

Burroughs

**Computer
Management
System
(CMS)
B 1000
Program Dump
Analysis
USER'S GUIDE**

This Manual Replaces All Previous Editions of Form 2018750

COPYRIGHT © 1982, BURROUGHS MACHINES LIMITED, Hounslow, England
COPYRIGHT © 1982, BURROUGHS CORPORATION, Detroit, Michigan, 48232
PRICED ITEM

Burroughs cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded using the Remarks form at the back of the manual, or may be addressed directly to TIO Europe Documentation, Burroughs Machines Limited, Cumbernauld, Glasgow, Scotland G68 0LN.

LIST OF EFFECTIVE PAGES

Page	Issue
iii	Original
iv	Blank
v	Original
vi	Blank
1-1	Original
1-2	Blank
2-1 thru 2-3	Original
2-4	Blank
3-1 thru 3-2	Original
4-1 thru 4-57	Original
4-58	Blank
A-1 thru A-2	Original

TABLE OF CONTENTS

Section	Title	Page
1	INTRODUCTION	1-1
2	OPERATING INSTRUCTIONS	2-1
	EXAMPLES	2-3
3	PROGRAM DUMP ANALYZER OUTPUT	3-1
	HOW TO USE A PROGRAM DUMP	3-1
	COMPILER OPTIONS	3-2
4	SECTION DESCRIPTION	4-1
	PROGRAM PARAMETERS	4-1
	RUNNING PARAMETERS	4-2
	INTERFACE CONTROL BLOCK	4-4
	COMMUNICATE PARAMETER AREA	4-10
	PERFORM STACK/CONTROL STACK	4-19
	Perform Stack (COBOL and RPG)	4-19
	Control Stack (MPL)	4-20
	PROGRAM SEGMENT TABLE	4-21
	DATA SEGMENT TABLE	4-22
	INTERNAL FILE NAME BLOCK	4-22
	FILE INFORMATION	4-23
	Information for an Opened File	4-23
	Information from FCB	4-23
	File Attributes from FPB	4-28
	Control Information from FIB	4-32
	Buffers	4-37
	Extension for Indexed Files	4-37
	Information for a Closed File	4-41
	DATA STACK ANALYSIS/CURRENT OPERAND (COP) TABLE	4-42
	Data Stack Analysis (MPL)	4-42
	Data Stack Structure	4-42
	Heading Description	4-42
	Data Description	4-43
	Current Operand (COP) Table (COBOL and RPG)	4-44
	DATA SEGMENTS	4-46
	CURRENT CODE SEGMENT	4-55
	LOCKED SLICE	4-55
	Program Segment Table	4-56
	Data Segment Table	4-57
	Task, Control Block Preset Area	4-57
	Control Stack	4-57
	Code Control Block Preset Area	4-57
	Internal File Name Block	4-57
APPENDIX		
A	GLOSSARY OF TERMS	A-1

SECTION 1 INTRODUCTION

THIS MANUAL IS RELATIVE TO THE B 1000 CMS SYSTEM SOFTWARE RELEASE LEVEL
3.04

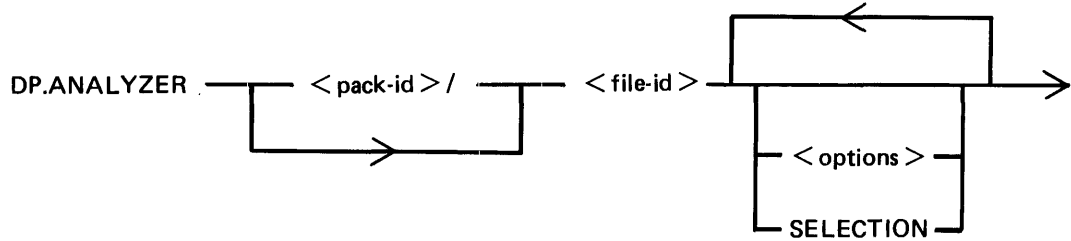
Execution of a program may lead to a DS/DP condition. When a program is DP'ed (either by the operator or by the COBOL interpreter if the "NO USE PROCEDURE" condition is encountered after an error), a dump file is created on the same disk as the program file. The name of the dump file is DMFILnn, where nn is the mix number.

In addition, the DM command can be used to force the creation of a dump file for an executing program (that is, a program which is not in the DS/DP state). The program will be suspended when the DM command is given and a dump file will be created in the same manner as if a DP was issued. The program will then require the GO command to be issued by the operator before resuming processing.

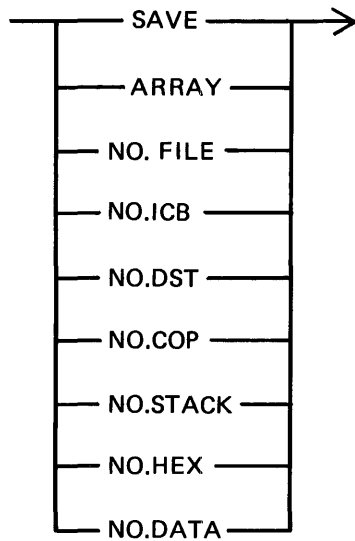
The program dump analyzer, named DP.ANALYZER, analyzes this dump file, in relation to the program file itself, giving a listing which contains information about the program parameters, the files, the segments and so on.

A glossary of the terms used in this manual appears in Appendix 'A' at the end of this book.

SECTION 2 OPERATING INSTRUCTIONS



Options



< pack-id > when not specified defaults to the system disk

< file-id > is the name of a dump file.

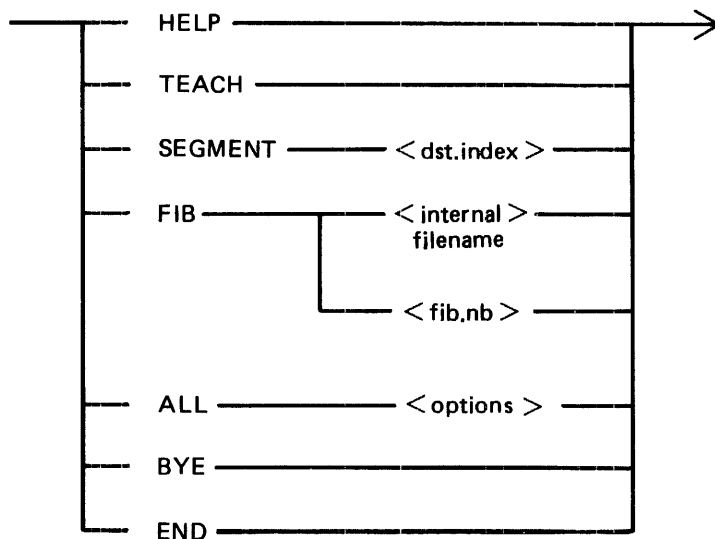
1. If no parameter is specified, all the dump file is analyzed. The dump file is then removed.
2. The options, entered in any order, have the following effect:

SAVE	This will cause DP.ANALYZER to retain the input dump file rather than remove it after analysis (which is the default).
ARRAY	This is applicable only to COBOL/RPG dump files. It will cause all elements of all arrays to be printed. The default is to print a maximum of twenty elements of each array.
NO.FILE	This will suppress printing of any file information.
NO.ICB	This will suppress printing of any ICB information.
NO.DST	This will suppress the DST and PST analysis.

NO.COP	This will suppress the COP table analysis (only for COBOL and RPG programs).
NO.STACK	This will suppress the Data Stack Analysis (only for MPL II programs).
NO.HEX	This will suppress the printing of the current code segment and locked slice.
NO.DATA	This will suppress the printing of data segments.

3. If SELECTION is specified, the PROGRAM and RUNNING PARAMETERS, the ICB, the CPA, the CONTROL STACK/PERFORM STACK, the PST, the DST and the IFNB are printed, except if otherwise specified in the initiating message.

The user has the option to request for the analysis of specific parts of the dump file, via accepts.



HELP TEACH	The list of all the parameters which may be entered via accepts is displayed.
SEGMENT	The data segment whose index number is specified by “dst.index” is printed.
FIB	The FIB of the file specified either by its “internal filename” or its “fib.nb” is printed. “fib.nb” is the index number of the data segment which contains the FIB.
ALL	The sections of the dump file selected by the “options” are printed. The dump file is then removed (if not otherwise specified), and DP.ANALYZER goes to EOJ.
BYE	DP.ANALYZER goes to EOJ. The dump file is not removed.
END	

EXAMPLES

1. DP.ANALYZER DMFIL03 SAVE

DP.ANALYZER will expect to find a dump file called DMFIL03 on the system disk, will print a formatted dump listing and will retain the dump file.

2. DP.ANALYZER USER/DMFIL02 NO.ICB NO.HEX

DP.ANALYZER will expect to find a dump file called DMFIL02 on a disk labelled USER, will print a formatted listing omitting the Interface Control Block, current code segment and locked slice sections and will remove the file USER/DMFIL02 at the end of the job.

3. DP.ANALYZER DMFIL04 ARRAY SAVE

DP.ANALYZER will expect to find a dump file called DMFIL04 on the system disk, will print a formatted listing including all the elements of all arrays and will retain DMFIL04 at the end of the job.

4. DP.ANALYZER DMFIL04 NO.ICB NO.DST SELECTION

DP.ANALYZER will expect to find a dump file called DMFIL04 on the system disk, will print a formatted listing containing the program and running parameters, the CPA, the CONTROL STACK/PERFORM STACK and the IFNB. Parameters may then be entered via accepts.

NOTE

Dumps submitted with a Field Communication Form (FCF) must be full dumps, that is, ARRAY must be specified and no other option may be specified.

Sometimes, due to the BIL interpreter, the contents of the end of an MPLII program's working stack is unpredictable. In this case, the printing will be limited to the analyzable portion.

SECTION 3

PROGRAM DUMP ANALYZER OUTPUT

The program dump analyzer output is described section by section, in the order it occurs in the DP.ANALYZER listing. In each section, differences and similarities for each of the three languages, MPL, COBOL and RPG are given.

The sections as they occur in a DP.ANALYZER listing are as follows:

TITLE

PROGRAM PARAMETERS

RUNNING PARAMETERS

INTERFACE CONTROL BLOCK (ICB)

COMMUNICATE PARAMETER AREA (CPA)

PERFORM STACK/CONTROL STACK

PROGRAM SEGMENT TABLE

DATA SEGMENT TABLE (DST)

INTERNAL FILE NAME BLOCK

FILE INFORMATION

DATA STACK ANALYSIS/CURRENT OPERAND (COP) TABLE

DATA SEGMENTS

CURRENT CODE SEGMENT

LOCKED SLICE

The sections PROGRAM PARAMETERS till DATA STACK ANALYSIS/CURRENT OPERAND (COP) TABLE give labelled information taken from the Internal Control Block, and/or the Locked Slice, and/or the Data Segments.

The sections DATA SEGMENTS till LOCKED SLICE contain the display of the corresponding memory area; these displays may be useful when some control information has been destroyed.

Note that the sections Interface Control Block, Code Segment and Locked Slice analysis require a good knowledge of the operating system; a good practice would be to specify, when calling the DP.ANALYZER program, the parameters NO.ICB NO.HEX.

HOW TO USE A PROGRAM DUMP

If a dump is requested because of I/O problems, the section to look at is the COMMUNICATE PARAMETER AREA. When the reason for the I/O problem has been determined, its location may be delimited using THE NEXT INSTRUCTION in the RUNNING PARAMETERS section.

More information, if required, can be found in the sections FILE INFORMATION, DATA STACK ANALYSIS/CURRENT OPERAND (COP) TABLE and DATA SEGMENTS.

For other types of analysis, after locating the NEXT INSTRUCTION, the DATA STACK ANALYSIS/CURRENT OPERAND (COP) TABLE section may give useful information on the state of the variables.

COMPILER OPTIONS

To facilitate reading the DP.ANALYZER output, compiler options have to be specified; that is:

- in MPL : **\$ XMAP**
- in COBOL : **\$LINE-CODE**, **\$LINE-MAP**, **\$COP-TABLE** or **\$OPTCODE**
- in RPG : column 15 of the H card specification must be set to 1.
\$MAP or **\$PARMAP** or **\$XMAP**, **\$NAMES**.

The implications of those options are stated in the appropriate section.

SECTION 4

SECTION DESCRIPTION

Along with the title "C.M.S. DUMP-ANALYZER", the compile date and release level of DP.ANALYZER are given. Note that this information applies to DP.ANALYZER and not to the system release information.

PROGRAM PARAMETERS

PROGRAM NAME	This is the name of the user's program being analyzed in the DP.ANALYZER listing.
COMPILER NAME	This is the name of the compiler employed by the user's program.
COMPILER SYSTEM	This gives the processor name and MCP version of the system on which the user's program was compiled.
COMPILATION DATE	This is the date on which the user's program was last compiled.
INTERPRETER PACK.ID	This is the name of the pack on which the interpreter employed by the user's program is located.
INTERPRETER NAME	This is the name of the interpreter employed by the user's program. The released names of the interpreters are as follows: MPL = BILINTERP COBOL = COBOLINT RPG = COBOLINT
PRIORITY CLASS	This is the priority assigned by the compiler to the user's program. In general, priorities are assigned as follows: A = normal user tasks B = utilities C = data comm tasks Depending on the Program Parameter Block flags, one or all of the following messages may be printed after Priority Class: MAY OPEN SYSTEMEM SUPPRESS BOJ-EQJ MESSAGES MCS PROGRAM FILE NDL PROGRAM FILE PROGRAM USES DATA COMM COMMUNICATES This information is read from the field ICB.PRIORITY.CLASS which is fully described in the section entitled INTERFACE CONTROL BLOCK.

INITIATING MSG SEGMENT	If there is a segment for the initiating message declared in the user's program, this will give the initiating message segment number.																								
S-PROGR START ADDRESS	This is the SEGMENT number and DISPLACEMENT of the code segment containing the first executable instruction. It is also called the "S-CODE entry address".																								
INTERPRETER PRESET AREA (COBOL and RPG only)	This reflects the values of the eight edit characters. The following table indicates the edit table default values.																								
	<table> <tr><td>position 0</td><td>=</td><td>" + "</td></tr> <tr><td>1</td><td>=</td><td>" - "</td></tr> <tr><td>2</td><td>=</td><td>" "</td></tr> <tr><td>3</td><td>=</td><td>" * "</td></tr> <tr><td>4</td><td>=</td><td>" . "</td></tr> <tr><td>5</td><td>=</td><td>" , "</td></tr> <tr><td>6</td><td>=</td><td>" \$ "</td></tr> <tr><td>7</td><td>=</td><td>" 0 "</td></tr> </table>	position 0	=	" + "	1	=	" - "	2	=	" "	3	=	" * "	4	=	" . "	5	=	" , "	6	=	" \$ "	7	=	" 0 "
position 0	=	" + "																							
1	=	" - "																							
2	=	" "																							
3	=	" * "																							
4	=	" . "																							
5	=	" , "																							
6	=	" \$ "																							
7	=	" 0 "																							

RUNNING PARAMETERS

MIX NUMBER	This is the mix number the user program had at the time it was DP'ed or DM'ed.
BOJ AT:	This is the time and the date at which the user job began executing. The time is in the format HH:MM:SS.
DUMP AT:	This is the time and the date at which the user job was DP'ed. The time is in the format HH:MM:SS. If the program was DM'ed, the time and date of the dump will not be available. In this case, an appropriate message will be given.
PROGRAM STATUS	A message is printed which reflects the status of the user program at the time of the DP or DM action. There are three basic states a user task may be in during its lifetime: executing, delayed (that is, executing but waiting for some condition which may be satisfied at any moment; for example, "WAITING FOR I/O") or suspended (that is, no longer executing, waiting for some condition which will not be satisfied without operator intervention; for example, "WAITING FOR DIRECTORY SPACE").
REASON FOR DUMP	In the case of an error condition, the message given here will give the event number and description of the error. If the program was DP'ed while executing, the message "OPERATOR ACTION" will be given. If the program was DM'ed while executing, the message "DM FUNCTION" will be given.
NEXT INSTRUCTION	This gives the location that the execution was proceeding to at the time of the abnormal termination.

In the case of an error causing a DS/DP condition, this would be the location at which the error occurred; but if the program was DP'ed and no error caused the DS/DP condition, it gives the location of the next S-OP to be handled by the interpreter. In the case of DM with specified breakpoints, it corresponds to the location of the breakpoint. Or, if the program was DM'ed without specifying breakpoints, it gives the location of the next S-OP to be handled by the interpreter.

MPL related

SEGMENT (PSN) This is the code segment number where the next instruction is to be found.

SEGM. DISPL This is the displacement, in bytes, in the specified code segment where the next instruction is to be found.

PROCEDURE (SPN) This is the number of the procedure in which the next instruction is located.

PROC.DISPL This gives the location, in bytes, relative to the start of the procedure, of the next instruction.

REGION 1
REGION 2 They reflect the most frequently used lexical levels found in the procedure which was active at the time of the abnormal terminate. This information is useful only when the code segment has to be decoded.

LEXICAL LEVEL This gives the current lexical level at which the next instruction resides.

COBOL/RPG related

SEGMENT This is the code segment number where the next instruction is to be found.

DISPLACEMENT This is the displacement, in bytes, in the specified code segment where the next instruction is to be found.

LINE-COUNT This is the line number of the next instruction. For COBOL, the \$LINE-CODE option must be specified in order to obtain line count information. In RPG, column 15 of the H card specification must be set to 1, indicating DEBUG, in order to obtain the same information. If this option has been specified at compile time, a code will have been added to cause the interpreter to update the LINE-NUMBER register. If the option has not been specified, the words "NOT AVAILABLE" will appear.

CARRY (MPL only) This reflects the value of the CARRY register (16-bit field) employed by the MPL compiler in arithmetic operations.

VSN (MPL only) The administration of the Virtual Segments (see SEGMAP instruction in MPL) is controlled through an 8-bit field found in the Interface Control Block. This field holds the segment number corresponding to the first page of the first virtual segment. If no virtual segments have been declared, the value of the VSN field is not significant.

OVERFLOW FLAG (COBOL/ RPG only)
 This reflects the value of the OVERFLOW flag employed by the COBOL and RPG interpreters in arithmetic operations. In COBOL, the "ON SIZE ERROR" clause controls the use of this flag. In RPG, the flag is not accessible.

INTERFACE CONTROL BLOCK

The Interface Control Block resides in the user program's partition. It is a run time data structure containing all the parameters needed by the MCP to execute the program. As previously stated, an in depth knowledge of the MCP is required to analyze this section.

For each ICB field listed, its name, as it appears on the DP.ANALYZER listing, and its description are given. Note that for each compound field its data type description has been stated as a memorandum; a complete description of the members of the structure is given in the manual "B 1000 CMS MEMORY DUMP ANALYSIS USER'S GUIDE. Form no.2018909".

ICB.TRACE.AREA	It contains internal debugging fields.
ICB.COMM.ROUTING	This is the group name for fields used by COMMUNICATE.SWITCH to transfer control and make entries to the return stack. Data type description: 01 ICB.COMM.ROUTING CHAR(7) 02 ICB.CALLING BIT(24) 03 ICB.CALLING.MOD.NB BIT(8) 03 ICB.CALLING.DISPL BIT(16) 02 ICB.SAVE.MY.PLACE BIT(8) 02 ICB.CALLED BIT(24) 03 ICB.CALLED.MOD.NB BIT(8) 03 ICB.CALLED.DISPL BIT(16)
ICB.STACK.PTR	This gives the bit displacement to the next free entry in the return stack; that is, the end of the last entry.
ICB.STACK.ENTRIES	This is the group name for the fields which make up an entry of the return stack. Data type description: array of nine entries 01 ICB.STACK.ENTRY BIT(24)

	02 ICB.STACK.MOD.NB	BIT(8)
	02 ICB.STACK.DISPL	BIT(16)
ICB.PRIORITY.CLASS	This field is defined as follows:	
	bit 0 – when set, load only if mix is suitable (nothing in the mix, not even SYS-SUPERUTL).	
	bit 1 – when set, the program may open files with file type @20@ -@48@, maintain and create non-data files and use the “wild” file type.	
	bit 2 – when set, the BOJ/EOJ messages display headings are suppressed. This includes the suppression of the zip header on a zip with display (DS-ED messages are not suppressed).	
	bits 3,4 – reserved for expansion.	
	bit 5 – when set, it indicates the priority A.	
	bit 6 – when set, it indicates the priority B.	
	bit 7 – when set, it indicates the priority C.	
	bits 8,9 = 00 : Non data comm program file 01 : Reserved for expansion 10 : NDL Program file 11 : MCS Program file	
	bit 10 – when set, the program may maintain/create all files other than those with file types @20@ -@48@ and may use the “wild” file type.	
	bits 11-14 – reserved for expansion	
	bit 15 – when set, the program contains data comm communicates.	
ICB.FETCH.VALUE	It contains the fetch value returned to an interpreter when processing of an I/O request from that interpreter has been completed. See COMMUNICATE PARAMETER AREA section for the fetch value list.	
ICB.ACTUAL.CPA	This is the communicate parameter area for communication between an interpreter and the MCP.	

Data type description:

01 ICB.ACTUAL.CPA	
02 ICB.VERB	BIT(8)
02 ICB.OBJECT	BIT(8)
02 ICB.ADVERB	BIT(104)

See the COMMUNICATE PARAMETER AREA section for a complete description of those fields.

ICB.INTERNAL.CPA This is the communicate parameter area used by the MCP.

Data type description:

01 ICB.INTERNAL.CPA	BIT(216)
02 ICB.INTERNAL.VERB	BIT(8)
02 ICB.INTERNAL.CPA.ENTRY	BIT(208)
03 ICB.INTERNAL.ADVERB	BIT(8)

ICB.REPLY.WORD This area is used to indicate success or failure of an internal communicate; that is, between MCP modules.

ICB.PHY.COUNTER It indicates PHYSICAL.IO activity unless it is set to zero. PHYSICAL.IO increments this field by one when invoked and decrements upon exiting.

ICB.PHY.SAVE.FCB This field contains the pointer to the top of the FCB queue of FCBs needing processing.

ICB.VM.FLAGS This contains a group of flags characterizing the rollin and rollout of a partition.

Data type description:

01 ICB.VM.FLAGS	BIT(8)
02 ICB.ROLLIN.ROLLOUT	BIT(2)
02 ICB.ROLLOUT.DISALLOW	BIT(4)
03 ICB.PERM.RO.DISALLOW	BIT(1)
03 ICB.SCL.RO.DISALLOW	BIT(1)
03 ICB.DCCH.RO.DISALLOW	BIT(1)
03 ICB.RO.AND.COMP.DISALLOW	BIT(1)
02 ICB.PLAY.ALONE.PART	BIT(1)

ICB.COMMON.DUMP.AREA This area is used by the COBOL Interpreter to save state when giving up control.

ICB.FILE.NB.ASSIGNMENT Each bit of this field corresponds to a file. This field is used to assign file numbers at OPEN time. The position of the first reset bit determines the file number of the file being opened, if any. The bit is reset at CLOSE time unless the file was half-closed. The first bit (bit 0) corresponds to the program file, the second bit (bit 1) to the virtual file, the third bit to file number three and so on.

ICB.FIB.REQSTED.ATTENTION This is a 72-bit field set by PHYSICAL.IO to indicate that further processing by LOGICAL.IO must be done before processing of the external communicate will be complete. Each bit corresponds to a file with the same relationship as the file number. That is, the first bit (bit 0) will correspond to the program file, the second bit (bit 1) to the virtual file, the third bit to file number three and so on.

ICB.FILE.LIST Each possible file for a program is represented by one byte. This byte contains the index of the data segment containing the corresponding FIB. Files are in position according to their file number in ICB.FILE.NB.ASSIGNMENT.

ICB.NON.COMMON.DUMP.AREA This area is used by the MPL Interpreter to save state when giving up control.

ICB.EOJ.DUMP.AREA This field contains various EOJ parameters like the EOJ date (in julian format), the EOJ time (HHMMSS), etc.

Data type description:

01	ICB.EOJ.DUMP.AREA	BIT(108)
02	ICB.EOJ.DATE	BIT(20)
02	ICB.EOJ.TIME	BIT(24)
02	FILLER	BIT(4)
02	ICB.EOJ.PROGRAM.STATUS	BIT(24)
02	ICB.EOJ.SAVE.ACTUAL.CPA	BIT(32)
02	ICB.EOJ.VARIANT	BIT(4)

ICB.DATAKOM This area is used for the processing of data comm communicates.

Data type description:

01	ICB.DATACOM	CHAR(16)
02	ICB.NEXT.ACTION	BIT(4)
02	ICB.ERROR.OPTION	BIT(1)
02	ICB.MCS.GONE	BIT(1)
02	ICB.PROGRAM.DSDP	BIT(1)
02	FILLER	BIT(1)
02	ICB.DATACOM.NAME	CHAR(12)

ICB.LOCKED.SLICE.SIZE	This is the size of the LOCKED SLICE.
ICB.CURRENT.VM.POINTER	It contains the bit offset, relative to the start of a partition, of the available reserved space in that partition.
ICB.VM.MIN.SIZE	It indicates the size, in bytes, of the partition size required to roll-in a partition. This includes the ICB, Locked Slice and necessary data and code segments. It is calculated at roll-out time.
ICB.VM.ACTUAL.SIZE	It contains the size of the partition in bytes. This is always an even number.
ICB.VM.MY.RESERV	It indicates the portion of the available space already reserved (in bytes).
ICB.VM.LENGTH	It contains the length, in bytes, of the available memory in the partition. This is always an even number.
ICB.VM.CUR.SEG.OFFSET	It is used by VM, when loading code or data segments into memory, to store the PST or DST index of this code or data segment.
ICB.VM.SAVE.AREA	This area is used by VM to store the relative record number in the virtual file of the first FIB rolled-out to disk. This avoids loosing space in the virtual file when opening and closing the same file frequently.
ICB.VM.PHYSIO.SAVED. ADDR	This area is used by VM to store the absolute bit address in the case of a roll-out or a DP.
ICB.MIX.VM.USE	It is used by VM for roll-in/roll-out decisions.

Data type description:

01	ICB.MIX.VM.USE	BIT(24)
02	ICB.MIX.IN.OUT	BIT(4)
02	ICB.MIX.IN.OUT.TIME	BIT(20)

ICB.INTERP.INDEX	It contains the Memory Assignment Table index of the interpreter employed by the program represented by this ICB.
ICB.ENTRY.SEGM	It indicates the index of the code segment containing the first executable S-OP.
ICB.ENTRY.OFFSET	It contains a byte offset into the code segment indicated by the field ICB.ENTRY.SEGM. This points to the first executable S-OP.
ICB.PST.PTR	This is the pointer to the Program Segment Table in bits, relative to the start of the ICB.
ICB.PST.LGTH	It contains the length of the Program Segment Table in bytes.
ICB.DST.PTR	It contains a pointer to the Data Segment Table in bits, relative to the start of the ICB.
ICB.DST.LGTH	It contains the Data Segment Table length in bytes.
ICB.TCB.PA.PTR	It contains a pointer to the Task Control Block Preset Area in bits, relative to the start of the ICB.
ICB.TCB.PA.LGTH	It contains the Task Control Block Preset Area length in bytes.
ICB.CNTL.STACK.PTR	It contains a pointer to the interpreter perform stack. The Task Scheduler returns to a point in the interpreter based on the last entry in this stack.
ICB.CNTL.STACK.LGTH	It contains the length of the interpreter perform stack.
ICB.CCB.PA.PTR	It contains the bit address, relative to the ICB of the Code Control Block Preset Area.
ICB.CCB.PA.LGTH	It contains the length of the Code Control Block Preset Area in bytes.
ICB.IFNB.PTR	It contains the bit address relative to the start of the ICB of the Internal File Name Block.
ICB.IFNB.LGTH	It contains the length of the Internal File Name Block in bytes.
ICB.DUAL.DC.FLAGS	It is used in a dual processor environment.

Data type description:
01 ICB.DUAL.DC.FLAGS
02 ICB.TEMP.NOT.FOR.SLAVE BIT(1)

	02 ICB.RUN.ON.MASTER	BIT(1)
	02 ICB.IN.USE.ON.SLAVE	BIT(1)
	02 ICB.DC.RELATED.TASK	BIT(1)
ICB.MIX.WAIT.COUNTER	It is used to indicate the number of seconds a program has still to be delayed, according to a WAIT communicate issued.	
ICB.SCL.SAVE.AREA	This area is used by SCL to save necessary fields when giving up control; that is, when handling AD and SF intrinsics.	
ICB.PROG.FIB	It contains the Program file FIB.	
	Data type description:	
	01 ICB.PROG.FIB	BIT(2976)
	02 FILLER	BIT(1360)
	02 ICB.PROG.PACK.ID	BIT(56)
	02 ICB.PROG.FILE.ID	BIT(112)
	03 FILLER	BIT(104)
	03 ICB.OWNER.HASH.VALUE	BIT(8)
	02 FILLER	BIT(488)
ICB.VIRT.FIB	It contains the FIB of the Virtual file.	
	Data type description:	
	01 ICB.VIRT.FIB	BIT(2976)
	02 FILLER	BIT(1360)
	02 ICB.VIRT.PACK.ID	BIT(56)
	02 ICB.VIRT.FILE.ID	BIT(112)
	03 FILLER	BIT(104)
	03 ICB.EXECUTING.HASH.VALUE	BIT(8)
	02 FILLER	BIT(488)
ICB.PRIORITY.EXTENSION	Reserved for expansion	
ICB.USER.TYPE	Reserved for expansion	
ICB.SHARED.USER.NB	Reserved for expansion	
ICB.SLAVE.INTERNAL.CPA	This field contains the ICB.INTERNAL.CPA which is destroyed when control is given to PHYSICAL.IO upon detection of an ICB.PHY.SAVE.FCB.	

COMMUNICATE PARAMETER AREA

This is the last external communicate sent by the interpreter to LOGICAL.IO on behalf of the user's program.

The information analyzed here is taken from the fields ICB.ACTUAL.CPA and ICB.FETCH.VALUE in the Interface Control Block. This fetch value reflects the success or failure of the communicate analyzed here.

VERB This gives the hexadecimal value and analysis of the last verb executed; a list of those verbs is given below, class by class.

CLASS	VERB VALUE	VERB SEMANTIC
CLASS A – file type I/O Where appropriate class A verbs are made conditional by adding 1 to the assigned value; for example, 89 is a conditional delete.	80	test status
	82	read (not console)
	84	write (not console)
	86	rewrite
	88	delete
	8A	stream control
	8C	start
	8E	overwrite
	90	read-write
	92	read (console)
	94	write (console)
	96	get
	98	put
	9C	free block
	A2	read (shared)
	A4	write (shared)
	A6	rewrite (shared)
	A8	delete (shared)
	AA	overwrite (shared)
	CLASS B – file assignment	01
02		close
CLASS C – field oriented	11	zip
	12	display
	13	zip and display
	14	pause
	15	zip and pause
	16	display and pause
	17	zip, display and pause
	1A	conditional display
	1B	zip and conditional display
	1C	display without logging
	1D	system display
	20	accept
	21	super accept
	30	MCS control
	CLASS D – data communication	31
32		MCS redefinition
33		user data comm
34		MCS DCP oriented implementation
35		dependent

CLASS E – miscellaneous	40	date-time
	41	terminate
	42	wait
	43	system status
	44	complex wait
	49	append text
	4A	SPO size
	50	log on
	51	log off
	52	run (unconditional)
	53	run (conditional)
	54	read messages queue

OBJECT

The object definition is dependent on the verb specified, its values are detailed class by class.

CLASS
CLASSES A AND B

VERB

OBJECT MEANING

The hexadecimal number given here is the number of the Data Segment which contains the File Information Block for the file to which the communicate applies. When the number has been converted to decimal, one can search for the data segment in the "DATA SEGMENTS" section of the DP.ANALYZER to determine which file is involved in the communicate.

CLASS C

The hexadecimal number is the number of the Data Segment containing the text area.

CLASS D

30-34

The verb of a CLASS D communicate is used to specify a general type of function; the second byte of the ICB.ACTUAL.CPA, decoded as the OBJECT, defines the actual function. See the manual DATA COMMUNICATIONS SUBSYSTEM form no.1090909 for a complete description of those values.

	35	02 – DC job goes to EOJ 03 – MCS job goes to EOJ 05 – MCS job goes to BOJ
CLASS E	40	This is the index, in hexadecimal, into the DST of the related program.
	41	If bit 0 is set, two bytes are passed back to the initiating task. If bit 1 is set, the VM file is locked.
	42,44	The OBJECT field does not exist.
	43, 4A	The hexadecimal number is the number of the Data Segment containing the response area.
	49	The hexadecimal number is the number of the Data Segment containing the text area.
	52, 53	The hexadecimal number is the number of the Data Segment for the data area containing the text.
	54	The hexadecimal number is the number of the Data Segment for the data area receiving the text.

ADVERB

The adverb definition is dependent on the verb specified. Its values are listed below, byte by byte for each class.

Note that for OPEN and CLOSE communicates, the adverb field has no meaning, as in this case all attributes are stored in the file's FPB; see the fields ADVERB FOR OPEN and ADVERB FOR CLOSE, in the section entitled FILE ATTRIBUTES FROM FPB.

CLASS	VERB	ADVERB MEANING
CLASS A	80 < VERB < = 8F	<p>If the file organization is sequential and ((the accessmode is random and the VERB > 81) or (the access mode is sequential and the VERB = START)) then:</p> <p>BYTES 0, 1, 2 represent the relative record number of the required record.</p> <p>If the file organization is indexed and the VERB = START or the file organization is indexed, the accessmode is random and the VERB = READ, then only byte 0 is significant; the relevant bit positions are:</p> <p>bits 0-5 are always set.</p> <p>bits 6-7</p> <p>= 11 for READ equal, START equal</p> <p>= 10 for READ next, START greater than or equal</p> <p>= 01 for READ next, START greater than</p> <p>= 00 for READ next, START next</p>
	90 < = VERB < = 99	<p>byte 0 – this is the DST index of the work area segment.</p> <p>bytes 1, 2 – this is the offset of the work area within the named segment.</p> <p>bytes 3, 4 – this is the length of the work area.</p>
	9A	bytes 0-2 – they give the binary sector address
	A0 < VERB < = AF	These verbs map onto the corresponding verbs in the range 80-8F.
CLASS C	10 < VERB < 1F	<p>bytes 0, 1 – this is the offset into the segment of the text area.</p> <p>bytes 2,3 – this is the length of the text area in bytes.</p>

CLASS	VERB	ADVERB MEANING
	20	<p>bytes 0,1 – this is the offset into the segment of the text area.</p> <p>bytes 2, 3 – this is the maximum length of the text area in bytes.</p>
	21	<p>bytes 0, 1 – this is the offset into the code of the text area .</p> <p>bytes 2, 3 – this is the maximum length of the text area in bytes.</p> <p>byte 4 – if bit 0 is set, two bytes are passed back to the initiating task.</p> <p>bytes 7, 8 – they contain the message passed back to the initiating task .</p> <p>This field describes the entity on which the data communication communicate has to operate. See the manual DATA COMMUNICATIONS SUBSYSTEM form no. 1090909 for a complete description of the values.</p>
CLASS D		
CLASS E	40	<p>bytes 0, 1 – this is the offset in the segment of the date or time value.</p> <p>byte 2 – only bits 0 and 1 are significant. The possible values are: 00 – the date is stored in BCD in the format YYMMDD 01 – the date is stored in BCD in the format YYDDDD0 10 – the time is stored in BCD in the format HHMMSS 11 – the time is stored as a binary number of tenths of a second.</p>

(continued)

CLASS	VERB	ADVERB MEANING
	41	bytes 0, 1 – they contain the message for the initiating task, if bit 0 of the OBJECT is set.
	42	bytes 0, 1 – they contain the binary number of seconds.
	43	bytes 0, 1 – they contain the offset into the segment for the response area. bytes 2, 3 – they contain the maximum length of the response area in bytes.
	44	bytes 0, 1 – they give the binary number of seconds. byte 2 – bit 0 is set if the NOLOCK option has been specified. bytes 3-9 – each byte contains 2 event class indicator values, the first 0 terminates the list. The event class indicators are listed in priority order. byte 10 – it contains the DST index of the response area. bytes 11, 12 – they contain the bit offset into the segment of the response area. bytes 13, 14 – they contain the maximum length of the response area in bytes.
	49	bytes 0,1 – they contain the bit offset into the segment of the text area. bytes 2,3 – they contain the length of the text area in bytes.
	4A	bytes 0,1 – they contain the bit offset into the segment for the response area.
	52, 53, 54	bytes 0,1 – they contain the bit offset into the segment for the data area. bytes 2,3 – they contain the length of the data area in bytes.

FETCH VALUE The hexadecimal value is the fetch value returned to the interpreter by LOGICAL.IO. This field is three bytes long; all the values are displayed in hexadecimal.

BYTE 0	BYTE 1	BYTE 2	MEANING
80			fatal error, for example, invalid communicate; if bytes 1 and 2 are equal to Fxxx, see section 5-2 System Dependent Fetch Values in the B 1000 CMS 3.03 Release letter; if not, bytes 1 and 2 contain an event number (see section 7 in the CMS Systems Software Operation Guide form no. 2007258-004).
40			a resource is temporarily unavailable (conditional failure). bytes 1 and 2 contain the event number corresponding to the message that is printed on the SPO. If no message corresponds to the conditions, bytes 1 and 2 contain 0 with the exception of the BLOCK-LOCKED condition, when they contain @9020@.
20 and the previous communicate was a file access	10		the end of file condition is encountered on input for sequential access
	20	00	an invalid key has been processed
		10	a sequence error on output to an indexed file is detected
		20	a duplicate key exists on an indexed file
		30	no such record exists (attempt to read beyond the end of file)
		40	there is a boundary violation (attempt to write beyond the allocated area)
	30	00	a permanent hardware error is detected on this file
		10	a read error on the data file is detected
		20	a write error on the data file is detected
		30	a read error on the key file is detected
		40	a write error on the key file is detected
	40		a block count error is detected on close
	90	10	the communicate is defined but not implemented

(continued)

BYTE 0	BYTE 1	BYTE 2	MEANING
		20	a block-lock condition is encountered
20 and the previous communicate was a ZIP	00	10	the program file is not found
		20	the interpreter file is not found or there is a release level mismatch
		30	there is no memory available
		40	there is no user disk or the directory is full
		50	the mix is full
		60	there is a usercount error
		70	there is duplicate pack
		80	the load request is invalid
		90	an MCS is already present
		A0	a disk error is detected
		B0	a code file error is detected
		C0	the data communication load request is invalid
		D0	the zipped program has been DS'ed
		D1	the zipped program has been DP'ed
		F0	a suitable mix is required
		F1	dual alphabet/reverse escapement is not supported
		F2	there is insufficient real store
		F3	the disk is locked
		F4	a faulty file equate is detected
	80		an invalid non-load request is detected
20 and the previous communicate was a complex wait	90	30	unknown or non-implemented event in event list
		40	SCLQ wait requested by a task not controlling remote system operation
		50	SUBQ wait requested on no queues attached or all queues have become detached during the wait.

(continued)

BYTE 0	BYTE 1	BYTE 2	MEANING
10	00	00	the queue is empty on a conditional receive or a no space condition is encountered on a conditional send.
		01	a limit exceeded condition is encountered on a conditional send
00	00	00	no error is detected

PERFORM STACK/CONTROL STACK

PERFORM STACK (COBOL and RPG)

Each "PERFORM" in COBOL and subroutine call in RPG causes an entry in the Perform Stack. Each "EXIT" in COBOL and subroutine end in RPG causes an entry to be removed. Therefore, the Perform Stack reflects the level of nesting in a program.

The 'PERFORM STACK' area of an RPG or COBOL DP.ANALYZER listing reflects the level of nesting at the time of the abnormal terminate. All active PERFORMs or subroutines will have entries on the stack.

The last entry listed reflects the top of the stack; that is, the most recently activated paragraph or subroutine.

NOTE

Due to the nature of RPG and the "program cycle" concept, a lot of code is generated by the compiler which the user does not know about. It is, therefore, common to have entries on the Perform Stack though no subroutines are employed in the user's source.

- KEY** This is used by the interpreter to correlate PERFORM entrance and exit statements for COBOL programs. In RPG this field is always 0.
- SEGMENT** This gives the code segment number of the code segment containing the calling PERFORM or subroutine.
- DISPL** This gives the displacement into the code segment given in "SEGMENT" at which the calling PERFORM or subroutine is found.
- SEGMENT/DISPL can be related to the corresponding line number in the source by using \$ options to obtain a map.
- In RPG, the \$ options "\$MAP", "\$PARMAP" or "\$XMAP" will cause the "LINE MAP", needed to correlate SEGMENT/DISPL to line number, to be generated.
- In COBOL, the "\$LINE-MAP" option will generate the needed map.

LINE COUNT

This gives the contents of the line count register at the time of the calling PERFORM or subroutine. To obtain this information, certain options must be specified at compile time:

- for COBOL, the “\$LINE-CODE” option must have been specified;
- for RPG, column 15 of the H card specification must have been set to 1.

With this information, the instruction containing the calling PERFORM or subroutine is identified.

CONTROL STACK (MPL)

When a procedure call is encountered, information about the currently active procedure is stored as an entry on the Control Stack. This information allows control to return to this procedure when the called procedure is exited. An exit from a procedure, therefore, causes an entry to be removed from the Control Stack.

The ‘CONTROL STACK’ area of an MPL DP.ANALYZER listing displays the state of the Control Stack at the time of the abnormal terminate, and therefore reflects the level of nesting at this time. There will be an entry in the Control Stack for every active procedure with the exception of the procedure being executed at the time of the abnormal terminate. Corresponding information about the currently executing procedure is found in the ‘RUNNING PARAMETERS’ area of the DP.ANALYZER listing. The last entry listed in the Control Stack reflects the top of the stack; that is, the last calling procedure.

Any executable instruction can be addressed using a code segment number and offset into that segment. The offset of the start of each procedure within a code segment, relative to that code segment, is also stored in that code segment, and associated with a procedure number. A reference to a code segment number (PSN) and a procedure number (SPN) within that code segment will, therefore, uniquely address a procedure.

Correlation between the procedure number/return address and the line number in the source may be found at the end of the compiler listing if the \$XMAP option has been specified.

LEXICAL LEVEL This is the lexical level of the procedure whose state is saved by this entry. This corresponds to the lexical level given in the first column on the left-hand side of a source listing generated at compile time.

SEGMENT (PSN) This is the code segment number in which the procedure is located.

PROCEDURE (SPN)	This is the procedure number within the code segment of the procedure
RETURN ADDRESS	This is a byte offset from the start of the code segment representing the address of the next executable instruction to which control will return when the procedure is reinstated.
REG1/REG2	This reflects the most frequently used lexical level of the procedure. This information is useful only when the code segment has to be decoded.

PROGRAM SEGMENT TABLE

The 'PROGRAM SEGMENT TABLE' section of a DP.ANALYZER listing displays information about a user program's code segments.

All the information listed in this section is read from the structure "PROGRAM SEGMENT TABLE" allocated in the Locked Slice. A complete description of the layout of this structure is stated in the chapter entitled LOCKED SLICE; each member of the structure has been labelled.

For each field listed, its name shown on the DP.ANALYZER listing, its name shown in the PROGRAM SEGMENT TABLE structure and its description, are stated.

SEGMENT NUMBER	It represents the index number of the segment.
SEGM TYPE	SEG.DESCR.TYPE It contains the value 00.
IN VIRTUAL	IN.VIRTUAL.FILE It indicates, when set, that the segment has already been copied to the virtual file and therefore has a disk address assigned (relative record number).
TO BE LOADED	TO.BE.LOADED It indicates, when set, that the corresponding segment has to be loaded by VM.
IN USE	IN.USE It indicates, when set, that the segment is in use and may not be rolled-out or must be rolled-in if not present before giving the control back to the interpreter.
IN CORE	IN.CORE It indicates, when set, that the segment is present in memory.
LOCK IN MEMORY	LOCK.IN.MAIN.STORE It indicates, when set, that the segment is locked in memory.
READ/WRITE	READ.WRITE.SEG It indicates, when set, that the segment is of the READ/WRITE type; that is, the user may read and write in it.

DISK ADDRESS	SEG.DESCR.DSK.ADD This gives the relative sector number of the segment in the Program file or in the Virtual file.
SEGM LENGTH	SEG.DESCR.LGTH This gives the length of the segment in bytes.
MEMORY ADDRESS	SEG.DESCR.MEM.ADD This gives the memory address, in bits, of the segment relative to the ICB start address.
DESCR FN	SEG.DESCR.FN This gives the file number in the ICB.FILE.LIST, used at half close time to keep a link with the file.
ROLL-IN COUNTER	SEG.DESCR.ROLL.IN.COUNTER It indicates the number of times the segment has been rolled in.

DATA SEGMENT TABLE

The 'DATA SEGMENT TABLE' area of a DP.ANALYZER listing displays information about a user program's data segments. The format of this table is exactly the same as for the Program Segment Table except for the following fields:

SEGM TYPE	SEG.DESCR.TYPE It indicates the segment type. The possible values are 01 for an FIB and 00 for anything else.
SEGM LENGTH	SEG.DESCR.LGTH If the segment contains an FIB and the file is not opened, it indicates the number of the FPB segment. Otherwise, it gives the length, in bytes, of the segment.

INTERNAL FILE NAME BLOCK

The INTERNAL FILE NAME BLOCK section contains a table of elements, one per file declaration in the program.

FIB.NB	It gives the index number of the data segment which contains the associated FIB.
FPB.NB	It gives the index number of the data segment which contains the associated FPB.
INTERNAL FILE NAME	It gives the internal file name.

FILE INFORMATION

Information listed in this section is read from a DATA SEGMENT containing

- the File Information Block (FIB), if the file is opened.
- the File Parameter Block (FPB) in all other cases.

The correspondence between the Data Segment and the described FIB or FPB is stated explicitly in the Data Segment section. It may also be found using the Internal File Name Block allocated in the Locked Slice; moreover, the latter structure IFNB may be used as an index to the Data Segment Table to determine the state of the file: opened or closed.

This chapter is divided into two main sections:

- one describes the parameters displayed for an opened file; the File Control Block (FCB), file's buffers, control information for its buffers, extended information for indexed files, data file FIB for an indexed file are listed.
- one describes the parameters displayed for a closed file, the FPB is listed.

In this section, for each field displayed, the following is stated:

- its name, as shown on the DP.ANALYZER listing.
- its name, as shown in the FIB or FPB structure description detailed in the DATA SEGMENT section.
- its description.

Note that for each compound field, its data type description has been given as a memorandum; a complete description of the members of those structures is given in the manual "B 1000 CMS MEMORY DUMP ANALYSIS USER'S GUIDE, form no. 2018909.

INFORMATION FOR AN OPENED FILE

INFORMATION FROM FCB

In the 'FILE INFORMATION' area of a DP.ANALYZER listing, the first three rows of information refer to the File Control Block (FCB) of the file.

CODE POINTER	FCB.CODE.PTR It contains the absolute bit address to which a return is effected after completion of the current I/O. If bits 0-3 are zero, it indicates a return to the PHYSICAL I/O resident code, otherwise the value in bits 0-3 is the PHYSICAL I/O overlay identification.
BUFFER ADDRESS	FCB.CUR.BUF.ADDR This field contains the absolute bit address of the buffer area for the last or current I/O operation.

BUFFER LENGTH	FCB.CUR.BUF.LENGTH This is the length in bytes of the buffer for the current or last I/O operation.																																						
OP CODE	FCB.OP.CODE This is the actual opcode sent to the I/O control for the current or last I/O operation. See the appropriate technical manual referring to the device.																																						
DCB ADDRESS	FCB.DCB.ADDRESS This field contains the absolute bit address of the Device Control Block (DCB) to which this FCB is currently attached.																																						
FILE STATUS	FCB.FILE.STATUS This field contains a set of one bit flags used by PHYSICAL.IO to control I/O activity. Data type description: 01 FCB.FILE.STATUS <table border="0" style="margin-left: 2em;"> <tr><td>02 FCB.USE.SYSMEM</td><td>BIT(1)</td></tr> <tr><td>02 FCB.CUR.USE.SYSMEM</td><td>BIT(1)</td></tr> <tr><td>02 FCB.USE.AVAIL.TABLE</td><td>BIT(1)</td></tr> <tr><td>02 FCB.CUR.USE.AVAIL.TABLE</td><td>BIT(1)</td></tr> <tr><td>02 FCB.NEW.OLD</td><td>BIT(1)</td></tr> <tr><td>02 FCB.OPENING</td><td>BIT(1)</td></tr> <tr><td>02 FCB.CLOSE</td><td>BIT(1)</td></tr> <tr><td>02 FCB.USE.FPB.VN</td><td>BIT(1)</td></tr> <tr><td>03 FCB.TRANSLATE.COMPLETE</td><td>BIT(1)</td></tr> <tr><td>02 FCB.SEARCH</td><td>BIT(1)</td></tr> <tr><td>02 FCB.ERRB</td><td>BIT(1)</td></tr> <tr><td>02 FCB.QUEUED</td><td>BIT(1)</td></tr> <tr><td>02 FCB.SUSPENDING.TASK</td><td>BIT(1)</td></tr> <tr><td>02 FCB.ACTIVE</td><td>BIT(1)</td></tr> <tr><td>02 FCB.IN.USE</td><td>BIT(1)</td></tr> <tr><td>02 FCB.DISPLAY</td><td>BIT(1)</td></tr> <tr><td>02 FCB.READING.LABEL</td><td>BIT(1)</td></tr> <tr><td>02 FCB.HALF.CLOSED</td><td>BIT(1)</td></tr> <tr><td>02 FILLER</td><td>BIT(7)</td></tr> </table>	02 FCB.USE.SYSMEM	BIT(1)	02 FCB.CUR.USE.SYSMEM	BIT(1)	02 FCB.USE.AVAIL.TABLE	BIT(1)	02 FCB.CUR.USE.AVAIL.TABLE	BIT(1)	02 FCB.NEW.OLD	BIT(1)	02 FCB.OPENING	BIT(1)	02 FCB.CLOSE	BIT(1)	02 FCB.USE.FPB.VN	BIT(1)	03 FCB.TRANSLATE.COMPLETE	BIT(1)	02 FCB.SEARCH	BIT(1)	02 FCB.ERRB	BIT(1)	02 FCB.QUEUED	BIT(1)	02 FCB.SUSPENDING.TASK	BIT(1)	02 FCB.ACTIVE	BIT(1)	02 FCB.IN.USE	BIT(1)	02 FCB.DISPLAY	BIT(1)	02 FCB.READING.LABEL	BIT(1)	02 FCB.HALF.CLOSED	BIT(1)	02 FILLER	BIT(7)
02 FCB.USE.SYSMEM	BIT(1)																																						
02 FCB.CUR.USE.SYSMEM	BIT(1)																																						
02 FCB.USE.AVAIL.TABLE	BIT(1)																																						
02 FCB.CUR.USE.AVAIL.TABLE	BIT(1)																																						
02 FCB.NEW.OLD	BIT(1)																																						
02 FCB.OPENING	BIT(1)																																						
02 FCB.CLOSE	BIT(1)																																						
02 FCB.USE.FPB.VN	BIT(1)																																						
03 FCB.TRANSLATE.COMPLETE	BIT(1)																																						
02 FCB.SEARCH	BIT(1)																																						
02 FCB.ERRB	BIT(1)																																						
02 FCB.QUEUED	BIT(1)																																						
02 FCB.SUSPENDING.TASK	BIT(1)																																						
02 FCB.ACTIVE	BIT(1)																																						
02 FCB.IN.USE	BIT(1)																																						
02 FCB.DISPLAY	BIT(1)																																						
02 FCB.READING.LABEL	BIT(1)																																						
02 FCB.HALF.CLOSED	BIT(1)																																						
02 FILLER	BIT(7)																																						
DEV.KIND	FCB.DEVICE.KIND This value indicates the type and capabilities of the device to which this FCB is connected.																																						
RETRY COUNT	FCB.RETRY.COUNT It contains the number of retries attempted so far for the current I/O. It is reset to zero when an I/O is successful.																																						
CHANNEL NR	FCB.CHANNEL This is the physical channel number to which the file associated with this FCB is attached.																																						

FILE NR	FCB.FILE.NR This is the internal file number assigned to the file associated with this FCB. The file number is assigned by LOGICAL.IO when the file is opened and will be in the range @03@ - @48@ . A value of @FF@ indicates that PHYSICAL.IO is currently using the FCB for a SEARCH or for loading one of its overlays.
FIB ADDRESS	FCB.FIB.ADDRESS This is the absolute bit address of the FIB of the file associated with this FCB.
MIX NUMBER	FCB.MIX It contains the mix number of the program owning the file associated with this FCB.
QUEUE LINK	FCB.Q.LINK This is the absolute bit address of the next FCB in either the initiate queue or the completion queue.
RESULT DESCR.	FCB.RD It contains the result descriptor returned from an unsuccessful I/O operation or the last result descriptor with an I/O exception from an I/O operation that succeeded after one or more retries.
FCB.SAVE.SCRATCHPADS	FCB.SAVE.SCRATCHPADS It contains the length, begin address and end address of an area to be allocated or de-allocated from the Available Table. It is used by PHYSICAL.IO to update the disk file header.
FCB.RETURN.LIST	FCB.RETURN.LIST This is the list of return addresses queued by PHYSICAL.IO while processing an I/O communicate.
FCB.RETURN	FCB.RETURN This contains the pointer to the FCB.RETURN.LIST.
MY.USE OTHER.USE	FCB.MYUSE FCB.OTHERUSE Those fields are valid only for a disk FCB. The first field contains the value of MYUSE with which the file was opened (input, output, input-output).

The second field contains the value of OTHERUSE with which the file associated with this FCB was opened.

Data type description:

01 FIELD.DESCRPTION	
02 FCB.MYUSE	BIT(2)
03 TO.USE.AS.OUTPUT	BIT(1)
03 TO.USE.AS.INPUT	BIT(1)
02 FCB.OTHERUSE	BIT(3)
02 FILLER	BIT(3)

The possible FCB.OTHERUSE values are:

- 110 or 100 meaning free access
- 010 or 000 meaning lock access
- 001 meaning shared access.

MEMORIZE
DCB

FCB.MEMORIZE.DCB

This field is valid only for a disk FCB. The address of a DCB is stored here by PHYSICAL.IO when needed.

DISK ADDRESS

FCB.DISK.ADDRESS

This field is valid only for a disk FCB. It contains the absolute disk address of the sector on which the current or last I/O operation took place.

CUR.AREA
ADDRESS

FCB.CUR.AREA.AD

This field is valid only for a disk FCB. It contains the absolute sector address of the current area of the disk file.

DFH.ADDRESS

FCB.DFH.ADDRESS

This field is valid for a disk FCB. It contains the absolute sector address on disk of the disk file header for the file which is associated with this FCB.

DIR.ENTRY
ADDRESS

FCB.DIRENTRY.ADDRESS

This is valid only for a disk FCB. It contains the absolute disk address of the File Name List sector which contains the entry for the file associated with this FCB.

AREA NR

FCB.AREA.NR and FCB.AREA.COUNT

This is the number of the area FCB.AREA.NR (@0@-@F@) in which the last or current I/O operation (READ or WRITE) occurred.

FCB.AREA.COUNT represents the number of areas required to allocate the whole file.

	Data type description:	
	01 AREA NR	
	02 FCB.AREA.NR	BIT(4)
	02 FCB.AREA.COUNT	BIT(4)
DIR.INDEX	FCB.DIRENTRY.IX	
	This field is valid only for a disk FCB. It contains the index within the File Name List sector of the entry for the file associated with this FCB. The range is @0@-@A@ (there are 11 entries in each of the File Name List).	
SECTOR NUMBER	FCB.SECTOR.NR	
	This field is valid only for a disk FCB. It contains the absolute disk address of the sector currently under examination during a SEARCH. If the SEARCH has terminated, this will be the absolute disk address of either the sector containing the search argument, if the search was successful, or the first sector after the search area, if the search was unsuccessful.	
RETR.ITEM	FCB.RETRIEVED.ITEM	
	It contains the absolute disk address giving the location of an item on which a successful SEARCH has been performed.	
SEARCH ST.	FCB.SEARCH.STATUS	
	This field is valid only for a disk FCB. The format of this field is:	
	bits 0-3 are not used	
	bit 4: if set, the device is a disk pack	
	bit 5: if set, the search has terminated	
	bit 6: if set, the search was successful	
	bit 7: if set, the indexed search has terminated.	
SEARCH.OP.	FCB.SEARCH.OP.CODE	
	This field is valid only for a disk FCB. It contains the opcode for a SEARCH. Possible values are:	
	@01@ for a disk directory search	
	@03@ for an indexed search (key file)	
SEARCH SKIP	FCB.SEARCH.SKIP	
	The first digit gives, in hexadecimal format, the number of entries to skip in the directory for a SEARCH SKIP.	

	The maximum value is @B@. The second digit is used to save the value of the first digit in the case of a retry.
ARGUM.ADDRESS	FCB.ARGUMENT.ADDRESS This is the absolute bit address of a string which is the argument for a SEARCH operation.
SPACE.TO ALLOC.	FCB.SPACE.TO.ALLOCATE This field is valid only for a disk FCB. It contains the length in sectors needed for allocation of the next area of a disk file.
CURRENT.AREA RANGE	FCB.CUR.AREA.RANGE This field is valid only for a disk FCB. It consists of two 24 bit fields which contain the block numbers of the first and last blocks within the current area. It is used to determine whether a READ or WRITE points outside the current area.

FILE ATTRIBUTES FROM FPB

IMPLEMENTATION LEVEL NUMBER	FIB.FPB.IMPL.LEVEL.NB It contains the implementation level number set by the compiler.
MULTIFILE-ID	FIB.FPB.PACK.ID It contains the name of the device containing the file, if applicable.
FILE-ID	FIB.FPB.FILE.ID It contains the file name.
FILE-ID – HASH.VALUE	FIB.FILE.NB It contains the hash code of the user code site. The value is @20@.
REEL NUMBER	FIB.FPB.REEL.NB It contains the current reel or cassette number. It is non-significant when the device is not a tape.
FILE TYPE	FIB.FPB.FILE.TYPE The different file type values are: 00 Normal Data File 01-0E Source Language File

0F Source Library File
10-12 Ordinary Program (S-code)
13 Protected System Program for
example, SYS-SUPERUTL
1C-1F Interpreter
20 SYSMEM File
21 SYSLANGUAGE File
22 MCP-related File for example,
SYSCONFIG
30 DUMP File
31 LOG File
80 Indexed Data File
81 Indexed Key File
A0 Printer Backup File

HIGHEST RECORD NUMBER	FIB.FPB.HIGHEST.RECD.NB It contains the relative record number of the highest record within the file, that is, closest to End-Of-File.
DEVICE KIND	FIB.FPB.DEVICE.KIND This is the hexadecimal value and analysis of the device containing the file.
WORK AREA SEGMENT NUMBER	FIB.FPB.WA.SEGM.NB It contains the Data segment table index of the segment containing the work area for this file.
OFFSET OF A WORK AREA WITHIN SEGMENT	FIB.FPB.WA.OFFSET It contains the offset into the work area data segment where the work area is found.
RECORD SIZE (BYTES)	FIB.FPB.REC.SIZE It contains the record size in bytes.
BUFFER SIZE (BYTES)	FIB.FPB.BUF.SIZE It contains the size of a buffer (block) in bytes.
MAXIMUM FILE SIZE (RECORDS)	FIB.FPB.MAX.FILE.SIZE It contains the maximum number of records the file can contain.

NUMBER OF BUFFERS	<p>FIB.FPB.NB.BUF</p> <p>It contains the number of buffers requested for the file. However, for some file types this field value is fixed. If the file is</p> <ul style="list-style-type: none"> - SHARED, this field is set to 1; - RANDOM, if one buffer is specified, this field is set to 1; otherwise it is set to 2; - a backup file, this field is set to 2. 																								
FLAGS	<p>FIB.FPB.FLAGS</p> <p>This is the group name for various flags associated with the file attributes.</p> <p>Data type description:</p> <table border="0" style="width: 100%;"> <tr> <td style="width: 5%;">01</td> <td style="width: 85%;">FIB.FPB.FLAGS</td> <td style="width: 10%;">BIT(8)</td> </tr> <tr> <td>02</td> <td>FIB.FPB.SPEC.FORM</td> <td>BIT(1)</td> </tr> <tr> <td>03</td> <td>FIB.FPB.DUP.ALLOWED</td> <td>BIT(1)</td> </tr> <tr> <td>02</td> <td>FIB.FPB.UPT.FILE</td> <td>BIT(1)</td> </tr> <tr> <td>02</td> <td>FIB.FPB.NO.LABEL</td> <td>BIT(1)</td> </tr> <tr> <td>02</td> <td>FIB.FPB.COND.FILE</td> <td>BIT(1)</td> </tr> <tr> <td>02</td> <td>FIB.FPB.PROG.FILE</td> <td>BIT(1)</td> </tr> <tr> <td>02</td> <td>FILLER</td> <td>BIT(3)</td> </tr> </table>	01	FIB.FPB.FLAGS	BIT(8)	02	FIB.FPB.SPEC.FORM	BIT(1)	03	FIB.FPB.DUP.ALLOWED	BIT(1)	02	FIB.FPB.UPT.FILE	BIT(1)	02	FIB.FPB.NO.LABEL	BIT(1)	02	FIB.FPB.COND.FILE	BIT(1)	02	FIB.FPB.PROG.FILE	BIT(1)	02	FILLER	BIT(3)
01	FIB.FPB.FLAGS	BIT(8)																							
02	FIB.FPB.SPEC.FORM	BIT(1)																							
03	FIB.FPB.DUP.ALLOWED	BIT(1)																							
02	FIB.FPB.UPT.FILE	BIT(1)																							
02	FIB.FPB.NO.LABEL	BIT(1)																							
02	FIB.FPB.COND.FILE	BIT(1)																							
02	FIB.FPB.PROG.FILE	BIT(1)																							
02	FILLER	BIT(3)																							
ADVERB FOR CLOSE	<p>FIB.FPB.ADV.CLOSE</p> <p>It contains the adverb describing the CLOSE attributes. The bits are defined as follows:</p> <ul style="list-style-type: none"> bit 1 - when set this bit indicates: <ul style="list-style-type: none"> - in the case of HALF-CLOSE, no rewind has to be performed - otherwise, a CHANGE REEL has to be performed leaving the file opened. bits 2,3,4 - 000 for HALF-CLOSE 011 for CLOSE with LOCK 101 for CLOSE with PURGE 111 for CLOSE with REMOVE 001 for CLOSE with RELEASE <p>All other combinations will be treated as HALF-CLOSE.</p> <ul style="list-style-type: none"> bit 5 - when set, a CRUNCH is requested. bit 6 - when set, a MERGE of the overflow region into the index region is requested. 																								
ADVERB FOR OPEN	<p>FIB.FPB.ADV.OPEN</p> <p>It contains the adverb describing the OPEN attributes. The bits are defined as follows:</p>																								

bit 0 – reserved for expansion.

bit 1 – when set, open with EXTEND has to be performed.

bits 2, 3, 4 –
 110 or 100 = free access
 010 or 000 = lock access
 001 = shared access.

bit 5 – when set, the backup is named.

bit 6 – when set, the usage is OUTPUT.

bit 7 – when set, the usage is INPUT.

bits 8, 9 – 01 = do not backup
 10 = must backup

bits 10, 11 – reserved for expansion.

bits 12, 13 –
 00 illegal
 01 RANDOM
 10 SEQUENTIAL
 11 STREAM

bits 14, 15 – reserved for expansion.

Bits 2, 3 and 4 are collectively referred to as OTHERUSE.

Bits 5, 8 and 9 are relevant only to printer files.

Bits 6 and 7 are collectively referred to as MYUSE.

Bits 12 and 13 are collectively referred to as ACCESSMODE. ACCESSMODE RANDOM is only applicable to disk and ICMD files.

CYCLE NUMBER	FIB.FPB.CYCLE Reserved for expansion.
GENERATION NUMBER	FIB.FPB.GEN.NB It contains the generation number of the file.
CREATION DATE (YYDDD)	FIB.FPB.CREAT.DATE It contains the file creation date in the format YYDDD.
LAST ACCESS DATE YYDDD	FIB.FPB.LAST.ACCESS.DATE It contains the last access date in the form YYDDD.
SPARE BYTES IN LAST STREAM	FIB.FPB.SPARE.CHAR It contains the number of spare bytes in the last record of a STREAM file.

SAVE FACTOR FIB.FPB.SAVE.FACT
 It indicates the number of days from the creation date for which the file is valid. The file is closed PURGE when it becomes no longer valid. The default is 999. The file is never closed PURGE if the save factor is 999.

CONTROL INFORMATION FROM FIB

FIB.AREA.ADDR.AND.LGTH FIB.AREA.ADDR.LGTH
 This 64 byte field is reserved for the 16 areas that might be used by the file. For each area allocated:

- the 2 first bytes contain the address, in allocation units, of the area
- the next 2 bytes contain the length of the area in allocation units.

FIB.COMMUNICATION.AREA FIB.COMM.AREA
 It is used to communicate with PHYSICAL.IO. It contains an internal communicate describing the desired action.

Data type description:

01 FIB.COMM.AREA	
02 FIB.COMM.VERB	BIT(8)
02 FIB.COMM.BLK.NB	BIT(24)
03 FIB.SEARCH.ADD	BIT(24)
02 FIB.COMM.MEM.BLK.ADD	
03 FIB.OPEN.CLOSE.BUF.AD	
04 FIB.BUF.AD	
05 FIB.KEY.MEM.ADD	BIT(24)
02 FIB.FILE.NB	BIT(8)
02 FIB.REPLY.WORD	BIT(24)
02 FIB.CUR.BLK.LGTH	BIT(24)
03 FIB.SEARCH.LGTH	BIT(24)

FIB.FILE.STATUS FIB.FILE.STATUS
 The first bit is set if the file has been half closed and must be half opened.

FIB.VARIOUS.FLAGS FIB.VN
 These bits are defined as follows:

bit 0 — it controls the OPEN/CLOSE messages. It prevents the messages from being displayed multiple times.

bits 1,2— they are used by PHYSIO

- bit 3 – it indicates that the file-id in the FIB should be used for display and not the file-id in the FPB
- bit 4 – it indicates that the buffer pointed to by PH.OUT.BUF.INDEX has been flushed
- bit 5 – it indicates the opening of a backup file is in process
- bit 6 – it indicates that a backup file should be opened when an open of a printer file has failed
- bit 7 – it indicates that this FIB is for a backup file

FIB.NB.SECT.PER.BLK

FIB.NB.SECT.PER.BLK

It is set by PHYSICAL.IO and used by LOGICAL.IO to compute area range.

FIB.BUF.N.FILE.STATUS

FIB.BUF.N.FILE.STATUS

This is the group name of fields containing information concerning the management of the file buffers and the current status of the file. Displacements, in digits, relative to the start of the structure (0-relative) have been stated allowing an easier access to one field.

Data type description:

01 FIB.BUF.N.FILE.STATUS		
02 FIB.FPB.INDEX		BIT(8)
02 FIB.FILE.STAT		BIT(4)
03 FIB.FILE.OPENING		BIT(1)
04 FIB.NEW.FILE		BIT(1)
03 FIB.FILE.OPEN		BIT(1)
03 FIB.CLOSING.FILE		BIT(1)
03 FIB.SHORT.BLK		BIT(1)
02 FIB.LAST.COMM		BIT(4) DIS 3
02 FIB.STOP.FOR.READ		BIT(1)
02 FIB.INH.READ.AGAIN		BIT(1)
02 FIB.ERROR		BIT(1)
02 FIB.LAST.BLK.DET		BIT(1)
02 FIB.LOG.EOF		BIT(1)
02 FIB.ACT		BIT(1)
02 FIB.EOF		BIT(1)
02 FIB.BWD.REQ		BIT(1)
03 FIB.OPEN.EXTEND.ALLOWED		BIT(1)
02 FIB.ERROR.REPLY.WORD		BIT(24)
03 FIB.AD.DCB		BIT(24)
02 FIB.SAVE.RETURN.ADDR		BIT(24)
02 FIB.REC.PER.BLK		BIT(16)
02 FIB.TOT.REC.COUNTER		BIT(20)

02 FIB.NB.REC.IN.LAST.BLK	BIT(16)
02 FIB.LAST.BLK.NB	BIT(24)
02 FIB.PH.BLK.NB	BIT(24)
03 RET.ADDR.IN.I.S	BIT(24)
02 FIB.OUT.PH.BLK.NB	BIT(24)
03 CURRENT.LOG.BUFFER	BIT(24)
02 FIB.BUF.INDEXES	BIT(24)
03 OUT.PH.BUF.INDEX	BIT(8)
03 LOG.BUF.INDEX	BIT(8) DIS 51
03 PH.BUF.INDEX	BIT(8)
02 FIB.LOG.REC.NB	BIT(24) DIS 55
03 FIB.DATA.FILE.PTR	BIT(24)
02 FIB.LOG.BLK.NB	BIT(24) DIS 61
02 FIB.HIGHEST.KEY	BIT(24)
02 FIB.BUF.STAT.AREA (17)	BIT(24)
03 BUF.STAT	BIT(8)
03 FIB.PRINT.CNTL	BIT(16)
04 FIB.SAVE.BUF.INDEX	BIT(8)

The following fields aid in determining the last communicate and the current processed record, if existing.

FIB.LAST.COMM It indicates the last communicate processed for the file. Possible values and their meaning are given below.

@1@ - open
 @2@ - read
 @3@ - write
 @4@ - delete
 @5@ - rewrite
 @6@ - overwrite
 @7@ - start
 @9@ - test status
 @A@ - read next
 @B@ - unsuccessful read next
 @F@ - unsuccessful start

This field is initialized to zero.

FIB.LOG.EOF It indicates, when set, that the logical end-of-file has been detected. It is set after the last record of the file has been read.

LOG.BUF.INDEX It indicates the relative buffer number (0-relative) used for the current communicate to LOGICAL.IO. It is initialized to zero.

FIB.LOG.REC.NB It contains the relative record number (1-relative) of the record involved in the

	current communicate. This field is not significant for the data file of an I/S file. It is initialized to zero.
FIB.LOG.BLK.NB	It contains the relative block number (0-relative) of the block contained in the buffer pointed to by LOG.BUF.INDEX. This field is not significant for the key file of an I/S file. It is initialized to @FFFF@. The relationship between the block number and the record number is : $\text{FIB.LOG.BLK.NB} = (\text{FIB.LOG.REC.NB} - 1) / (\text{BUFFER SIZE} / \text{RECORD SIZE})$
FIB.TO.CONSIDER	FIB.TO.CONSIDER This is the group name for fields pertaining to an indexed file FIB. Data type description: 01 FIB.TO.CONSIDER 02 WORK.WITH.EXTENDED.FIB BIT(1) 02 FIB.COMM.FOR.LOG.IO BIT(1) 02 FIB.COMM.FOR.LOG.IO.TERM BIT(1) 02 FIB.I.S.FILE BIT(1) 02 EXTENDED.FIB.ADD BIT(24) 03 FIB.PCW BIT(16)
FIB.COR.PTR	FIB.CALLING This area is used when transferring control back to LOGICAL.IO to identify the calling module and displacement into that module desired by LOGICAL.IO. Data type description: 01 FIB.CALLING BIT(24) 02 FIB.CALLING.MOD.NB BIT(8) 02 FIB.CALLING.DISPL BIT(16)
FIB.SUPERLOGIO/ INDEXED STATUS	FIB.SUPERLOGIO.FLAGS This first bit is set when the LOGIO module works in SUPERLOGIO mode.
FIB.AREA.RANGE	FIB.AREA.RANGE It is set up by PHYSICAL.IO at area allocation time. It indicates the first and last sector allocated. Data type description: 01 FIB.AREA.RANGE BIT(48) 02 FIB.AREA.MIN BIT(24) 02 FIB.AREA.MAX BIT(24)
FIB.BACKUP.INFO	FIB.BACKUP.INFO This field is used as a save area for various backup information needed when a DIRECT TO BACKUP has been asked for, or if a backup file has been renamed.

Data type description:

01 FIB.BACKUP.INFO
02 FPB.PACK.ID.SAVE BIT(56)
02 FIB.FPB.FILE.ID.SAVED BIT(96)
02 FPB.FILE.ID.SAVED BIT(96)
02 FIB.FPB.FILE.TYPE.SAVED BIT(8)
02 FIB.FPB.DEVICE.KING.SAVED BIT(8)
02 FIB.BACKUP.NUMBER BIT(1)
02 FIB.NAME.SWAPPED BIT(1)
02 FIB.DIRECT.TO.BACKUP BIT(1)
02 FIB.DISK.FOR.BACKUP.CHECK BIT(1)
02 FILLER BIT(1)

FIB.TO.CONSIDER

FIB.TO.CONSIDER

This is the group name for fields pertaining to an indexed file FIB.

Data type description:

01 FIB.TO.CONSIDER
02 WORK.WITH.EXTENDED.FIB BIT(1)
02 FIB.COMM.FOR.LOG.IO BIT(1)
02 FIB.COMM.FOR.LOG.IO.TERM BIT(1)
02 FIB.I.S.FILE BIT(1)
02 EXTENDED.FIB.ADD BIT(24)
03 FIB.VIRT.FILE.AREA.LGTH BIT(16)
03 FIB.PCW BIT(16)

FIB.STREAM.INFO

STREAM.INFO

This is the group name for fields pertaining to STREAM I/O operations. Some fields of this area are also re-mapped to handle SHARED files as STREAM files cannot be opened SHARED.

Data type description:

01 STREAM.INFO
02 FIB.STR.PTR BIT(16)
03 PART.OF.FIB BIT(2)
03 FIB.TOSF.DST.ENTRY.PTR
04 FIB.TOSF.FILE.PTR BIT(5)
04 FIB.TOSF.MIX.PTR BIT(3)
02 FIB.STR.BUF.REM BIT(16)
03 REL.DISP.OF.FIRST.FIB BIT(16)
02 FIB.STR.NB.BYTES.IN.LAST.BLK BIT(16)

FIB.ADDR.OF.BUFFERS

FIB.ADDR.OF.BUFFERS

It contains the bit address of the first buffer relative to the FIB.

COMMON.BUF.STAT	COMMON.BUF.STAT This field is displayed only if LOGIO works in SUPERLOGIO mode. It remaps the first three entries of BUF.STAT.AREA defined in FIB.BUF.N.FILE.STATUS.
REAL.BLOCK.SIZE.ON.DISK	REAL.BLOCK.SIZE.ON.DISK This field is displayed only if LOGIO works in SUPERLOGIO mode. It contains the block size on disk in bytes.
NB.OF.BLOCK.PER.BUFFER	NB.OF.BLOCK.PER.BUFFER This field is displayed only if LOGIO works in SUPERLOGIO mode. It contains the number of blocks per buffer.
LOGICAL.BUFFER.SIZE	LOGICAL.BUFFER.SIZE This field is displayed only if LOGIO works in SUPERLOGIO mode. It contains the buffer size. It is equal to: REAL.BLOCK.SIZE.ON.DISK * NB.OF.BLOCK.PER.BUFFER
LOGICAL BLOCK INFO	LOGICAL.BLOCK.INFO These fields are displayed only if LOGIO works on SUPERSUPERLOGIO mode. LOGICAL BLOCK 2 and LOGICAL BLOCK 3 contain the top and tail block numbers of the blocks contained in buffers 1 and 2. A block number is expressed on 24 bits.

BUFFERS

The contents of the file's buffers and their status are displayed next.

EXTENSION FOR INDEXED FILES

If the file is a Key file, the Key File Parameter Block (KFPB) and the Data file FIB will be displayed. The KFPB contains information associating the Data file to the Key file.

KFPB.IMPL.LEVEL	FIB.KFPB.IMPL.LEVEL It contains the implementation level of the Data file associated with this Key file. It corresponds to FIB.FPB.IMPL.LEVEL.NB.
KFPB.PACK.ID	FIB.KFPB.PACK.ID It contains the name of the device containing the Data file, if applicable.
KFPB.FILE.ID	FIB.KFPB.FILE.ID It contains the name of the Data file associated with this Key file.

COT.SECT.RANGE	COT.SECT.RANGE It contains the number of sectors of the Overlay Region that can be represented by one entry in the COT.AREA.	
CRT.SECT.RANGE	CRT.SECT.RANGE It contains the number of sectors of the Rough Table that can be represented by one entry in the CRT.AREA.	
KFPB.FLAGS	FIB.KFPB.FLAGS This is the group name for some Key file related flags.	
	Data type description:	
	01 FIB.KFPB.FLAGS	CHAR(1)
	02 B80.R.T	BIT(1)
	02 B700.R.T	BIT(1)
	02 B1700.R.T	BIT(1)
	02 B 900.R.T	BIT(1)
	02 B 1000.R.T.	BIT(1)
	02 DATA.FILE.DUAL	BIT(1)
	02 OLD.DUPL.ALLOWED	BIT(1)
	02 DUPL.ALLOWED	BIT(1)
KFPB.RT.ADD	FIB.KFPB.RT.ADD It contains the address in relative records of the Rough Table region in the Key file (1-relative).	
KFPB.RT.LGTH	FIB.KFPB.RT.LGTH It contains the length of the Rough Table in records (180 bytes).	
KFPB.OV.ADD	FIB.KFPB.OV.ADD It contains the address in relative records of the beginning of the Overflow region in the Key file (1-relative).	
KFPB.OV.LGTH	FIB.KFPB.OV.LGTH It contains the length of the Overflow region in records (180 bytes).	
KFPB.IX.ADD	FIB.KFPB.IX.ADD It contains the address in relative records of the beginning of the Index region in the Key file (1-relative).	
KFPB.IX.LGTH	FIB.KFPB.IX.LGTH It contains the length of the Index region in records (180 bytes).	
KFPB.KEY.LGTH	FIB.KFPB.KEY.LGTH It contains the actual key length in bytes.	
KFPB.KEY.OFFSET	FIB.KFPB.KEY.OFFSET It contains the key offset into the record in bytes.	

KFPB.ZERO

FIB.KFPB.ZERO

It contains the value zero.

FIB.LAST.INPUT.COMM

FIB.LAST.INPUT.COMM

It contains the previous input communication for indexed files. START is considered as an input communicate.

FIB.IX.PARAM

FIB.IX.PARAM

This is the group name for the parameters associated with the buffer relating to the Index region of the Key file.

Data type description:

01 FIB.IX.PARAM

02 TOP.SECT.NB.IN.DISK BIT(24)

02 TAIL.SECT.NB.IN.DISK BIT(24)

02 IX.REC.SECT.ADD BIT(24)

02 IX.REC.IN.BLK.PTR BIT(8)

02 IX.FLAGS

03 IX.MEM.VALID BIT(1)

03 IX.KEY.FOUND BIT(1)

03 IX.EOF.REACHED BIT(1)

03 FILLER BIT(1)

FIB.OV.PARAM

FIB.OV.PARAM

This is the group name for the parameters associated with the buffer relating to the Overflow region of the Key file.

Data type description:

01 FIB.OV.PARAM

02 TOP.SECT.NB.IN.DISK.OV BIT(24)

02 TAIL.SECT.NB.IN.DISK.OV BIT(24)

02 OV.REC.SECT.ADD BIT(24)

02 OV.REC.SECT.ADD BIT(16)

02 OV.REC.IN.BLK.PTR BIT(8)

02 OV.FLAGS

03 OV.MEM.VALID BIT(1)

03 OV.KEY.FOUND BIT(1)

03 OV.EOF.REACHED BIT(1)

03 OV.DEL.FOUND BIT(1)

FIB.RGH.PARAM

FIB.RGH.PARAM

This is the group name for the parameters associated with the Rough table of the Key file of an indexed pair of files.

Data type description:

01 FIB.RGH.PARAM

02 TOP.SECT.NB.IN.DISK.RT BIT(24)

02 TAIL.SECT.NB.IN.DISK.RT BIT(24)

02 RGH.REC.SECT.ADD BIT(24)

	02 RGH.REC.SECT.ADD	BIT(16)
	02 RGH.REC.IN.BLK.PTR	BIT(8)
	02 RGH.FLAGS	
	03 RGH.MEM.VALID	BIT(1)
	03 RGH.KEY.FOUND	BIT(1)
	03 RGH.EOF.REACHED	BIT(1)
	03 FILLER	BIT(1)
FIB.SRCH.PARAM	FIB.SRCH.PARAM	
	This is the group name for the fields used when LOGICAL.IO requests a SEARCH by PHYSICAL.IO.	
	Data type description:	
	01 FIB.SRCH.PARAM	
	02 FIB.KEY.LGTH	BIT(8)
	02 FIB.REC.LGTH	BIT(16)
	02 FIB.KEY.REC.PER.SECT	BIT(8)
	02 FIB.REC.POSITION	BIT(16)
	02 FIB.SEARCH.REPLY.WORD	BIT(24)
	02 FIB.SEARCH.KEY	CHAR(32)
I.S.INFO	I.S.INFO	
	This is the group name for fields pertaining to indexed files.	
	Data type description:	
	01 I.S.INFO	
	02 I.S.HIGHEST.KEY	CHAR(29)
	02 COMM.RET.ADD	BIT(24)
	02 I.S.ACC.MODE	BIT(2)
	02 I.S.WRITE.PERFORMED	BIT(1)
	02 I.S.NULL.KEY.FILE	BIT(1)
	02 FIB.KFPB.ALREADY.UPDATED	BIT(1)
	02 FILLER	BIT(3)
KEY.RETAINED	KEY.RETAINED	
	It contains the last key which has been read.	
READ.POINTERS	READ.POINTERS	
	These are the fields which contain the location of the last key which has been read on disk.	
	Data type description:	
	01 READ.POINTERS	
	02 INDEX.POINTER	
	03 INDEX.READ.SECTOR	BIT(24)
	03 INDEX.READ.KEY.NB	BIT(8)
	02 OVERFLOW.POINTER	
	03 OVERFLOW.READ.SECTOR	BIT(24)
	03 OVERFLOW.READ.KEY.NB	BIT(8)
	02 READ.FLAGS	
	03 READ.OVERFLOW.BEFORE.NEXT	BIT(1)

03	INDEX.READ.FLAGS	
04	INDEX.KEY.FOUND	BIT(1)
04	INDEX.EOF.REACHED	BIT(1)
03	OVERFLOW.READ.FLAGS	
04	OVERFLOW.KEY.NOT.FOUND	BIT(1)
04	OVERFLOW.EOF.REACHED	BIT(1)
03	SKIP.OV.BEFORE.NEXT	BIT(1)
03	SKIP.IX.BEFORE.NEXT	BIT(1)

SAVE.PUSH.PTR

SAVE.PUSH.PTR

It contains the disk address (sector number plus offset) of the key to be pushed in the key file.

Data type description:

01	SAVE.PUSH.PTR	
02	SAVE.PUSH.SECTOR	BIT(24)
02	SAVE.PUSH.KEY.NB	BIT(8)

SAVE.PUSH.KEY

SAVE.PUSH.KEY

It contains the value of the key to be pushed in the key file.

INFORMATION FOR A CLOSED FILE

For a closed file, only the FPB contents are printed, as there is no FCB and no FIB. Refer to FILE ATTRIBUTES FROM FPB of Information For An Opened File for a complete description of the fields. The names of the members of the FIB structure have to be modified as follows: each prefix of those names, namely FIB.FPB, has to be replaced by FPB; for example, the name FIB.FPB.PACK.ID becomes FPB.PACK.ID; the resulting name is the label of a member of the structure FPB.

For indexed files, additional fields are analyzed.

DATA FILE
PACK-ID

FPB.DATA.FILE.P.ID
It contains the name of the device containing the data file.

DATA FILE ID

FPB.DATA.FILE.ID
It contains the name of the data file associated with this key file.

ROUGH TABLE
SIZE IN CORE

FPB.RGH.TABLE.SIZE
It contains the length of the rough table in records (= 180 bytes).

LENGTH OF
KEY IN BYTES

FPB.KEY.LGTH
It contains the key length in bytes.

OFFSET OF
KEY

FPB.KEY.OFFSET
It contains the key offset into the record in bytes.

Following this data, one of the following messages will appear:

‘ FILE HAS NOT BEEN OPENED’
“FILE HAS BEEN CLOSED”
“FILE IS IN PROCESS OF OPEN”
“FILE HAS BEEN HALF CLOSED”

This message describes the file’s status at the time of the abnormal terminate.

DATA STACK ANALYSIS/CURRENT OPERAND (COP) TABLE

DATA STACK ANALYSIS (MPL)

DATA STACK STRUCTURE

The ‘DATA STACK ANALYSIS’ section of an MPL DP.ANALYZER listing provides information about a user program’s active data. The value of all fields active at the time of the abnormal terminate is given along with descriptive information for each of these fields.

The data is presented according to its entry in the CONTROL STACK. All global data is presented first. Data local to the procedure next represented on the CONTROL STACK is presented next (if there is an entry) and so on.

The last group of data listed in lexical level order is that which is local to the procedure active at the time of the abnormal terminate. There is no CONTROL STACK entry for this currently active procedure.

The last active procedures group of data may be followed by a group of data related to a Working Stack. During the execution of a statement, one or more descriptors are temporarily loaded onto a Working Stack. If the contents of this Working Stack are accessible, the descriptors loaded on it will be printed. If it can be determined that the Working Stack is empty, the following message is printed:

“(EMPTY WORKING STACK)”

HEADING DESCRIPTION

The groups of data are separated by a box of asterisks containing information identifying the procedure, and, when pertinent, one of the following messages:

(GLOBAL DECLARATIONS)	indicating that the subsequent data is global to the program.
(PRESENTLY OUT OF SCOPE)	indicating that the subsequent data is local to a procedure on the Control Stack but not to the currently active procedure.
(ACTIVE PROCEDURE)	indicating that the subsequent data is local to the procedure being executed at the time of the abnormal terminate.

If there is no data declared in a procedure which is active or on the Control Stack, the message:

“NO DECLARATION IN THIS PROCEDURE”

is given.

Various messages not described in this section are self-explanatory, and are printed in the case of errors detected by DP.ANALYZER, for example:

“** ERROR -UNEXPECTED END OF DATA SEGMENT 0 WHILE LOOKING FOR A DESCRIPTOR **
** DATA STACK ANALYSIS ABORTED **”

The number of parameters passed to a procedure may validly be less than the number of parameters expected by the procedure. The interpreter builds invalid descriptors in place of the missing parameters. In the case of missing parameters, the number of invalid descriptors built by the interpreter will be printed. The message given is:

“NUMBER OF INVALID DESCRIPTORS: <integer>”

The following fields appear in the box of asterisks:

LL	This is the procedure lexical level corresponding to the first column on the left-hand side of a compiled source listing.
PSN	This is the code segment number (Program Segment Number) of the segment containing the procedure concerned.
SPN	This is the Segment Procedure Number. That is, the number representing the position of the procedure within the code segment, relative to its other procedures.
STACK BASE	This is the byte offset into the Data Segment where the descriptors for the procedure represented are found. The descriptors for a procedure are always contiguous. The offset is given in hexadecimal and then in decimal.

DATA DESCRIPTION

INDEX NR	It allows the identification of the data item. The compiler associates each declared data item to an index number (it appears on the compiler source listing in the third column from the left); this number is relative to the procedure, that is, the first declared data item of every procedure has an index number of zero.
DESCRIPTOR	<p>A 4-byte descriptor exists for each data item and is used to locate and describe the data associated with that item. The descriptor contains information as to the type of the data and where it is located, that is, either in a data segment at an offset or, in the case of a self-relative data item, in the descriptor itself.</p> <p>Each descriptor is checked to see if it refers to valid data. When the descriptor is found to point outside the segment boundary, the following warning is printed:</p> <p>“** ERROR – DESCRIPTOR POINTS OUTSIDE SEGMENT **”</p> <p>If a descriptor is found to be invalid, an appropriate message is printed.</p>

	The Data Stack is often destroyed by invalid use of indices.
DATA ADDRESS - SEGM/DISPL	These two columns give the data segment and offset into that segment in bytes at which the data is located. This information is taken from the data descriptor. If the item is self-relative, that is, if the data is located in the descriptor itself, the words "SELF RELATIVE" will appear.
BIT OFFSET	If the data item describes a field of type BIT, this column gives the bit position in the byte given by "DATA ADDRESS - SEGM/DISPL" at which the bit string begins .
DATA TYPE/LGTH	This gives the type and length of the data represented by this entry. 'TYPE' is one of the following: BIT a bit string of maximum length 8 bits. FIXED a 16-bit numeric field CHAR a character string up to 255 bytes in length. MSG. REF a four byte description of a data communication message. 'LGTH' represents the length in bits for type BIT fields, in bytes for type CHAR and is not given for type FIXED as this length is always 16 bits.

NOTE

The field containing the length of a bit string is four bits long, therefore a length of up to 15 bits can be represented. BILINTERP will sometimes use bit strings of greater than eight bits in length. The MPL language, however only provides for a maximum of eight bits.

VALUE	The contents of the data are displayed. This represents the value of the data at the time of the abnormal terminate. The data is displayed in hexadecimal. For fields of type CHAR, the value is displayed in alphanumeric (ASCII) format as well. Invalid ASCII representations appear as the period character.
-------	--

CURRENT OPERAND (COP) TABLE (COBOL and RPG)

The COP table consists of a set of entries, each being a descriptor containing the attributes of a data item. All data declared in a user's program is presented in this section of a DP.ANALYZER listing.

COP INDEX	It allows the identification of the data item; it is assigned by the COBSVERTER pass of the RPG and COBOL compilers.
COBOL related	To find the value of a data item COP INDEX 1. Find the declaration in the source listing to obtain the "old" COP index found at the right of the declaration in brackets.

2. Use this index to find the “new” COBSVERTER index in the COP-TABLE transformation map generated with either the “**\$COP-TABLE**” or “**\$OPTCODE**” options.

3. Use the “new” index found in (2) to identify the desired item in the DP.ANALYZER listing by searching the column headed “COP INDEX”.

NOTE

Some COP-TABLE entries are removed by COBSVERTER and replaced by in-line code, therefore, they will not be seen in the COP-TABLE. One easy way to ensure a field’s appearance is to use the desired field more than once in the program.

RPG related	To find the value of a data item COP INDEX 1. Find the field name in the “FIELDS” table, generated by the “ \$NAMES ” option, to obtain the “old” COP index found in the column entitled “COP NUMBER” of this table. 2. Use this “old” index to find the “new” COBSVERTER index found in the COP-TABLE transformation map generated with the “ \$XMAP ” option. This new index is found in the column entitled “C.O.P.”. 3. Use the “new” index found in (2) to identify the desired item in the DP.ANALYZER listing by searching the column headed “COP INDEX”.
ADDRESS - SEG/DISP.	These two columns give the data segment and offset into that segment in digits (that is, 4-bit units) at which the data is located. With this information the data can be located in the appropriate data segment.
DATA TYPE	This gives the type of the data represented by this entry. The possible types are as follows: 4U – 4-bit packed, unsigned alphanumeric 8U – 8-bit unpacked, unsigned alphanumeric 4LS – 4-bit packed, sign on left, numeric 4TS – 4-bit packed, trailing sign (on right), numeric 8SLS – 8-bit unpacked, separate sign on left, numeric 8OLS – 8-bit unpacked, overpunched sign on left, numeric 8STS – 8-bit unpacked, separate trailing sign (on right), numeric 8OTS – 8-bit unpacked, overpunched trailing sign (on right), numeric
UNIT LENGTH	This gives the length of the data represented by this entry. When the type of the field is “4U”, “4LS” or “4TS” (that is, packed) the length is given in digits (that is, 4-bit units). Otherwise the length is given in bytes.

In the case of an array declaration, the array element length is given. Up to 20 elements of the array will be displayed consecutively. Additional elements of the array will be displayed only in the case of multi-dimensional arrays. Enough elements will be displayed to allow the location of any other element in the corresponding data segment to be determined.

For example, in the case of a 3-dimensional array, elements (1,1,1) through (1,1,20) will be displayed. The elements (1,2,1) and (2,1,1) will also appear. These can assist in determining the location of the other elements of the array. In this case, therefore, 22 elements of the array will be displayed.

For array elements the words "ELEM (n)" will appear where "n" is replaced by the index of the element listed.

NOTE

If the word "ARRAY" is given in the initiating message passed to DP.ANALYZER, all elements of all arrays will be printed.

VALUE

The contents of the data are displayed. This represents the value of the data at the time of the abnormal terminate. The data is displayed in hexadecimal for all unpacked fields, followed by the ASCII representation. In the case of packed fields, the value of the data is displayed in digits with the sign if appropriate.

When a descriptor refers to a signed numeric value, the message:

"UNEXPECTED SIGN"

is printed if the sign is not valid.

In the case of an array declaration, information about the array is given here as its value will be listed element by element. The array information includes the size of the array (that is, number of elements), and whether it is a subscripted or indexed array. In the case of a subscripted array, the word "SUBSCRIPTED" will appear. In the case of an array referred to with indexes, the number of indexes declared is given here. The element contents are then displayed; only the first 180 bytes are printed. If only a portion of the array is printed, the remaining portion can be seen in the appropriate data segment.

DATA SEGMENTS

The contents of the user program's data segments are listed in numerical order, using hexadecimal and ASCII representations. Each segment's contents are separated by a line identifying the segment and giving its size in bytes.

To the left of a data segment's contents are numbers identifying the displacement of the first digit or byte in the corresponding line. In the case of RPG and COBOL, these numbers identify digit (4-

bit unit) displacements. In the case of MPL, these numbers represent byte displacements. In all cases, the numbers are 0-relative.

When lines of text are repetitive, the message 'SAME AS LINE ABOVE' appears in the data segment. The position number to the left helps to determine how many lines are suppressed.

If a pair of digits does not represent a valid ASCII character, a period appears in the display representation of that byte position.

Some messages identifying the contents of a data segment are given, where appropriate. These messages are as follows:

```
“INITIATING MESSAGE”
“FPB FOR FILE: <file-id>”
“FIB FOR FILE: <file-id>”
```

If the option “NO.INIT” is used in an MPL segment declaration and that segment was never accessed by the interpreter, the message:

```
“NO VALUE ATTRIBUTED TO SEGMENT”
```

is printed.

The structures of the FPB and the FIB are given below; the description of significant fields may be found in the section FILE INFORMATION. Displacements, in bytes, relative to the start of the structure have been stated allowing an easier access in the adequate data segment.

01	FPB		
02	FPB.IMPL.LEVEL.NB	BIT(8)	DISPL 0
02	FPB.PACK.ID	BIT(56)	DISPL 1
02	FPB.FILE.ID	BIT(96)	DISPL 8
02	BLANK	BIT(8)	
02	FPB.REEL.NB	BIT(24)	DISPL 21
02	FPB.FILE.TYPE	BIT(8)	DISPL 24
02	FPB.HIGHEST.RECD.NB	BIT(24)	DISPL 25
02	FPB.DEVICE.KIND	BIT(8)	DISPL 28
02	FPB.WA.SEGM.NB	BIT(8)	DISPL 29
02	FPB.WA.OFFSET	BIT(16)	DISPL 30
02	FPB.REC.SIZE	BIT(16)	DISPL 32
02	FPB.BUF.SIZE	BIT(16)	DISPL 34
02	FPB.MAX.FILE.SIZE	BIT(24)	DISPL 36
02	FPB.NB.BUF	BIT(8)	DISPL 39
02	FPB.FLAGS	BIT(8)	DISPL 40
	03 FPB.SPEC.FORM	BIT(1)	
	04 FPB.DUP.ALLOWED	BIT(1)	
	03 FPB.OPT.FILE	BIT(1)	
	03 FPB.NO.LABEL	BIT(1)	
	03 FPB.COND.FILE	BIT(1)	
	03 FPB.PROG.FILE	BIT(1)	
	03 FPB.FILLER.A	BIT(3)	
02	FPB.ADV.CLOSE	BIT(8)	DISPL 41

	03	FILLER	BIT(1)		
	03	FPB.NO.REWIND	BIT(1)		
	04	FPB.CHANGE.REEL	BIT(1)		
	03	FPB.CLOSE.MODE	BIT(3)		
	03	FPB.CLOSE.CRUNCH	BIT(1)		
	03	FPB.IX.OV.MRG	BIT(1)		
	03	FILLER	BIT(1)		
02		FPB.ADV.OPEN	BIT(16)	DISPL	42
	03	FPB.FILLER.B	BIT(1)		
	03	FPB.EXTEND	BIT(1)		
	03	FPB.OTHERUSE	BIT(3)		
	03	FPB.FILLER.C	BIT(1)		
	03	FPB.MYUSE	BIT(2)		
	03	FPB.BACKUP.CNTL	BIT(2)		
	03	FPB.FILLER.D	BIT(2)		
	03	FPB.ACCESS.MODE	BIT(2)		
	03	FPB.FILLER.E	BIT(2)		
02		FPB.CYCLE	BIT(16)	DISPL	44
02		FPB.GEN.NB	BIT(16)	DISPL	46
02		FPB.CREAT.DATE	BIT(40)	DISPL	48
02		FPB.LAST.ACCESS.DATE	BIT(40)	DISPL	53
02		FPB.SPARE.CHAR	BIT(16)	DISPL	58
02		FPB.SAVE.FACT	BIT(24)	DISPL	60
02		FPB.DATA.FILE.P.ID	BIT(56)	DISPL	63
02		FPB.DATA.FILE.ID	BIT(96)	DISPL	70
02		FPB.BLANK	BIT(8)	DISPL	82
02		FPB.SPARE	BIT(8)	DISPL	83
02		FPB.RGH.TABLE.SIZE	BIT(16)	DISPL	84
02		FPB.ZERO1	BIT(8)	DISPL	86
02		FPB.KEY.LGTH	BIT(8)	DISPL	87
02		FPB.KEY.OFFSET	BIT(16)	DISPL	88
02		FPB.ZERO2	BIT(32)	DISPL	90
01		FIB			
	02	FIB.FCB	BIT(728)		
	03	FCB.CODE.PTR	BIT(24)	DISPL	0
	03	FCB.CUR.BUF.ADDR	BIT(24)	DISPL	3
	03	FCB.CUR.BUF.LENGTH	BIT(24)	DISPL	6
	03	FCB.OP.CODE	BIT(24)	DISPL	9
	03	FCB.DCB.ADDRESS	BIT(24)	DISPL	12
	03	FCB.FILE.STATUS	BIT(24)	DISPL	15
	04	FCB.USE.SYSMEM	BIT(1)		
	04	FCB.CUR.USE.SYSMEM	BIT(1)		
	04	FCB.USE.AVAIL.TABLE	BIT(1)		
	04	FCB.CUR.USE.AVAIL.TABLE	BIT(1)		
	04	FCB.NEW.OLD	BIT(1)		
	04	FCB.OPENING	BIT(1)		
	04	FCB.CLOSE	BIT(1)		
	04	FCB.USE.FPB.VN	BIT(1)		

	05	FCB.TRANSLATE.COMPLETE	BIT(1)		
	04	FCB.SEARCH	BIT(1)		
	04	FCB.ERRB	BIT(1)		
	04	FCB.QUEUED	BIT(1)		
	04	FCB.SUSPENDING.TASK	BIT(1)		
	04	FCB.ACTIVE	BIT(1)		
	04	FCB.IN.USE	BIT(1)		
	04	FCB.DISPLAY	BIT(1)		
	04	FCB.READING.LABEL	BIT(1)		
	04	FCB.HALF.CLOSED	BIT(1)		
	04	FILLER	BIT(7)		
03		FCB.DEVICE.KIND	BIT(8)	DISPL	18
03		FCB.RETRY.COUNT	BIT(4)	DISPL	19
03		FCB.CHANNEL	BIT(4)		
03		FCB.FILE.NR	BIT(8)	DISPL	20
03		FCB.FIB.ADDRESS	BIT(24)	DISPL	21
03		FCB.MIX	BIT(8)	DISPL	24
03		FCB.Q.LINK	BIT(24)	DISPL	25
03		FCB.COMM			
	04	FCB.RD	BIT(24)	DISPL	28
	04	FCB.SAVE.SCRATCHPADS	BIT(64)	DISPL	31
	05	SHORT.FCB.LENGTH	BIT(64)		
	04	FCB.RETURN.LIST	BIT(96)	DISPL	39
	04	FCB.RETURN	BIT(8)	DISPL	51
	04	FCB.MYUSE	BIT(2)	DISPL	52
	04	FCB.OTHERUSE	BIT(3)		
	04	FILLER	BIT(3)		
	04	FCB.MEMORIZE.DCB	BIT(24)	DISPL	53
	04	FCB.DISK.ADDRESS	BIT(24)	DISPL	56
	04	FCB.CUR.AREA.AD	BIT(24)	DISPL	59
	04	FCB.DFH..ADDRESS	BIT(24)	DISPL	62
	04	FCB.DIRENTRY.ADDRESS	BIT(24)	DISPL	65
	04	FCB.AREA.NR	BIT(4)	DISPL	68
	04	FCB.AREA.COUNT	BIT(4)		
	04	FCB.DIRENTRY.IX	BIT(8)	DISPL	69
	04	FCB.SECTION.NR	BIT(24)	DISPL	70
	04	FCB.RETRIEVED.ITEM	BIT(24)	DISPL	71
	05	FCB.DUAL.FILE	BIT(24)		
	04	FCB.SEARCH.STATUS	BIT(8)	DISPL	76
	04	FCB.SEARCH.OP.CODE	BIT(8)	DISPL	77
	04	FCB.SEARCH.SKIP	BIT(8)	DISPL	78
	04	FCB.ARGUMENT.ADDRESS	BIT(24)	DISPL	79
	04	FCB.SPACE.TO.ALLOCATE	BIT(24)	DISPL	82
	04	FCB.CUR.AREA.RANGE	BIT(48)	DISPL	85
02		FIB.AREA.ADDR.LGTH. (16)	BIT(32)	DISPL	91
02		FIB.COMM.AREA			
	03	FIB.COMM.VERB	BIT(8)	DISPL	155
	03	FIB.COMM.BLK.NB	BIT(24)	DISPL	156
	04	FIB.SEARCH.ADD	BIT(24)		

03	FIB.COMM.MEM.BLK.ADD		
04	FIB.OPEN.CLOSE.BUF.AD		
05	FIB.BUF.AD		
06	FIB.KEY.MEM.ADD	BIT(24)	DISPL 159
03	FIB.FILE.NB	BIT(8)	DISPL 162
03	FIB.REPLY.WORD	BIT(24)	DISPL 163
03	FIB.CUR.BLK.LGTH	BIT(24)	DISPL 166
04	FIB.SEARCH.LGTH	BIT(24)	
02	WORKING.COPY.OF.FPB		
03	FIB.FPB.IMPL.LEVEL.NB	CHAR(1)	DISPL 169
03	FIB.FPB.PACK.ID	CHAR(7)	DISPL 170
03	FIB.FPB.FILE.ID	CHAR(12)	DISPL 177
03	FIB.FPB.PSEUDO.PACK.TAG	BIT(8)	DISPL 189
03	FIB.FPB.FILE.ID.HASH	BIT(8)	DISPL 190
03	FIB.FPB.FILE.NB	BIT(8)	DISPL 191
03	FIB.FPB.REEL.NB	CHAR(3)	DISPL 192
03	FIB.FPB.FILE.TYPE	BIT(8)	DISPL 195
03	FIB.FPB.HIGHEST.RECD.NB	BIT(24)	DISPL 196
03	FIB.FPB.DEVICE.KIND	BIT(8)	DISPL 199
03	FIB.FPB.WA.SEGM.NB	BIT(8)	DISPL 200
03	FIB.FPB.WA.OFFSET	BIT(16)	DISPL 201
03	FIB.FPB.REC.SIZE	BIT(16)	DISPL 203
03	FIB.FPB.BUF.SIZE	BIT(16)	DISPL 205
03	FIB.FPB.MAX.FILE.SIZE	BIT(24)	DISPL 207
03	FIB.FPB.NB.BUF	BIT(8)	DISPL 210
03	FIB.FPB.FLAGS	BIT(8)	DISPL 211
04	FIB.FPB.SPEC.FORM	BIT(1)	
05	FIB.FPB.DUP.ALLOWED	BIT(1)	
04	FIB.FPB.UPT.FILE	BIT(1)	
04	FIB.FPB.NO.LABEL	BIT(1)	
04	FIB.FPB.COND.FILE	BIT(1)	
04	FILLER	BIT(3)	
03	FIB.FPB.ADV.CLOSE	BIT(8)	DISPL 212
04	FILLER	BIT(1)	
04	FIB.FPB.NO.REWIND	BIT(1)	
05	FIB.FPB.CHANGE.REEL	BIT(1)	
04	FIB.FPB.CLOSE.MODE	BIT(3)	
04	FIB.FPB.CLOSE.CRUNCH	BIT(1)	
04	FIB.FPB.IX.OV.MRG	BIT(1)	
03	FIB.FPB.ADV.OPEN	BIT(16)	DISPL 213
04	FIB.FPB.FILLER.B	BIT(1)	
04	FIB.FPB.EXTEND	BIT(1)	
04	FIB.FPB.OTHERUSE	BIT(3)	
04	FIB.FPB.CREATENAMED	BIT(1)	
04	FIB.FPB.MYUSE	BIT(2)	
04	FIB.FPB.BACKUP.CNTL	BIT(2)	
04	FIB.FPB.INTEG.LEVEL	BIT(2)	
04	FIB.FPB.ACCESS.MODE	BIT(2)	
04	FIB.FPB.FILLER.E	BIT(2)	

03	FIB.FPB.CYCLE	CHAR(2)	DISPL	215
03	FIB.FPB.GEN.NB	BIT(16)	DISPL	217
03	FIB.FPB.CREAT.DATE	CHAR(5)	DISPL	219
03	FIB.FPB.LAST.ACCESS.DATE	CHAR(5)	DISPL	224
03	FIB.FPB.SPARE.CHAR	BIT(16)	DISPL	229
03	FIB.FPB.SAVE.FACT	CHAR(3)	DISPL	231
02	FIB.FPB.USERCODE	BIT(136)	DISPL	234
02	FIB.FPB.SECURITY.FLAGS	BIT(8)	DISPL	251
03	FIB.FPB.SECURITYTYPE	BIT(2)		
03	FIB.FPB.SECURITYUSE	BIT(2)		
03	FILLER	BIT(4)		
02	FIB.FPB.GUARD.MFID	BIT(56)	DISPL	252
02	FIB.FPB.GUARD.FID	BIT(96)	DISPL	259
03	FIB.FPB.VM.LGTH.FORWARD	BIT(24)		
02	FIB.FILE.STATUS		DISPL	271
03	FIB.FILE.HALF.CLOSE	BIT(1)		
03	FIB.GUARD.FILE.OPENED	BIT(1)		
03	FIB.HOST	BIT(1)		
03	FIB.EXEC	BIT(1)		
03	FIB.OWN	BIT(1)		
03	FIB.PROG	BIT(1)		
03	FILLER	BIT(10)		
02	FIB.VN	BIT(8)	DISPL	271
03	FIB.ALREADY.DISPLAYED	BIT(1)		
03	FIB.FLAGS.USED.BY.PHYSICAL	BIT(2)		
03	FIB.USED.TO.DISPLAY	BIT(1)		
03	FIB.FILE.TESTED	BIT(1)		
03	FIB.OPENING.BACKUP	BIT(1)		
03	FIB.PRINTER.CHECK	BIT(1)		
03	FIB.FOR.BACKUP.FILE	BIT(1)		
02	FIB.NB.SECT.PER.BLK	BIT(24)	DISPL	274
02	FIB.BUF.N.FILE.STATUS			
03	FIB.FPB.INDEX	BIT(8)	DISPL	277
03	FIB.FILE.STAT	BIT(4)	DISPL	278
04	FIB.FILE.OPENING	BIT(1)		
05	FIB.NEW.FILE	BIT(1)		
04	FIB.FILE.OPEN	BIT(1)		
04	FIB.CLOSING.FILE	BIT(1)		
04	FIB.SHORT.BLK	BIT(1)		
03	FIB.LAST.COMM	BIT(4)		
03	FIB.STOP.FOR.READ	BIT(1)	DISPL	279
03	FIB.INH.READ.AGAIN	BIT(1)		
03	FIB.ERROR.	BIT(1)		
03	FIB.LAST.BLK.DET	BIT(1)		
03	FIB.LOG.EOF	BIT(1)		
03	FIB.ACT	BIT(1)		
03	FIB.EOF	BIT(1)		
03	FIB.BWD.REQ	BIT(1)		
04	FIB.OPEN.EXTEND.ALLOWED	BIT(1)		

03	FIB.ERROR.REPLY.WORD	BIT(24)	DISPL 280
	04 FIB.AD.DCB	BIT(24)	
03	FIB.SAVE.RETURN.ADDR	BIT(24)	DISPL 283
03	FIB.REC.PER.BLK	BIT(16)	DISPL 286
03	FIB.NB.REC.IN.LAST.BLK	BIT(16)	
03	FIB.LAST.BLK.NB	BIT(24)	
03	FIB.PH.BLK.NB	BIT(24)	
03	FIB.OUT.PH.BLK.NB	BIT(24)	
03	FIB.BUF.INDEXES	BIT(24)	
	04 OUT.PH.BUF.INDEX	BIT(8)	
	04 LOG.BUF.INDEX	BIT(8)	
	04 PH.BUF.INDEX	BIT(8)	
03	FIB.LOG.REC.NB	BIT(24)	
	04 FIB.DATA.FILE.PTR	BIT(24)	
03	FIB.LOG.BLK.NB	BIT(24)	
03	FIB.HIGHEST.KEY	BIT(24)	
03	FIB.BUF.STAT.AREA. (17)	BIT(24)	DISPL 311
	04 BUF.STAT	BIT(8)	
	04 FIB.PRINT.CNTL	BIT(16)	
03	SUPERLOGIO.BUFFER.STATUS.REMAPS FIB.BUF.STAT.AREA		
	04 COMMON.BUF.STAT	BIT(72)	
	04 REAL.BLOCK.SIZE.ON.DISK	BIT(16)	
	04 NB.OF.BLOCK.PER.BUFFER	BIT(8)	
	04 LOGICAL.BUFFER.SIZE	BIT(16)	
	04 LOGICAL.BLOCK.INFO (3)		
	05 TOP.BLOCK.NB	BIT(24)	
	05 TAIL.BLOCK.NB	BIT(24)	
02	FIB.CALLING	BIT(24)	DISPL 362
	03 FIB.CALLING.MOD.NB	BIT(8)	
	03 FIB.CALLING.DISPL	BIT(16)	
02	FIB.COMM.FOR.SUPERLOGIO	BIT(1)	DISPL 365
02	FIB.SUPERLOGIO.FLAGS	BIT	
	03 FIB.INTERPRETER.RESTARTED	BIT(1)	
	04 FIB.RESTART.COMM	BIT(1)	
	04 FIB.AREA.RANGE.UPDATED	BIT(1)	
	03 FIB.END.OF.AREA.REACHED	BIT(1)	
	03 FIB.REWRITE.OPERATION	BIT(1)	
02	FIB.SEARCH.IX	BIT(1)	
02	FIB.CLOSE.MERGE.FAILURE	BIT(1)	
02	FILLER	BIT(1)	
02	FIB.AREA.RANGE	BIT(48)	DISPL 366
	03 FIB.AREA.MIN	BIT(24)	
	03 FIB.AREA.MAX	BIT(24)	
02	FIB.BACKUP.INFO		
	03 FPB.PACK.ID.SAVE	BIT(56)	DISPL 372
	03 FIB.FPB.FILE.ID.SAVED	BIT(96)	DISPL 379
	03 FPB.FILE.TO.SAVED	BIT(96)	DISPL 391
	03 FIB.FPB.FILE.TYPE.SAVED	BIT(8)	DISPL 403
	03 FIB.FPB.DEVICE.KIND.SAVED	BIT(8)	DISPL 404
	03 FIB.BACKUP.NUMBER	BIT(8)	DISPL 405

	03	FIB.NAME.SWAPPED	BIT(1)	
	03	FIB.DIRECT.TO.BACKUP	BIT(1)	
	03	FIB.DISK.FOR.BACKUP.CHECK	BIT(1)	
	03	FILLER	BIT(1)	
02		FIB.TO.CONSIDER		
	03	WORK.WITH.EXTENDED.FIB	BIT(1)	
	03	FIB.COMM.FOR.LOG.IO	BIT(1)	
	03	FIB.COMM.FOR.LOG.IO.TERM	BIT(1)	
	03	FIB.I.S.FILE	BIT(1)	
	03	EXTENDED.FIB.ADD	BIT(24)	DISPL 407
	04	FIB.VIRT.FILE.AREA.LGTH	BIT(16)	
	05	FIB.PCW	BIT(16)	
02		STREAM.INFO		
	03	FIB.STR.PTR	BIT(16)	DISPL 410
	04	PART.OF.FIB	BIT(2)	
	04	FIB.TOSF.DST.ENTRY.PTR		
	05	FIB.TOSF.FILE.PTR	BIT(5)	
	05	FIB.TOSF.MIX.PTR	BIT(3)	
	03	FIB.STR.BUF.REM	BIT(16)	DISPL 412
	04	REL.DISP.OF.FIRST.FIB	BIT(16)	
	03	FIB.STR.NB.BYTES.IN.LAST.BLK	BIT(16)	DISPL 414
02		FIB.ADDR.OF.BUFFERS	BIT(24)	DISPL 416
02		BUFFER.SPACE		
	03	IX.BUFFER	CHAR(1800)	DISPL 419
	04	IX.AREA	CHAR(180)	
	03	OV.BUFFER	CHAR(1800)	
	04	OV.AREA	CHAR(180)	
	03	RGH.BUFFER	CHAR(1800)	DISPL 4019
	04	RGH.AREA	CHAR(180)	
	03	CRT.AREA	CHAR(180)	
02		KFPB.AREA		
	03	FIB.KFPB.IMPL.LEVEL	CHAR(1)	DISPL 5999
	03	FILLER	CHAR(2)	
	03	FIB.KFPB.PACK.ID	CHAR(7)	DISPL 6002
	03	FIB.KFPB.FILE.ID	CHAR(12)	DISPL 6009
	03	FIB.KFPB.BLANK	CHAR(1)	
	03	FIB.KFPB.LINK.TO.DATA.FILE	CHAR(5)	DISPL 6022
	04	COT.SECT.RANGE	BIT(8)	
	04	CRT.SECT.RANGE	BIT(8)	
	03	FIB.KFPB.FLAGS	CHAR(1)	DISPL 6027
	04	B 80 R.T.	BIT(1)	
	04	B 700 R.T.	BIT(1)	
	04	B 1700 R.T.	BIT(1)	
	04	B 900 R.T.	BIT(1)	
	04	B 1000 R.T.	BIT(1)	
	04	DATA.FILE.DUAL	BIT(1)	
	04	OLD.DUPL.ALLOWD	BIT(1)	
	04	DUPL.ALLOWED	BIT(1)	
	03	FIB.KFPB.RT.ADD	BIT(24)	DISPL 6028

	03	FIB.KFPB.RT.LGTH	BIT(16)	
	03	FILLER	CHAR(1)	
	03	FIB.KFPB.OV.ADD	BIT(24)	
	03	FIB.KFPB.OV.LGTH	BIT(24)	
	03	FIB.KFPB.IX.ADD	BIT(24)	
	03	FIB.KFPB.IX.LGTH	BIT(24)	
	03	FILLER	BIT(8)	
	03	FIB.KFPB.KEY.LGTH	BIT(16)	DISPL 6047
	03	FIB.KFPB.KEY.OFFSET	BIT(16)	
	03	FIB.KFPB.ZERO	BIT(32)	
02		FIB.LAST.INPUT.COMM	BIT(4)	DISPL 6055
02		FIB.IX.PARAM		
	03	TOP.SECT.NB.IN.DISK.IX	BIT(24)	
	03	TAIL.SECT.NB.IN.DISK.IX	BIT(24)	
	03	IX.REC.SECT.ADD	BIT(24)	
	03	IX.REC.IN.BLK.PTR	BIT(8)	
	03	IX.FLAGS		
	04	IX.MEM.VALID	BIT(1)	
	04	IX.KEY.FOUND	BIT(1)	
	04	IX.EOF.REACHED	BIT(1)	
	04	FILLER	BIT(1)	
02		FIB.OV.PARAM		
	03	TOP.SECT.NB.IN.DISK.OV	BIT(24)	DISPL 6066
	03	TAIL.SECT.NB.IN.DISK.OV	BIT(24)	
	03	OV.REC.SECT.ADD	BIT(24)	
	03	OV.REC.IN.BLK.PTR	BIT(8)	
	03	OV.FLAGS		
	04	OV.MEM.VALID	BIT(1)	
	04	OV.KEY.FOUND	BIT(1)	
	04	OV.EOF.REACHED	BIT(1)	
	04	OV.DEL.FOUND	BIT(1)	
02		FIB.RGH.PARAM		
	03	TOP.SECT.NB.IN.DISK.RT	BIT(24)	
	03	TAIL.SECT.NB.IN.DISK.RT	BIT(24)	
	03	RGH.REC.SECT.ADD	BIT(24)	
	03	RGH.REC.IN.BLK.PTR	BIT(8)	
	03	RGH.FLAGS		
	04	RGH.MEM.VALID	BIT(1)	
	04	RGH.KEY.FOUND	BIT(1)	
	04	RGH.EOF.REACHED	BIT(1)	
	04	FILLER	BIT(1)	
02		FIB.SRCH.PARAM		
	03	FIB.KEY.LGTH	BIT(8)	DISPL 6087
	03	FIB.REC.LGTH	BIT(16)	
	03	FIB.KEY.REC.PER.SECT	BIT(8)	
	03	FIB.REC.POSITION	BIT(16)	
	03	FIB.SEARCH.REPLY.WORD	BIT(24)	
	03	FIB.SEARCH.KEY	CHAR(32)	
02		I.S.INFO		

03	I.S.HIGHEST.KEY	CHAR(29)	DISPL 6128
03	COMM.RET.ADD	BIT(24)	
03	I.S.ACC.MODE	BIT(2)	
03	I.S.WRITE.PERFORMED	BIT(1)	
03	I.S.NULL.KEY.FILE	BIT(1)	
03	FILLER	BIT(8)	
02	READ.POINTERS		
03	KEY.RETAINED	CHAR(29)	DISPL 6161
03	INDEX.POINTER		
04	INDEX.READ.SECTOR	BIT(24)	DISPL 6190
04	INDEX.READ.KEY.NB	BIT(8)	
03	OVERFLOW.POINTER		
04	OVERFLOW.READ.SECTOR	BIT(24)	
04	OVERFLOW.READ.KEY.NB	BIT(8)	
03	READ.FLAGS	BIT(8)	DISPL 6198
04	READ.OVERFLOW.BEFORE.NEXT	BIT(1)	
04	INDEX.READ.FLAGS		
05	INDEX.KEY.FOUND	BIT(1)	
05	INDEX.EOF.REACHED	BIT(1)	
04	OVERFLOW.READ.FLAGS		
05	OVERFLOW.KEY.NOT.FOUND	BIT(1)	
05	OVERFLOW.EOF.REACHED	BIT(1)	
04	SKIP.OV.BEFORE.NEXT	BIT(1)	
04	SKIP.IX.BEFORE.NEXT	BIT(1)	
02	SAVE.PUSH.PTR		
03	SAVE.PUSH.SECTOR	BIT(24)	DISPL 6199
03	SAVE.PUSH.KEY.NB	BIT(8)	
02	SAVE.PUSH.KEY	CHAR(29)	DISPL 6203
02	SND.PART.OF.FIB	BIT(1)	

CURRENT CODE SEGMENT

The contents of the code segment active at the time of the abnormal terminate are listed here. The last instruction executed is located in this code segment. The contents are displayed in hexadecimal.

To the left of the text are numbers indicating the byte position or displacement of the first pair of digits in the corresponding line.

The code segment number and size in bytes are also listed.

Repetitive lines are suppressed and the message 'SAME AS LINE ABOVE' is given. The displacement to the left helps to determine the number of lines suppressed.

LOCKED SLICE

The contents of a program's LOCKED SLICE are displayed here. The Locked Slice of a program is located in the program's Interface Control Block (ICB) residing in the user program's partition.

The contents are displayed in hexadecimal.

To the left of the text are numbers indicating the bit position or displacement of the left-most bit of the first digit in the corresponding line, relative to the ICB base.

The LOCKED SLICE contains the following structures:

- the Program Segment Table
- the Data Segment Table
- the Task Control Block Preset Area
- the Control Stack
- the Code Control Block Preset Area
- the Internal File Name Block

Those structures are located in the LOCKED SLICE using pointers stored in the ICB.

PROGRAM SEGMENT TABLE

This table contains information about a user program's code segments. It is used by the DP.ANALYZER program to display the PROGRAM SEGMENT TABLE section.

Address : ICB.PST.PTR

Length : ICB.PST.LGTH

Data type : array

01 PST.ENTRY	BIT(8)
02 SEG.DESCR.TYPE	
02 SEG.DESCR.FLAGS	
03 ALREADY.HANDLED	BIT(1)
03 SEG.HAS.BEEN.UPDATED	BIT(1)
03 IN.VIRTUAL.FILE	BIT(1)
03 TO.BE.LOADED	BIT(1)
03 IN.USE	BIT(1)
03 IN.CORE	BIT(1)
03 LOCK.IN.MAIN.STORE	BIT(1)
03 READ.WRITE.SEG	BIT(1)
02 SEG.DESCR.DSK.ADD	BIT(16)
02 SEG.DESCR.LGTH	BIT(16)
02 SEG.DESCR.MEM.ADD	BIT(24)
02 SEG.DESCR.FN	BIT(8)
02 SEG.DESCR.ROLL.IN.COUNTER	BIT(16)

DATA SEGMENT TABLE

This table contains information about a user program's data segments. It is used by the DP.ANALYZER program to display the DATA SEGMENT TABLE section.

Address : ICB.DST.PTR

Length : ICB.DST.LGTH

Data type : identical to PST data type

TASK CONTROL BLOCK PRESET AREA

It contains the value of the eight edit characters used in COBOL and RPG programs. It is used to print the Interpreter Preset Area in the PROGRAM PARAMETERS section.

Address : ICB.TCB.PA.PTR

Length : ICB.TCB.PA.LGTH

CONTROL STACK

It contains

- the CONTROL STACK for MPL programs
- the PERFORM STACK for COBOL and RPG programs.

It is used to print the PERFORM STACK/CONTROL STACK section.

Address: ICB.CNTL.STACK.PTR

Length: ICB.CNTL.STACK.LGTH

CODE CONTROL BLOCK PRESET AREA

It contains the Message Reference Table for MPL programs, the COP table for COBOL and RPG programs.

Address : ICB.CCB.PA.PTR

Length : ICB.CCB.PA.LGTH

INTERNAL FILE NAME BLOCK

It indicates the correspondence between the name by which the program refers to the file and the indices to the data segments containing the FIB and FPB.

Address : ICB.IFNB.PTR

Length : ICB.IFNB.LGTH

Data type : array

01	IFNB.ENTRY	
02	DST.INDEX.FOR.FIB	BIT(8)
02	DST.INDEX.FOR.FPB	BIT(8)
02	FILE.NAME	CHAR(28)

APPENDIX A

GLOSSARY OF TERMS

BCD

Binary Coded Decimal — A decimal digit (0-9) is implemented in binary in a 4-bit field.

COP

Current OPerand table—contains the attributes and address of the data item pertaining to COBOL/RPG programs.

CPA

Communicate Parameter Area — is a block of contiguous bytes containing a message sent from an S-Program to the MCP requesting a function.

DST

Data Segment Table—contains information about a user program's data segments.

FCB

File Control Block — a structure, built at run time, associated with each file contained in a program. It is part of the FIB structure.

FIB

File Information Block — a structure, built at run time, associated with each file contained in a program.

FPB

File Parameter Block — a structure, built at compile time, associated with each file contained in a program.

ICB

Interface Control Block — a run time structure containing all the parameters needed by the MCP to execute the program.

IFNB

Internal File Name Block—indicates the correspondence between the name by which the program refers to the file and the numbers of the data segments containing the related FIB and FPB.

PPB

Program Parameter Block — the first record of the program file for which it gives general information.

PSN

Program Segment Number — a code segment number.

PST

Program Segment Table — contains information about a user program's code segments.

SPN

Segment Procedure Number — represents the position of an MPL procedure within a code segment.

Documentation Evaluation Form

Title: B 1000 CMS Program Dump Analysis User's Guide

Form No: 2018750
 Date: August 1982

Burroughs Machines Ltd is interested in receiving your comments and suggestions regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Suggestion:

- Addition Deletion Revision Error

Comments:

From:

Name _____
 Title _____
 Company _____
 Address _____

 Phone Number _____ Date _____

Remove form and mail to:
 Documentation Dept., TIO - Europe
 Burroughs Machines, Ltd.
 1 Tollpark Place, Wardpark East
 Cumbernauld
 Glasgow, G68 OLN
 Scotland