	read password	4	Specifies a password governing read access to the file.
61 to 64	write password	4	Specifies a password governing read and write access to the file.
65 to 68	Execute password	4	Specifies a password that must be entered in order to execute a load module contained in the file.
69	Access method	1	Specifies the valid access mode: 0 : read and write access permitted. 1 : only read access permitted.
70 to 80	Reserved	11	Spaces.

Optional file header labels (HDR4 through HDR9)

The other file header labels are optional and contain implementation-specific information.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	HDR
4	Label number	1	Digits 4 through 9
5 to 80	Reserved for the implementation	76	There are no conventions or restrictions with regard to the recording and contents of this field.

24.1.4 User file header labels (UHL)

The user file header labels are optional.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	UHL
4	Label number	1	“a” characters. Must be defined by the user. BS2000/OSD: “b” characters.
5 to 80	Reserved for the user	76	There are no conventions or restrictions with regard to the recording and contents of this field.

BS2000 supports up to 255 user file header labels. The label number (position 4) is defined by the user.

24.1.5 End-of-volume labels (EOV1 through EOV9)

Every file section extended on a continuation volume contains at least two end-of-volume labels (EOV1 and EOV2) and at most nine. End-of-volume labels EOV3 through EOV9 are optional.

First end-of-volume label (EOV1)

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	EOV
4	Label number	1	1
5 to 54	As for the corresponding fields in HDR1	Total 50	As for the corresponding fields in HDR1
55 to 60	Block count	6	“n” characters. Specifies the number of data blocks that make up the file section.
61 to 80	As for the corresponding fields in HDR1	Total 20	As for the corresponding fields in HDR1

Second end-of-volume label (EOV1)

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	EOV
4	Label number	1	2
5 to 80	As for the corresponding fields in HDR2	Total 76	As for the corresponding fields in HDR2

Third end-of-volume label (EOV3)

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	EOV
4	Label number	1	3
5 to 80	As for the corresponding fields in HDR3	76	As for the corresponding fields in HDR3.

Optional end-of-volume labels (EOV4 through EOV9)

The end-of-volume labels contain implementation-specific information.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	EOV
4	Label number	1	4 to 9
5 to 80	Reserved for the implementation	76	There are no conventions or restrictions with regard to the recording and contents of this field.

24.1.6 End-of-file labels (EOF1 through EOF9)

Each file contains at least two end-of-file labels (EOF1 and EOF2) and at most nine. Endof-file labels EOF3 through EOF9 are optional.

First end-of-file label (EOF1)

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	EOF
4	Label number	1	1
5 to 54	As for the corresponding fields in HDR1	Total 50	As for the corresponding fields in HDR1
55 to 60	Block count	6	“n” characters. Specifies the number of data blocks that make up the file section.
61 to 80	As for the corresponding fields in HDR1	Total 20	As for the corresponding fields in HDR1

Second end-of-file label (EOF2)

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	EOF
4	Label number	1	2
5 to 80	As for the corresponding fields in HDR2	Total 76	As for the corresponding fields in HDR2

Third end-of-file label (EOF3)

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	EOF
4	Label number	1	3
5 to 80	As for the corresponding fields in HDR3	76	As for the corresponding fields in HDR3.

Optional end-of-file labels (EOF4 through EOF9)

The other end-of-file labels are optional and contain implementation-specific information.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	EOF
4	Label number	1	4 to 9
5 to 80	Reserved for the implementation	76	There are no conventions or restrictions with regard to the recording and contents of this field.

24.1.7 User file trailer labels (UTL)

The user file trailer labels are optional.

Format

Position	Field name	Length	Contents
1 to 3	Label identifier	3	UTL
4	Label number	1	“a” characters. Must be defined by the user. BS2000/OSD: “b” characters.
5 to 80	Reserved for the user	76	There are no conventions or restrictions with regard to the recording and contents of this field.

BS2000 supports up to 255 user file trailer labels. The label numbers (position 4) are defined by the user.

24.2 Processing of label fields

- Requirements for a sending system
- Requirements for a receiving system

24.2.1 Requirements for a sending system

Files

The records of the files must be passed to the implementation by the application program.

Labels

The installation must transfer the recording information required in each of the label fields listed below to the implementation; otherwise the implementation must supply this information.

For each volume in a volume set:

- volume identifier (VOL1, positions 5 through 10)
- volume access indicator (VOL1, position 11).

For each file in a file set:

- file access indicator (HDR1, position 54)

If the implementation permits the installation to supply the information to be processed in each of the label fields listed below, the implementation must process this information. If the installation does not supply this information, it must be supplied by the implementation.

For each volume in a volume set:

- owner identifier (VOL1, positions 38 through 51)

For each file in a file set:

- file set identifier (HDR1, positions 22 through 27)

The implementation must permit the application program to supply the information to be processed in each of the label fields listed below. If the application program does not supply this information, the implementation must supply the information for the appropriate fields.

For each file in a file set:

- file name (HDR1, positions 5 through 21)
- record format (HDR2, position 5)
- block length (HDR2, positions 6 through 10)
- record length (HDR2, positions 11 through 15)

If the implementation permits the application program to supply the information to be processed in each of the label fields listed below, the implementation must process this information. If the application program does not supply this information, it must be supplied by the implementation.

For each file in a file set:

- generation number (HDR1, positions 36 through 39)
- version number (HDR1, positions 40 and 41)

For each file section of a file set:

- creation date (HDR1, positions 42 through 47)

- expiration date (HDR1, positions 48 through 53)

If the implementation is in a position to process a set of installation volume header labels (UVL), the implementation must permit the installation to supply the information to be processed from the label fields listed below. Processing of the corresponding labels is not requested if the installation does not supply the information.

For each label from a set of installation volume header labels recorded on each volume in a volume set:

- reserved for the installation (UVL, positions 5 through 80)

If the implementation is in a position to process a set of user file header labels (UHL) or of user file trailer labels (UTL), the implementation must permit the application program to supply the information to be entered in the label fields listed below for a label set. Processing of the corresponding labels is not requested if the application program does not supply the information.

For each label in a set of user file header/trailer labels for a magnetic tape:

- label number (UHL/UTL, position 4)
- reserved for the user (UHL/UTL, positions 5 through 80)

The implementation can impose the following restrictions on the user with regard to the record length (HDR2, positions 11 through 15).

If the implementation is processing segmented records, it can define a maximum record length. This should not be less than the maximum permissible block length minus the length of the buffer displacement field and the length of the segment control word (SCW).

If the implementation is processing variable-length records, it can define a maximum record length corresponding to the maximum block length minus the length of the buffer displacement field and the length of the record length word (RLW).

24.2.2 Requirements for a receiving system

Files

The implementation must supply the application program with the contents of the records and the length of each record. The segment control word (SCW) and the record length word (RLW) are not part of the record.

Labels

The implementation must permit the user to supply enough information to enable him/her to select both the requested files and the volume on which they are recorded.

The implementation must supply the installation with the information contained in the following label fields:

For each volume in a volume set:

- volume identifier (VOL1, positions 5 through 10)
- volume access indicator (VOL1, position 11).

For each file in a file set:

- file access indicator (HDR1, position 54)

The implementation must supply the application program with the information contained in the following label fields:

For each file in a file set:

- file name (HDR1, positions 5 through 21)
- record format (HDR2, position 5)
- block length (HDR2, positions 6 through 10)
- record length (HDR2, positions 11 through 15)

The implementation need not supply the user with the information contained in the following label fields:

For each volume in a volume set:

- owner identifier (VOL1, positions 38 through 51)

For each file in a file set:

- file set identifier (HDR1, positions 22 through 27)
- generation number (HDR1, positions 36 through 39)
- version number (HDR1, positions 40 and 41)

For each file section of a file set:

- creation date (HDR1, positions 42 through 47)
- expiration date (HDR1, positions 48 through 53)

If the implementation is in a position to supply the installation with the information recorded in the set of installation volume header labels (UVL), the information contained in the following label fields must be provided:

For each label of a set of installation volume header labels:

- reserved for the installation (UVL, positions 5 through 80)

If the implementation is in a position to supply the user with the information recorded in the set of user file header labels (UHL) or user file trailer labels (UTL), the information contained in the following label fields must be supplied:

- label number (UHL/UTL, position 4)
- reserved for the user (UHL/UTL, positions 5 through 80)

25 Glossary

This glossary provides short definitions of some of the terms used in this manual.

Access Control List (ACL)

File protection with an ACL (Access Control List) has not been supported since SECOS V4.0. The ACL file attribute is still contained in the catalog entry but usually contains the value NO (no ACL protection).

In case a file is ACL protected (might be possible for very old data), file access is not possible. In this case, the owner can protect the file with GUARDS. The protection with GUARDS “overwrites” the ACL protection and thus makes the file accessible again.

access method

A conventional data management technique which defines, for the user, the data organization and the method of data transfer between I/O devices and the main memory of the system.

Access methods supported by DMS are:

- EAM = Evanescent Access Method
- SAM = Sequential Access Method
- ISAM = Indexed-Sequential Access Method
- UPAM = User Primary Access Method
- BTAM = Basic Tape Access Method

Alias Catalog Service (ACS)

Files and job variables can be addressed by means of alias names and stored in special catalogs, the alias catalogs, together with their assignment to the real file/JV. The Alias Catalog Service (ACS) incorporates three basic functions: Alias name definition, catid insertion for temporary spool files and prefix insertion.

alphanumeric

The alphanumeric characters consist of *alpha*betic and *numeric* characters, i.e. the letters A-Z and the digits 0-9.

batch mode

An operating mode in which a job is started with an ENTER-JOB or ENTER-PROCEDURE command; in contrast to interactive mode, the sequence of operations is predefined and stored in an ENTER file (started with ENTER-JOB) or in a procedure file (started with ENTER-PROCEDURE).

batch processing

See batch mode.

batch processing

See batch mode.

block

See PAM page, data block.

blocked record

A record in a file in which each data block may contain several records.

BS2000 file

Is a file which is only created and processed by BS2000. BS2000 files on Net-Storage (FILE-TYPE=BS2000) have been supported since BS2000/OSD-BC V9.0. They are located directly on a Net-Storage volume. Open systems may only access them in read mode.

buffer

A contiguous area in main memory into which data is written and from whichdata is read.

CALL procedure

Sequence of commands/statements executed within a job and called by means of the CALL-PROCEDURE command (see the “Commands” manual [3]).

catalog ID

The identifier of a pubset (see pubset Id); it is specified in a full file name or path name in the form : catid:.

class 1 memory

That part of virtual user memory which is occupied by the main memory-resident modules of the Executive. All class 1 pages are marked as privileged and non-pageable. The pages are not mapped on paging memory. The pages remain in main memory throughout the session.

class 5 memory

That part of virtual user memory which contains the pageable areas allocated dynamically by the Executive when needed for a user job.

class 6 memory

That part of virtual user memory containing the user programs; space is allocated dynamically by the Executive.

data block

A block which contains one or more records of a file.

double tape mark

Two directly adjacent tape marks which indicate the logical end-of-tape. Two adjacent tape marks may also occur if the tape contains an empty file or an empty file section; in this case, they are not regarded as a double tape mark, but as two single tape marks.

“Empty” in this sense means that there are no data blocks between the tape marks before the file header labels and after the trailer labels.

file

Related records are grouped together in a named entity called a file. Typical examples of files are: conventional input and output data of programs; load modules and object module libraries; text information which is created and processed with a file editor.

file control block (FCB)

A file control block contains all the necessary information about a file.

file link name

A name with up to 8 characters which establishes the link between the file control block and the file via the task file table.

file section

That part of a file which is stored on a single tape.

file set

A set of files recorded sequentially on one or more tapes. There must be no sections of other files between the sections of a file.

first in – first out (FIFO)

Queue structure according to which information is processed in the order in which it is input (in contrast to last in – first out, LIFO).

foreign file

A foreign file is a file on a private volume or on a Net-Storage volume which is not cataloged on a pubset.

interactive mode

The operating mode in which a job is initiated at a (remote) terminal and executed; the sequence of processing is not predefined.

job

The totality of all operational sequences between the commands SET-LOGON-PARAMETERS or LOGON and EXIT-JOB or LOGOFF. For this definition, it is immaterial whether the job is already fully defined when it is submitted (batch mode) or whether the individual steps are defined in the course of execution (dialog mode).

label

A record at the beginning or end of a file or tape which is used to identify, describe and/or delimit the tape or file. A label is not regarded as part of the file.Each label is recorded separately in its own block (label block).

label group

An uninterrupted sequence of label sets which delimit a tape, a file section or a file.

label handling routine

A series of statements for the processing of labels.

label identifier

A three-character word which is recorded as part of the label and identifies this label.

label set

An uninterrupted sequence of labels with the same label identifier.

last-byte pointer (LBP)

Pointer to the last valid byte of the last logical block of a PAM file.

last-page pointer (LPP)

Pointer to the last PAM page occupied by a file. In the catalog entry corresponds to the highest-used page.

locate mode

In locate mode, the user requests the address of the current record, which is stored in a buffer area. The user is responsible for the data transfer from and to the buffer.

locate mode

In locate mode, the user requests the address of the current record, which is stored in a buffer area. The user is responsible for the data transfer from and to the buffer.

logical block number (LBN)

(Sequence) numbering of the PAM pages in a file.

move mode

In move mode, the user specifies the position of the record in his/her program. The system is responsible for data transfer from and to the buffer.

move mode

In move mode, the user specifies the position of the record in his program. The system is responsible for data transfer from and to the buffer.

Net client

Implements access to Net-Storage for the operating system using it.
In BS2000 the net client, together with the BS2000 subsystem ONETSTOR, transforms the BS2000 file accesses to corresponding UNIX file accesses and executes them on the net server using NFS.
The net client for SUs /390 and S servers is the HNC, and for SUs x86 and SQ servers the X2000 carrier system.

net server

A file server in the worldwide computer network which provides storage space (Network Attached Storage, NAS) for use by other servers and offers corresponding file server services.

Net-Storage

Storage space provided by a net server in the computer network and storage space released for use by foreign servers. Net-Storage can be a file system or also just a node in the net server’s file system.

Net-Storage file

Is a file which is created on a Net-Storage volume. On Net-Storage a distinction is made between the two file types BS2000 file and node file.

Net-Storage volume

Net-Storage volumes represent Net-Storage in BS2000 which provides systems support as an enhancement of data pubsets.
Net-Storage volumes are addressed by means of their Volume Serial Number (VSN) and the volume type NETSTOR. In the released file system of the net server the VSN of the Net-Storage volume corresponds to the directory
containing the user files and metadata.

NFS (Network File System)

BS2000 software product which enables distributed data storage in a heterogeneous computer network. The user can access remote files as if they were present on his/her local computer.
NFS thus supports connectivity between systems. In addition, NFS enables files to be saved automatically and reliably by BS2000.

node file

Is a Net-Storage file (FILE-TYPE=NODE-FILE) which can be created and processed by both BS2000 and open systems. Node files are supported in BS2000 OSD/BC V10.0 and higher. They are located on a Net-Storage volume in a userspecific directory (name of the user ID), and the file names comply with the BS2000 naming conventions.

null file

A file which is logically empty, i.e. a file which is cataloged and to which the systemhas allocated space, but contains no data.

PAM page

Contiguous storage space of 2048 bytes, beginning at an address divisible by 2048 .

privileged mode/program

All parts of the operating system that are not executed in the nonprivileged processing state.

procedure/procedure file

A file which contains a predefined sequence of commands or statements used for program input. Procedures are started with CALL-PROCEDURE or ENTER-PROCEDURE. Only if the file contains an ENTER job will it be started with ENTER-JOB. For detailed information on procedures, see the "Commands" manual [3]).

public volume set (PVS)

Term previously used for: pubset; see pubset

pubset

A set of disks designated as public. MPVS systems work with several mutually independent pubsets.

pubset Id

The pubset identifier. If the volume serial number of a public volume begins with the three characters “PUB”, the pubset Id is the fourth character; if the volume serial number contains a period, the pubset Id is formed by the characters preceding the period. In the path name the pubset Id is specified in the format :catid: (see catalog ID).

PVS Id

See pubset Id.

record

A group of data items which are treated as a logical unit.

fixed-length record:
A record in a file in which all records are declared as having the same length; no record length information is needed within the file.

variable-length record:
A record in a file in which the records may have different lengths. The record length must be specified in the first word in each record. This record length field is included in the length of the record; it is 4 bytes long and contains the record length left-aligned in decimal or hexadecimal.

record format

The definition of the length and segmentation of records for a file.

Record format V:
if a format V tape is written with non-EBCDIC code, the length is specified in hexadecimal notation.

Record format D:
if a format D tape is written with ISO 7-bit code, the length is specified as a four-digit decimal number.

shareable file

A file which is cataloged with the USER-ACCESS=ALL-USERS. A shareable file can be accessed from all user IDs provided the other protection attributes (e.g. file passwords) of the file permit this.

spoolout

Automatic spoolout: automatic output of the contents of system file SYSLST to a printer or by sending it by email when a job terminates (EXIT-JOB or LOGOFF)

SYSFILE environment

See system files; the SYSFILE environment consists of all the system files assigned to a job.

system files

System input/output files that are assigned to a job.
The (default) file names SYSDTA, SYSSTMT, SYSCMD, SYSIPT, SYSLST, SYSLST01, SYSLST02, ..., SYSLST99, SYSOPT and SYSOUT refer to the (system) files for data and command input to the operating system or for data output by the operating system. Each of these files is created by the task and specifies predefined I/O areas.
The user can cancel the primary assignment and assign the (default) file names to his/her own cataloged files or compound S variables (when the software product SDFG-P is used).
For detailed information on system files, see the "Commands" manual [3].

tape (volume)

An exchangeable unit of the data storage medium magnetic tape or magnetic tape cartridge. A tape may contain all or part of one file, several files and/or one or more file sections.

tape mark

A tape block which marks the boundary between data blocks and the tape label groups and also between certain label groups. The format of the tape mark is defined in the related standards for magnetic tapes.

tape set table (TST)

A table which shows the tapes requested for a job (together with the TFT).

task file table (TFT)

A table created using the LINK-NAME operand of the ADD-FILE-LINK command and from which the file and processing attributes are taken and entered in the file control block when a file is opened.

task sequence number (TSN)

The sequential number assigned by the system to the task (or job) with which the user can identify a task in some commands.

TSN (= task sequence number)

See task sequence number (TSN).

unblocked record

A record in a file in which each data block may contain only one record.

volume sequence number (VSEQ)

The number of a file section in a multivolume file.

volume serial number (VSN)

A six-character string assigned to the volume when it is initialized (VOLIN or INIT). It is kept in the standard volume label and is used to identify the volume.

volume set

The tape or tapes on which the files of a file set are stored.

volume table of contents (VTOC)

Directory in the F1 label of a private disk or on a Net-Storage volume.

26 Related publications

You will find the manuals on the internet at *http://bs2manuals.ts.fujitsu.com*. You can order printed versions of manuals which are displayed with the order number.

[1] **BS2000 OSD/BC**
DMS Macros
User Guide

[2] **BS2000 OSD/BC**
Executive Macros
User Guide

[3] **BS2000 OSD/BC**
Commands
User Guide

[4] **SPOOL** (BS2000)
User Guide

[5] **DAB** (BS2000)
Disk Access Buffer
User Guide

[6] **RFA** (BS2000)
Remote File Access
User Guide

[7] **BS2000 OSD/BC**
Introduction to System Administration
User Guide

[8] **SECOS** (BS2000)
Security Control System - Access Control

User Guide

[9] **ARCHIVE** (BS2000)
User Guide

[10] **HSMS** (BS2000)
Hierarchical Storage Management System

User Guide

[11] **HIPLEX MSCF** (BS2000)
BS2000-Processor Networks
User Guide

[12] **PERCON** (BS2000)
User Guide

[13] **BS2000 OSD/BC**
Utility Routines
User Guide

[14] **DRV** (BS2000)
Dual Recording by Volume
User Guide

[15] **BS2000 OSD/BC**
System Installation
User Guide

[16] **SDF-A** (BS2000)
User Guide

[17] **SDF-P** (BS2000)
Programming in the Command Language
User Guide

[18] **BS2000 OSD/BC**
Files and Volumes Larger than 32 GB
User Guide

[19] **RSO** (BS2000)
Remote SPOOL Output
User Guide

[20] **JV** (BS2000)
Job Variables
User Guide

[21] **XHCS** (BS2000)
8-Bit Code Processing in BS2000
User Guide

[22] **BS2000 OSD/BC**
System Managed Storage
User Guide

[23] **FUJITSU Server BS2000 SE Serie**
Operation and Administration
User Guide

[24] **CRYPT** (BS2000)
Security with Cryptography
User Guide

[25] **VM2000** (BS2000)
Virtual Machine System
User Guide

[26] **SHC-OSD** (BS2000)
Storage Management for BS2000
User Guide