CONTROL DATA CORPORATION

NETWORK OPERATING SYSTEM (NOS)

VERSION 2.0

LEVEL 562/552

FEATURE NOTES

NUMBER 5

SMD130316

NOS V2 Feature Notes - 5

This is the first in the set of features notes which apply to NOS V2. These feature notes are intended to supplement the material available through the normal publications available from Control Data. They are intended as a starting point in learning what NOS V2 is all about.

The overall system concepts used in NOS V2 are not very different from NOS V1. The user of the system will find he has to change very little to begin using NOS V2. The common product set is the same release level as NOS V1 level 552 except some products/capabilities are no longer available. In fact, the user will find some new capability which will make his job easier to do.

The system programmer and operations staff will find many new things in the system. For these people the feature notes are just beginning to document what changes are involved. It is recommended that each site use the copies of the new manuals as the basis for getting ready for NOS V2. Even though the system concepts are the same, the system programmer and operation staff should view any local change with care. NOS V2 may already support that capability implemented in that local modification. NOS V2 may accept your local modification without change. NOS V2 may have a new implmentation of an existing capability and local modifications to be migrated will have to be reimplemented.

Please take the time to study before moving that local mod. Included in your materials is the FNT Reorganization Design Note. This will be a good beginning in planning your approach to NOS V2.

Included in these feature notes are articles on -

- NOS V2 Manuals
- NOS V2 Support and NOS V1
- Model 835/855 Instruction Stack
- Usability Feature Notes
  - Default charge and automatic procedure call
  - Interactive CCL
  - IEDIT
  - Improved output formats of System Utilities
  - Family Pack Access
  - User Control of Submitted Jobs
  - NOS Global Library Set at NOS 2.0
  - LIBEDIT ENHANCEMENTS
- A glimpse of SORT 5
- TAF Features for NOS V2
- Shared Mass Storage
- CYBER 176/819 Support
- FNT Reorganization
- L-display

NOS V2 MANUALS

NOS V2 MANUALS

The following material has previously been published in the Publications Newsletter Vol. 2 No. 1, January, 1982, and contains an informative discussion of the NOS V2 manual situation. It is reproduced here to provide the reader of this document convenient access to this information.

PRODUCT SUPPORT MANUALS FOR NOS2.0

Control Data Corporation recognizes that product support manual considerations can be a significant aspect of acquiring a new product or migrating from one version of a product to a successor version. Recognizing that concern, we've taken an approach for NOS 2.0 product support manuals that attempts to protect our customers from incurring large additional product support manual costs, while maintaining appropriate support for NOS 1 operating system and product set manuals during the overlap support period.

The significant product enhancements, extensions, and feature additions to NOS that culminate in a new version, NOS 2.0, prompts a requirement to prepare new product support manuals for the operating system. We traditionally create new manuals (with new publication numbers) for new product versions to permit readily discerning one version from another and to separate and simplify future maintenance activity (e.g., new feature and corrective code releases). With the exception of some DMS-170 manuals and the loader manual, product set manuals for the initial release of NOS 2.0 will not depart significantly enough from NOS 1 product set manuals to require new manuals. So despite the fact that separate product set manuals might promote easier maintenance, etc, for us, we intend to carry the same product set manuals for NOS 1 forward to support NOS 2 products.

Any feature enhancements and/or CCRs to NOS 2.0 product set members will be documented in the manuals carried forward. Corrective code or other activity prompting changes to the NOS 1 product set manuals will be reflected in the same set of product set manuals carried forward to support NOS 2.0. We will take care to distinguish between NOS 1 and NOS 2 differences, if any. Some details follow:

- New manuals supporting the NOS 2.0 release will be distributed to VIM sites via the AID (Automatic Initial Distribution) mechanism (1 microfiche copy per site).

PRODUCT SUPPORT MANUALS FOR NOS 2.0 (con't)

- Customers will be required to purchase new operating system manuals and a small subset of product set manuals if they wish to have copies in addition to that manual set shipped with the product(s). Upon purchase of these manuals, customers will be given the opportunity to subscribe to the free automatic distribution of future revisions to these manuals.

- Other than the two DMS-170 manuals identified in DMS-170 manuals and the loader manual identified in the list of new manuals which follows, product set manuals will not, as a matter of course, be assigned new publication numbers nor new revision levels. In fact we do not plan to issue any special revisions to these manuals solely to change the cover, title page, preface, etc., to note the fact that they reflect NOS 2.0. We will do this when the manuals require revision in normal feature and/or corrective code release contexts.

- We will prepare an update to the Software Publications Release History (publication number 60481000) prior to the product release that will identify all product support manuals, by revision levels, that will be released at the NOS 2.0 PSR level. We will clearly identify the revision levels of product set manuals that reflect NOS 1 unique products.

- NOS 1 manual inventories will be maintained in LDS for at least one year after the NOS 2 release. In reality, the retention period is usually longer before normal inventories deplete. After the normal inventory is depleted, we can meet special requests for manuals by providing printed or microfiche copies at customer expense.

- For any corrective code releases to the NOS 1 system during the one year CEMS overlap support period, we will issue formal revision packets to the operating system manuals. Upon termination of CEMS support for NOS 1, we will issue a final update to these manuals (if necessary) and no further revision activity to these manuals will occur.

- During the one year overlap support period, any NOS 1 changes that affect the product set manuals will be supported with normal change packets to the same product set manuals that are carried forward to serve NOS 2. Once CEMS support terminates for NOS 1, customers will need to retain the latest revision level of each product set manual should they elect to remain on NOS 1.

- Manuals not assigned new publication numbers for the NOS 2 release will continue to have revisions distributed to NOS 1 customers via the revision packet update service. Because of difficulties in managing the database for this service, we will not turn off the distribution of these manual updates to NOS 1 customers. Once we know a customer has migrated to NOS 2, LDS will change the database to reflect NOS 2 manuals for that customer. Otherwise, manuals that retain the same document number from NOS 1 to NOS 2 will have each revision distributed to NOS 1 customers who subscribe to the revision packet update service. To simplify future NOS 1 to NOS 2 migration, we suggest you retain these updates. With the eventual termination of CEMS support for NOS 1, the automatic revision packet service is retained for NOS 2 only.

Our manuals approach for NOS 2 is an attempt to provide documentation continuity where possible without the requirement to replace all manuals when migrating from NOS 1 to NOS 2. Though our approach may appear confusing at first, we trust that we haven't unduly compromised manual availability and usability while attempting to save our customers money.

Reference Set

For NOS Version 2 we have created a new four-volume reference set. The manuals in this set are described in the following abstracts; new features will be described in our April Newsletter.

NOS 2 Reference Set, Volume 1, Introduction to Interactive Usage
60459660 60 pages

This manual gives a simple introduction to the interactive use of NOS, including logging in and out. It describes in tutorial manner a subset of the commands that allow a beginning user to enter, run, and correct programs and create, retrieve, and maintain permanent files. This manual makes extensive use of examples and illustrations.

Reference Set (con't)

NOS 2 Reference Set, Volume 2, Guide to System Usage
60459670 80 pages

This manual describes the concepts of job processing, magnetic tape processing, procedures, files, source file maintenance, and file execution. Included are descriptions on how to use Xedit. Modify and the CYBER Loader. Although many commands are described, not all parameters for each command are shown.

This manual is oriented toward the applications or systems programmer who is new to NOS. It is assumed that the reader understands the material presented in Volume 1, Introduction to Interactive Usage, and also that the reader will use Volume 3, System Commands, as a primary resource once the material in Volume 2 is mastered.

NOS 2 Reference Set, Volume 3, System Commands
60459680 600 pages

This manual is written for the applications programmer who uses a higher level language. The first five sections describe the structure of the system - its hardware and software components, files, jobs, procedures, and commands. The remaining sections describe the specific commands the user can employ for the following tasks.

- Job control
- Creation and manipulation of files on mass storage
- Creation and manipulation of files on magnetic tape
- Debugging
- Library maintenance

The reader is assumed to understand the concepts described in the NOS Reference Set Volumes 1 and 2.

Reference Set (con't)

NOS 2 Reference Set, Volume 4, Program Interface
60459690 400 pages

This manual is written for the COMPASS applications programmer. The first two sections provide an overview of system macros and requests, and communications between user programs and NOS. The remaining sections describe the system macros with which the user creates and manipulates mass storage and magnetic tape files, obtains or changes job and file status, and controls job processing.

It is assumed the reader understands the concepts described in the COMPASS Reference Manual and the NOS Reference Set Volumes 1 through 3.

The following table shows the NOS Version 1 counterparts to the NOS Version 2 manuals.

NOS 2	NOS 1
NOS Version 2 Reference Set, Volume 1, Introduction to Interactive Usage, Publication Number 60459660	Interactive Facility Version 1 User's Guide Publication Number 60455260
NOS Version 2 Reference Set, Volume 2, Guide to System Usage, Publication Number 60459670	NOS Version 1 Batch User's Guide Publication Number 60436300
NOS Version 2 Reference Set, Volume 3, System Commands Publication Number 60459680	NOS Version 1 Reference Manual Volume 1, Publication Number 60435400 and Interactive Facility Version 1 Reference Manual Publication Number 60455250
NOS Version 2 Reference Set, Volume 4 Program Interface Publication Number 60459690	NOS Version 1 Reference Manual Volume 2, Publication Number 60445300

New Manuals for NOS 2.0

Manual Title	Publication Number
NOS 2.0 Reference Set Vol. 1, Introduction to Interactive Usage	60459660
NOS 2.0 Reference Set Vol. 2, Guide to System Usage	60459670
NOS 2.0 Reference Set Vol. 3, System Commands	60459680
NOS 2.0 Reference Set Vol. 4, Program Interface	60459690
NOS 2.0 System Maintenance Reference Manual	60459300
NOS 2.0 Operator/Analyst Handbook	60459310
NOS 2.0 Installation Handbook	60459320
NOS 2.0 System Overview	60459270
NOS 2.0 Applications Programmer's Instant	60459360
NOS 2.0 Systems Programmer's Instant	60459370
Network Terminal User's Instant	60459380
NOS 2.0 Diagnostic Index	60459390
TAF 1 Reference Manual	60459500
TAF/CRM Data Manager Reference	60459510
TAF 1 User's Guide	60459520
DMS-170 CYBER Database Control System Version 2 Data Administration Reference Manual	60458200
DMS-170 CYBER Database Control System Version 2 Application Programming Reference Manual	60485200
Loader User's Guide	60482300

These manuals will be available in LDS on or after April 26, 1982.

NOS V2 SUPPORT AND NOS V1

NOS V2 SUPPORT AND NOS V1

As has previously been published in the VIM Newsletter, support of some products/capabilities will no longer be available on NOS V2. Equivalent product/capabilities have previously been available on NOS V1 along with conversion aids to ease the migration to the newer product/capability. It is believed that the customer has used this time period to upgrade to the newer product/capability and no longer is reliant upon the older product/capability.

The following table lists the NOS V1 product/capability and the equivalent NOS V2 product/capability:

NOS V1 PRODUCT/CAPABILITY	NOS V2 PRODUCT/CAPABILITY	NOTE
HARDWARE		
CPU - no CEJ/MEJ or CEJ/MEJ disabled	CEJ/MEJ available and enabled	
DISK - 841	N/A	May use 844-2X, 844-4X, 885
TAPE - 65X	N/A	May use 66X, 67X
MULTIPLEXOR - 6671/6676 2550-100/ 2551-100	N/A	May use 2550-1, 2551-1, 2551-2
SOFTWARE		
COMMON PRODUCTS		
ALGOL4/ALGEDIT	ALGOL5	
AAM1	AAM2	
COBOL4	COBOL5	
C45 - COBOL4 COBOL5 CONVERSION AID	N/A	COBOL4 not available on Version 2
CDCS1	CDCS2	
DDL2	DDL3	

SIFT	N/A	
SORT4	SORT5	CEMS termination April, 1983
FTN4	FTN5	CEMS termination June, 1983

COMMUNICATIONS PRODUCTS

TELEX	IAF/NAM	
EI200	RBF/NAM	
TAF/TS	TAF/NAM	
TAF DM	TAF/CRM, TAF/CDCS	
UPMOD	N/A	MODIFY AND UPDATE BOTH ON VERSION 2
KCL	CCL	KCL TO BE REMOVED AT NOS 2.1

Questions about conversion from older products/capabilities to the newer ones with the common product set area should be addressed to User Support in Sunnyvale, with the communications products to NHP Field Support in Sunnyvale and/or NOS Field Support in Arden Hills.

CYBER 170 - 835/855 INSTRUCTION STACK

MODEL 835/855 INSTRUCTION STACK

1 INTRODUCTION

Control Data is introducing the 800 series to its CYBER 170 product line. These machines have a new hardware architecture that uses a micro coded instruction set to emulate CYBER 170 700 processing. This instruction set is as identical to that used by the other models as possible except where it would impact the performance of the machine. It is the unexpected exceptions that can cause problems.

The purpose of this article is to describe the difference between the instruction look ahead processing used on the Model 835/855 and on the other models of the CYBER 170 line. The 700 series and the Model 825 have a two word look ahead stack. The Model 835 has an additional branch look ahead. The Model 855 has a 12 instruction look ahead which assumes a branch will be taken.

2 THE LOOK AHEAD STACK

All the models in the 800 series utilize an instruction look ahead stack. The instruction look ahead stack should not be confused with the intruction stack of a stack machine like the 6600 or Model 176. The purpose of a look ahead stack is to minimize the delay in processing of instructions caused by accessing the word of memory that contains the instructions. To avoid this delay the next few words of memory are read into high speed registers so that they are available when the CPU needs them. This works very efficiently until the programmer tries to do execution time instruction modification. This is done in writing highly optimized code in which the programmer wishes to use the same section of code to perform two or more slightly different functions. In this case, the code is designed to modify itself upon detecting the type of function desired for this execution. The following program demonstrates a very simple form of instruction modification.

```
            IDENT  STACK
            ENTRY  STACK
TEMP        EQ     END
STACK       BSS    0            EXECUTION BEGINS HERE
            SA1    TEMP         PICK UP WORD OF INSTRUCTIONS
            BX6    X1           MOVE TO OUTPUT REGISTER
            SA6    MOD          STORE IN NEXT WORD
MOD         NO                  WORD OF NOPS TO ACCEPT CODE
+           ABORT               ABORT PROGRAM
END         ENDRUN              TERMINATE NORMALLY
            END    STACK
```

The above program simply takes the word of instructions assembled into the word TEMP , stores them into the word MOD and then executes the word MOD. The expected result is that the jump to the label END will be executed and the program will terminate normally. On a machine with instruction look ahead, this program will abort. The word MOD will be read into a high speed register before the store is done to MOD and therefore the word of no-ops will be executed instead of the EQ END.

An instruction look ahead stack is nothing new to CYBER machines and most self modifying code is designed to handle it, but the Model 835 has an additional conditional branch look ahead stack. The instruction look ahead stack is a 3 word stack. If the look ahead logic detects a conditional branch instruction it uses two additional registers as a branch look ahead. Into these registers it reads the first two words of memory pointed to by the conditional branch instruction. In this way, if the branch is taken two words of instructions are immediately available for execution. If the branch fails the branch look ahead registers are purged until another conditional branch instruction is detected.

The Model 855 has a 12 intruction stack which will hold the next 12 instructions to be executed. When a conditional branch instruction is detected, the look ahead code assumes that the branch will be taken. Instead of choosing the next contiguous instructions, it chooses the instructions pointed to by the conditional branch instruction.

The following variation on the stack example will illustrate these situations.

```
            IDENT   STACK
            ENTRY   STACK
TEMP        EQ      END
STACK       BSS     0           EXECUTION BEGINS HERE
            SA1     TEMP        PICK UP WORD OF INSTRUCTIONS
            SB2     B1          SET B2 EQUAL TO B1
            BX6     X1          MOVE TO OUTPUT REGISTER
            SA6     MOD         STORE INTO MOD WORD
            EQ      B1,B2,MOD   JUMP TO MODIFIED WORD
SPACE       BSSZ    100         RESERVE SPACE SO STACK CLEAR
MOD         NO                  WORD OF NOPS TO ACCEPT CODE
+           ABORT               ABORT PROGRAM
END         ENDRUN              TERMINATE NORMALLY
            END     STACK
```

This program would result in a normal termination on all older model CYBER 170 machines and on the new Model 825 machine because at the time of the store, the EQ instruction and two words of zero would be in the stack and after the jump the word at MOD would be read into the stack. On a Model 835, however, the word at MOD and the next word would be read into the conditional branch instruction stack before the store is done.

The no-ops and the ABORT macro would be executed and the program will abort. On a Model 855, the instructions at MOD would be read into the instruction look ahead stack because the look ahead logic is assuming the branch will be taken. The instructions at SPACE are not read unless the EQ condition is not met. In this case since B1 is equal to B2, the words at SPACE are not read. Since this is again a look ahead prodedure, the instructions at MOD are read before the store into MOD is performed and again the program will abort.

It is code that modifies instructions and then conditionally jumps to the modified instructions that will lead to problems when moving self modifying code to the Model 835/855 CYBER 170s.

3 CORRECTING THE PROBLEM

CDC does not recommend that users write code that does execution time modification. If it cannot be avoided, the procedure for writing self-modifying code is to void or purge the stack after the code has been stored and before executing any additional code. The only instruction that purges the stack on all CYBER machines is the return jump (RJ). Regardless of whether the RJ is to a location in or out of the stack, the entire stack is purged and new instructions are read from CM. To guarantee that either form of the STACK program above will terminate normally on all systems, the following instructions must be added after the SA6 instruction:

```
          RJ     *+1          VOID THE STACK
+         BSSZ   1
```

This will guarantee that all subsequent instructions will be read from CM before being executed. Some additional instructions void the stack on specific models but CDC will only quarantee that the RJ instruction will purge the stack on future machines.

4 ALTERNATIVES

If a program runs differently on the Model 835/855 than it did on a previous mainframe and execution time code modification is suspected, the 835/855 can be run with stack purging turned on. This mode dynamically changes the micro coding of the instruction set so that not only the RJ and JP instructions but also the conditional jump and word store instructions purge the stack. This greatly reduces the performance of the machine but will properly execute code written for any model CYBER 170 machine. Stack purging can be initiated or canceled by a command or a COMPASS macro.

The format of the command is:

MACHINE,EP=ON. - to initiate extended stack purging

MACHINE,EP=OFF. - to cancel extended stack purging

The format of the COMPASS macro is:

```
MODE  7,0,1       to initiate extended stack purging

MODE  7,0,0       to cancel extended stack purging
```

The command can be used to check if stack purging is really the problem. The command can also be used as a permanent solution if the performance of the job is not critical or if no source code is available. The macro should be used so the switching of modes is invisible to the casual user. If the source code is available but is so complex that recoding is not economical, the part of the code that does the code modification should be isolated as closely as possible. The MODE macro can be used to turn stack purging on before the code and off after the code to minimize the performance degradation. The MODE macro is a monitor request to set a bit in the jobs exchange package. Too frequent use of the MODE macro can cause a performance degradation of its own. Care should be taken not to turn stack purging off if it was on prior to entering the routine being modified. The GETEM (get exit mode) macro can be used to check the stack purge status. The GETEM macro and the MODE macro are described in the NOS Reference Set Vol. 4 (60459690). The GETEM macro returns the exit mode field of the exchange package right justified in a word of memory. Bit four of this word is set if stack purging is on and clear if stack purging is off. The status should always be checked before it is changed so that it can be restored to the value prior to the execution of this code.

5 STANDARD PRODUCTS

The current CDC standard Category I products were modified at PSR level 528 to use the RJ instruction to void the stack after any code modification. Any programs compiled and run after level 528 should run on a model 835/855 with stack purging off. CDC does not officially support running old binaries on the new machine but all binaries will probably work if stack purging is on and most should work if they were compiled and loaded on a system at level 528 or newer.

6 CAUTIONS

Because of the problems inherent in execution time instruction modification, CDC discourages use of this type of coding.

Some users may write some COMPASS programs to check out the exact workings of the stack and try to take advantage of some extra instructions that seem to void the stack. This can be dangerous since the instruction set is in micro code. This means that any future system release could include a new set of micro code that performs differently. CDC may find a more efficient algorithm for micro coding a certain instruction which makes it have a different impact on the instruction stack. Therefore, it must be stressed again that the only official and guaranteed way to purge the stack is with the RJ instruction.

NOS V2 USABILITY

Usability Feature Notes

NOS Version 2 is now more "friendly".
NOS Version 2 offers improved usability and flexibility for the end-user. The improvements are achieved with enhancements to several existing features and with the addition of several new features. Each feature is discussed individually. However, some of the features used together provide increased usability.

The three features, Default Charge and Automatic Procedure Calls, Interactive CCL, and Global Library Set can be used together to create a friendly user environment. A user logged in to the environment need only know a user name and password and no more of NOS command language. The user, whether engineer, financier, manufacturer, is then able to communicate in the language of his or her profession.

The remaining articles grouped under usability also provide ease-of-use benefits. IEDIT provides enhanced in-line editing capabilities. User control of submitted jobs allows the user to track and terminate executing jobs or queued files. Libedit enhancements to provide parameter consistency and less tedious processing. The use of both family and private pack files within a job session has been simplified and, last, under ease-of-use features, output formats have been changed to provide better readability.

NOS V2 USABILITY

DEFAULT CHARGE AND AUTOMATIC PROCEDURE CALL

This article covers usage for the default charge and automatic procedure call capabilities of NOS Version 2.

The default charge capability provides a method for automatically billing a job to a specific default charge number and project number.

The automatic procedure call capability provides a method for an optional site defined procedure file and an optional user defined procedure file to execute prior to user job processing (immediately following primary CHARGE processing).

To support these features the structure of the VALIDUZ file was changed. The answerback words were redefined and the answerback capability was eliminated. This requires conversion of VALIDUS files between NOS Version 1 and NOS Version 2 systems.

When the VALIDUS file has been converted to NOS Version 2 format, the site analyst can update the VALIDUS file fields used by this feature as follows:

1. Assign site defined system procedure file names using the new MODVAL SP parameter;

2. Assign user defined procedure file names using the new MODVAL UP parameter;

3. Assign default charge numbers using the MODVAL CN parameter;

4. Assign default project numbers using the MODVAL PN parameter.

The user can also assign or change the user procedure file name at any time using the new UPROC control statement.

UPROC(FN=FILE) or UPROC(FILE)

The user can determine what the default charge number, default project number, and user procedure file name are at any time by executing a LIMITS statement.

```
LIMITS

     USER PROLOGUE. . . . . . . . . . . . . . . . . . . . . . . . . . . . .  FILE
```

With the default charge feature installed, the user need not enter CHARGE information when logging into IAF. A CHARGE (*) statement will be automatically executed during IAF login. This CHARGE statement will perform CHARGE processing using the default charge and project numbers from the validation file. This CHARGE(*) statement will not be automatically added to non-interactive jobs but must be added by the user.

With the automatic procedure call feature installed, primary CHARGE processing will execute the system procedure file and the user procedure file if they are defined in the VALIDUS file. This procedure file execution will be charged to the customer.

However, if both procedure types exist, an UP parameter must be specified in each system procedure file so that the user procedure file will execute after the system procedure control statements have executed.

The following is an example of the system procedure file using the UP parameter. If the UP parameter has been specified on the system procedure file call, a statement to execute the user procedure file (UP) will execute at the end of the system procedure file.

```
.PROC,PROC1,UP=$$.
*
* SYSTEM PROCEDURE FILE STATEMENTS.
*
IFE,$UP$.NE.$$,END.
BEGIN,,UP.
ENDIF, END.
```

The user defined procedure must be a CCL type procedure file (CYBER Control Language). When the UP parameter on the BEGIN statement is encountered, the EPF subroutine (Execute Procedure Files) will obtain the name of the procedure which is to execute by accessing the VALIDUS file. The following example shows a user procedure called ABC.

```
.PROC,ABC.
   .
COMMENT.  USER PROCEDURE FILE STATEMENTS.
   .
REVERT.
```

The following is a dayfile resulting from execution of the above procedures.

```
XX.XX.XX.USER
XX.XX.XX.CHARGE,
XX.XX.XX.BEGIN,,ABC.
XX.XX.XX.    .
XX.XX.XX.COMMENT.  USER PROCEDURE FILE.
XX.XX.XX.    .
XX.XX.XX.REVERT.
```

1112J

NOS V2 USABILITY

INTERACTIVE CCL PROCEDURES

INTERACTIVE CCL PROCEDURES

CCL now provides interactive prompting for parameters. The procedure header has a new format to allow you to specify the required parameters, parameter descriptions and the values or syntax allowed for each parameter. The interactive procedure also provides a help facility to give you information about the procedure or the requested parameter. Prompting for a parameter occurs if any of the following situations exist.

- parameter value is in error
- required parameter is missing
- duplicate or unknown parameter appears
- help is requested

The format for the interactive procedure header is:

```
.PROC,pname*I,P1"description"=(check list),P2...Pn.
```

An "*I" appended to the procedure name denotes an interactive procedure. The "description" following the parameter is used in the prompt message as in the next example.

```
(Procedure:)
  .PROC,GETF*I,F1"INDIRECT ACCESS FILE NAME"=(*F).
  GET,F1.
  REVERT.

(Call:)
  begin,,getf,?

(System prompt:)
  PARAMETERS FOR GETF ARE  F1
  ENTER F1 INDIRECT ACCESS FILE NAME ?
```

In this example, "*F" represents the check list for the parameter F1. This tells us the values or syntax which can be specified for the parameter. In procedure GETF, F1 must have a value that follows file naming conventions. The next page contains a complete list of check list options and several procedure header examples.

Syntax	Description
*N	parameter may be omitted
*N=value	parameter=value if parameter is omitted
*N=	parameter is null if parameter is omitted
*K	parameter may be called as keyword - no substitution takes place
*K=value	parameter=value if called as keyword
*K=	parameter is null if called as keyword
*F	parameter value must be in file name format
*F=value	parameter=value if specified as a file name
*F=	parameter is null if specified as a file name
*A	parameter may be anything
*A=value	parameter=value no matter what is specified
*A=	parameter is null no matter what is specified
*Sn(set)	parameter must be an element of "set" - n indicates the maximum number of elements
*Sn(set)=value	parameter=value if set criteria is met
*Sn(set)=	parameter is null if set criteria is met
Char. String	parameter on call must match string
Char. String=value	parameter=value if parameter on call matches character string

EXAMPLES:

```
.PROC,RPT*I,DT"TODAY'S DATE"=(*A).
```
DT may be specified as any value.

```
.PROC,LIST*I,LC"MODIFY LIST OPTIONS"=(*S7(ACDEIMSTW),*N=ECTMWDS).
```
LC may be any members of the set given, with a maximum of 7 elements selected. If LC is omitted, "ECTMWDS" is the default.

```
.PROC,USERNO*I,UN"USER NUMBER"=(*K=,*F).
```
UN may be specified as any file name or called by keyword. If specified as UN, the default is null.

```
.PROC,TEXTS*I,T"SYSTEM TEXT"=(SYSTEXT,PPTEXT,N=NOSTEXT).
```
T may be specified as SYSTEXT, PPTEXT or N. If N is specified, NOSTEXT is used.

Help may be requested when using an interactive procedure. The prompt resulting from a help request contains information supplied by you and the system. You may insert information about the procedure and its parameters by using the .HELP (or .HELP,parameter) and .ENDHELP commands. The .HELP command immediately follows the procedure header and the .ENDHELP terminates the help information. The next example shows the usage of these commands.

```
(Procedure:)
  .PROC,ATTCHF*I,P1"FILE NAME"=(*N=FILE1,*F).
  .HELP
   ***THIS PROCEDURE ATTACHES A DIRECT ACCESS FILE.
  .HELP,P1
   ***IF OMITTED, FILE1 IS ATTACHED.
   ***IF SPECIFIED AS P1=FILENAM, FILENAM IS ATTACHED.
  .ENDHELP
  ATTACH,P1.
  REVERT.

(Call:)
  ATTCHF?
    or
  BEGIN,ATTCHF,,?

(System prompt:)
   ***THIS PROCEDURE ATTACHES A DIRECT ACCESS FILE.
  PARAMETERS FOR ATTCHF ARE  P1
  ENTER P1 FILE NAME ?

(Your response:)
  P1?

(System prompt:)
  ALLOWABLE VALUE(S)
  MAY BE A FILE NAME
  PARAMETER MAY BE OMITTED
   ***IF OMITTED, FILE1 IS ATTACHED.
   ***IF SPECIFIED AS P1=FILENAM, FILENAM IS ATTACHED.

  ENTER P1 FILE NAME?
```

Help for a specific parameter may be requested by entering the next call.

```
(Call:)
  -,ATTCHF,P1?

(Response:)
  ALLOWABLE VALUE(S)
  MAY BE A FILE NAME
  PARAMETER MAY BE OMITTED
   ***IF OMITTED, FILE1 IS ATTACHED.
   ***IF SPECIFIED AS P1=FILENAM, FILENAM IS ATTACHED.

  ENTER P1 FILE NAME?
```

Interactive procedures may be called from non-interactive sources, but diagnostics are issued to the dayfile if there are errors in the call. A request for help from a non-interactive call results in the help information being listed in the dayfile. The procedure is not executed.

Using a backslash (\) as a separator before a parameter on a procedure header statement means that the parameters following the backslash must be processed in equivalence mode only and can not be specified positionally. If the backslash is used as a separator before a keyword on a procedure call, it means the remaining parameters will be processed in equivalence mode.

The next 2 pages show a more detailed procedure and an interactive session using this procedure.

```
.PROC,LISTNG*I,FN"LFN"=(*N=OPL,*K=OPL552,*F),
               USNO"USER NUMBER"=(*N=LIBRARY,*K,*A),
               SYSTX"SYSTEM TEXT"=(*N=NOSTEXT,*A,*K=0),
               LOCTX"LOCAL TEXT"=(*N=0,*A),
               LOPT"LIST OPTIONS"=(*S7(ACDEIMSTW),*N=),
               DCK"EDIT DECK"=(*A).
.HELP
 THIS PROCEDURE IS USED TO GET DECK LISTINGS FROM OPL.
.HELP,FN
 SPECIFYING "FN" GIVES FILE "OPL552".
 OMITTING FN GIVES YOU FILE "OPL".
 SPECIFYING "FN=NAME" GIVES YOU FILE "NAME".
.HELP,USNO
 SPECIFYING "USNO" GIVES YOU YOUR USER NUMBER.
 OMITTING USNO GIVES YOU "UN=LIBRARY".
 SPECIFYING "USNO=STRING" GIVES YOU "UN=STRING".
.HELP,SYSTX
 SPECIFYING "SYSTX" GIVES YOU "CS=C" ON THE MODIFY COMMAND.
 OMITTING SYSTX GIVES YOU "CS=NOSTEXT".
 SPECIFYING "SYSTX=STRING" GIVES YOU "CS=STRING".
.HELP,LOCTX
 OMITTING LOCTX GIVES YOU "CG=0" ON THE MODIFY COMMAND.
 SPECIFYING "LOCTX=STRING" GIVES YOU "CG=STRING".
.HELP,LOPT
 OMITTING LOPT GIVES YOU "LO=" (SYSTEM DEFAULTS ARE USED).
.HELP,DCK
 DCK MAY BE SPECIFIED AS ANY DECK.
.ENDHELP
ATTACH,OPL=FN/UN=USNO.
MODIFY,X,Z,CS=SYSTX,CG=LOCTX,CL=OUT,LO=LOPT./*EDIT,DCK
ROUTE,OUT,DC=PR.
RETURN,OPL,LGO,COMPILE.
REVERT.
```

```
/begin,listng,,?
  THIS PROCEDURE IS USED TO GET DECK LISTINGS FROM OPL.
 PARAMETERS FOR LISTNG ARE  FN, USNO, SYSTX, LOCTX, LOPT, DCK
 ENTER FN LFN ? fn?
 ALLOWABLE VALUE(S)
    FN
 MAY BE A FILE NAME
 PARAMETER MAY BE OMITTED
  SPECIFYING "FN" GIVES FILE "OPL552".
  OMITTING FN GIVES YOU FILE "OPL".
  SPECIFYING "FN=NAME" GIVES YOU FILE "NAME".

 ENTER FN LFN ? fn
 ENTER USNO USER NUMBER ? usno?
 ALLOWABLE VALUE(S)
    USNO
 PARAMETER MAY BE OMITTED
 ANY 1-40 CHARACTER STRING
  SPECIFYING "USNO" GIVES YOU YOUR USER NUMBER.
  OMITTING USNO GIVES YOU "UN=LIBRARY".
  SPECIFYING "USNO=STRING" GIVES YOU "UN=STRING".

 ENTER USNO USER NUMBER ? usno
 ENTER SYSTX SYSTEM TEXT? systx?
 ALLOWABLE VALUE(S)
    SYSTX
 PARAMETER MAY BE OMITTED
 ANY 1-40 CHARACTER STRING
  SPECIFYING "SYSTX" GIVES YOU "CS=0" ON THE MODIFY COMMAND.
  OMITTING SYSTX GIVES YOU "CS=NOSTEXT".
  SPECIFYING "SYSTX=STRING" GIVES YOU "CS=STRING".

 ENTER SYSTX SYSTEM TEXT? systx
 ENTER LOCTX LOCAL TEXT ? loctx?
 ALLOWABLE VALUE(S)
 PARAMETER MAY BE OMITTED
 ANY 1-40 CHARACTER STRING
  OMITTING LOCTX GIVES YOU "CG=0" ON THE MODIFY COMMAND.
  SPECIFYING "LOCTX=STRING" GIVES YOU "CG=STRING".

 ENTER LOCTX LOCAL TEXT ? loctx=nostxt
 ENTER LOPT LIST OPTIONS? lopt?
 ALLOWABLE VALUE(S)
 PARAMETER MAY BE OMITTED
 ANY 1- 7 CHARACTERS FROM THE SET ACDEIMSTW
  OMITTING LOPT GIVES YOU "LC=" (SYSTEM DEFAULTS ARE USED).

 ENTER LOPT LIST OPTIONS? lopt=ctmwds
 ENTER DCK EDIT DECK? dck?
 ALLOWABLE VALUE(S)
 ANY 1-40 CHARACTER STRING
  DCK MAY BE SPECIFIED AS ANY DECK.

 ENTER DCK EDIT DECK? msm
REVERT.
```

NOS V2 USABILTIY

IEDIT/IN-LINE EDIT

IEDIT

IEDIT is an extension to the existing in-line editing functions available under NOS/IAF. The editing is performed only on line numbered primary files. A line of information (including the line numbers) may not exceed 160 characters, but the file may be of any length. You may intermix OS commands and IEDIT commands without explicitly entering or leaving the editor. IAF recognizes the IEDIT commands and generates calls to the CPU program IEDIT to process them. The following in-line editing functions are currently available with IAF:

- line number insert, replace and delete
- sequence and resequence a file (RESEQ command)
- auto mode input (AUTO command)
- text mode input (TEXT command)
- list function (LIST or LNH commands)

IEDIT provides the following new or revised functions:

- WRITE
- WRITEN
- LIST or LNH
- ALTER
- DELETE
- DUP
- MOVE
- READ

The READ, RESEQ and WRITEN commands behave in a way consistent with your current subsystem (e.g. BASIC or FORTRAN).

The IEDIT commands consist of a command verb followed by parameters. When specifying lines for any of the commands, an "*" may be used to indicate BOI, EOI or both. The term "lines" signifies one or more line numbers/ranges separated by commas.

Example: 100,200..250,350..*

These commands may not be used in procedure files. The following is a look at the eight commands and examples of their usage.

1) WRITE Command

WRITE,filenam,lines,/string/

The WRITE command copies all lines within the specified "lines" which contain "string" from the primary file to "filenam". If no string is given, then all the specified "lines" are copied. If no lines are specified, every line containing "string" is copied. Line numbers are included in the data written to "filenam". No changes are made to the primary file.

Examples:	Results:
WRITE,FILE1,200..300,/END/	writes every line between lines 200 and 300 that contains "END" to "FILE1".
WRITE,FILE1	Writes all of the primary file to "FILE1".

2) WRITEN Command

```
WRITEN,filenam,lines,/string/
```

The WRITEN command performs the same as the WRITE command except that the line numbers are not written to "filenam".

Examples:	Results:
WRITEN,FILE1,/COMMENT/	Writes every line from the primary file that contains "COMMENT" to "FILE1" without line numbers.
WRITEN,FILE1,200..300	Writes lines 200 to 300 to "FILE1" without line numbers.

3) LIST and LNH Commands

```
LIST,lines,/string/
LIST,f=filenam,q..r
```

The first form of LIST lists all lines within the specified "lines" which contain "string". If no string is given, then all the specified lines are listed. The second form lists lines q..r on "filenam". If q..r is not specified, the entire file will be listed. LNH performs the same functions as LIST, but is not preferred because it will be discontinued in the future.

Examples:	Results:
LIST,100..200,/FORMAT/	Lists all lines from line 100 to 200 that contains "FORMAT".
LNH,F=FILE1	Lists local file "FILE1".
LIST,F=FILE1,200..300	Lists lines 200 to 300 from local file "FILE1".

4) ALTER Command

```
ALTER,lines,/string1/string2/
```

The ALTER command changes all occurences of "string1" to "string2" within the specified "lines". If the second string is null, "string1" is deleted. If the first string is null, "string2" is appended to the end of the specified lines.

Examples:	Results:
ALTER,100,200..300,/2/TWO/	Changes all occurences of "2" to "TWO" in line 100 and line range 200 to 300.
ALTER,/DIMENSION//	Removes all occurences of "DIMENSION" in the entire file.
ALTER,250,//GO TO 13/	Adds "GO TO 13" to line 250.

5) DELETE Command

DELETE,lines,/string/

The DELETE command deletes all lines within the specified "lines" which contain "string". If no string is given, then all the specified lines are deleted. If no lines are specified every line in the file containing "string" is deleted.

Examples:	Results:
DELETE,150..175,/END/	Deletes every line in the range from 150 to 175 that contains "END".
DELETE,/STOP/	Deletes every line that contains "STOP".
DELETE,100..150,400	Deletes lines 100 to 150 and line 400.

6) DUP Command

DUP,q..r,n,z

The DUP command copies the lines in the range q..r after line n. The inserted lines will have new line numbers in increments of z, starting with n+z. If the new line numbers overlap existing line numbers, you will be asked whether or not to continue. If you continue, overlapped lines will be given new line numbers in increments of z until line overlap no longer exists. In the BASIC subsystem, all BASIC statements referencing lines which have been given new line numbers will be corrected to point to the new line number. The default value of n is the last line in the file, and default for z is 1.

Examples:	Results:
DUP,200..300,100,10	Lines 200 to 300 will be duplicated after line 100 and numbered in increments of 10.
DUP,100..400,,5	Lines 200 to 400 will be duplicated at the end of the file and numbered in increments of 5.

7) MOVE Command

MOVE,q..r,n,z

The MOVE command performs the same as the DUP command except that the copied lines are deleted from their original positions.

Examples:	Results:
MOVE,200..500,150,1	Lines 200 to 500 will be inserted after line 150 and numbered in increments of 1. They will be deleted from their original positions.
MOVE,205,100	Line 205 will be moved from its current position to after line 100.

8) READ Command

READ,filenam,n,z

The READ command performs the same as the DUP command except that the lines to be inserted are taken from "filenam". The new lines will either be resequenced if they have line numbers, or will have line numbers added.

Examples:	Results:
READ,FILE1,100,10	Lines from "FILE1" will be inserted after line 100 and numbered in increments of 10.
READ,FILE1	Lines from "FILE1" will be inserted at the end of the file and numbered in increments of 1.

The next 2 pages show the results of modifying the file SAMPLE by using IEDIT commands.

```
 primary;sample
PRIMARY;SAMPLE.
/list
00100 line 1   This is a sample file for use with iedit.
00110 line 7   It will be modified by iedit commands.
00120 line 3
00130 line 2
00140 line 4
/


write,samp1,100..110
/list,f=samp1
00100 line 1   This is a sample file for use with iedit.
00110 line 7   It will be modified by iedit commands.
/


writen,samp2,100..140,/iedit/
/list,f=samp2
line 1   This is a sample file for use with iedit.
line 7   It will be modified by iedit commands.
/


list,100..*,/line 7/
00110 line 7   It will be modified by iedit commands.
/


alter,*..*,/iedit/IEDIT/
00100 line 1   This is a sample file for use with IEDIT.
00110 line 7   It will be modified by IEDIT commands.
/list
00100 line 1   This is a sample file for use with IEDIT.
00110 line 7   It will be modified by IEDIT commands.
00120 line 3
00130 line 2
00140 line 4
/


move,120,130,10
INSERT LINES OVERLAP EXISTING LINES, AT 140.
ENTER Y TO CONTINUE OR N TO STOP
? y
00140 line 3
00150 line 4
/list
00100 line 1   This is a sample file for use with IEDIT.
00110 line 7   It will be modified by IEDIT commands.
00130 line 2
00140 line 3
00150 line 4



list,f=file1
line 5
line 6
/
```

```
 read,file1,100,10
INSERT LINES OVERLAP EXISTING LINES, AT 110.
ENTER Y TO CONTINUE OR N TO STOP
? y
00110 line 5
00120 line 6
00130 line 7   It will be modified by IEDIT commands.
00140 line 2
00150 line 3
00160 line 4
/list
00100 line 1   This is a sample file for use with IEDIT.
00110 line 5
00120 line 6
00130 line 7   It will be modified by IEDIT commands.
00140 line 2
00150 line 3
00160 line 4
/


dup,110..130,,10
00170 line 5
00180 line 6
00190 line 7   It will be modified by IEDIT commands.
/list
00100 line 1   This is a sample file for use with IEDIT.
00110 line 5
00120 line 6
00130 line 7   It will be modified by IEDIT commands.
00140 line 2
00150 line 3
00160 line 4
00170 line 5
00180 line 6
00190 line 7   It will be modified by IEDIT commands.
/


delete,110..130
00110 line 5
00120 line 6
00130 line 7   It will be modified by IEDIT commands.
/list,f=sample,*
00100 line 1   This is a sample file for use with IEDIT.
00140 line 2
00150 line 3
00160 line 4
00170 line 5
00180 line 6
00190 line 7   It will be modified by IEDIT commands.
/
```

NOS V2 USABILITY

IMPROVED OUTPUT FORMATS

IMPROVED OUTPUT FORMATS OF SYSTEM UTILITIES

The "Improved Output Formats" feature makes the information displayed at a terminal more useable and expands some of the capabilities. The commands affected include MODIFY, OPLEDIT, LIMITS, VERIFY, ENQUIRE and LIBEDIT. The areas which have been changed include:

1. All output messages displayed on a terminal will be limited to 72 characters.

2. The default value for the LO parameter (listing options) will be LO=E for all terminal commands (only errors are listed).

3. The LO and L options will be the same in all commands. This includes both the use of the parameter and its values. (Only the LIBEDIT command has changed the useage of these two parameters).

4. The HELP command has been expanded to include all commands which can be issued from a terminal.

5. The LIMITS command has been modified to list only those values which are meaningful to the terminal user.

NOS V2 USABILTIY

FAMILY PACK ACCESS

FAMILY PACK ACCESS

This feature is designed to simplify the process of using both family and private pack file residence. It includes:

* Support Of The "PN=0" parameter on permanent file requests (commands and macros) - This will allow you to access files on your family after a PACKNAM command or macro has been executed. The affected PF requests are:

  ATTACH, APPEND, CATLIST, CHANGE, DEFINE, GET, OLD, PERMIT, PURGALL, PURGE, REPLACE, and SAVE.

* R Parameter For The PACKNAM Statement
  This feature will allow you to specify the device type once, on the PACKNAM command or macro, and it won't be necessary to specify it on subsequent permanent file actions.

EXAMPLES

PRE-R6

```
------
PACKNAM, ABC.
GET, FILE1/R=DJ.
PACKNAM, .
ATTACH, FILE2.
PACKNAM, ABC.
GET, FILE3/R=DJ.
```

R6

```
--
PACKNAM, ABC/R=DJ.
GET, FILE1.
ATTACH, FILE2/PN=0.
GET, FILE3.
```

NOTE: This feature also includes a change to CATLIST to display the User Index when no USER command has been executed (e.g., after a SUI).

NOS V2 USABILITY

USER CONTROL OVER SUBMITTED JOBS

User Control Over Submitted Jobs

1.0 Overview

The "User Control Over Submitted Jobs" feature allows you to

a) Assign a user job name to each job.
b) Determine the current status of each job.
c) Drop jobs.
d) Place jobs into the new "WAIT" queue.
e) Retrieve jobs from queues.
f) Detach the terminal from the running job.
g) Connect the terminal to a recoverable job.
h) Control execution of a job.

These capabilities are included in the following commands:

New commands:

SETJOB, QGET, DROP, CFO, GO, PAUSE and Detach(CTRL/D).

Changed commands:

ROUTE, ENQUIRE, ONSW, OFFSW and RECOVER.

2.0 User Defined Job Name

You can assign a 1-7 character user job name (UJN) to each job you own. This name can be used throughout the life of the job for job identification. You can assign the user job name in two ways:

1) For the running job (see section 7.0):

SETJOB,UJN=ujn.

2) For a job being routed to a queue:

ROUTE,lfn,DC=dc,UJN=ujn.

The user job name can be used to enquire about the status of a job through the use of the ENQUIRE command (see section 4.0).

The default user job name assigned to each job depends on the jobs origin:

Batch jobs = First parameter of the job command.
Interactive jobs = 4 character user hash code.

3.0 Use of the WAIT queue

User owned jobs can be placed into a new "WAIT" queue to be held for future access. This can be accomplished by use of the DC=q parameter in the ROUTE and SETJOB commands. The SETJOB command parameters are described in section 7.0. The new values for the q parameter in the ROUTE command are:

TO = put job in input queue - output goes to wait queue
TT = put job in wait queue
NO = put job in input queue - output is thrown away
DF = put job in input queue - output depends on the job's origin type

4.0 Determining Job Status

The ENQUIRE command has been enhanced to allow you to display more status about your jobs and to better select which jobs to display. The new forms of the ENQUIRE command are:

ENQUIRE,JSN.	Returns the general status about all jobs and output files that are owned by you.
ENQUIRE,JSN=jsn.	Returns detailed status about the job with a job sequence name of jsn.
ENQUIRE,UJN.	Returns the JSN, UJN and the latest control point message for all jobs and output files which are active owned by you.
ENQUIRE,UJN=ujn.	Returns the JSN, UJN and the latest control point message for all jobs which have a user job name of ujn.

(Note: The JN parameter is no longer available with the ENQUIRE command.)

5.0 Dropping Jobs

The DROP command allows you to drop executing jobs and queued jobs. Only jobs which belong to you can be dropped by the DROP command. The DROP command will not drop the job from which it is issued. The formats for the DROP command are:

```
DROP,jsn,q,ujn.

DROP,JSN=jsn,DC=q,UJN=ujn.
```

jsn = The job sequence name of the job to be dropped.

ujn = The user job name of the jobs to be dropped.

q = The queue of the jobs to be dropped. (Must be specified if neither jsn nor ujn specified)

- PR = print
- PU = punch
- PL = plotter
- IN = input
- EX = executing (default)
- TT = wait
- ALL = all queued jobs

6.0 Retrieving Job Output

The new QGET command allows you to assign files from any output queue as local files to your job. The formats for the QGET command are:

```
QGET,jsn,q,ujn,lfn.

QGET,JSN=jsn,DC=q,UJN=ujn,FN=lfn.
```

jsn = The job sequence name of the queued file.

q = The queue in which the file is queued.

- PR = print
- PU = punch
- PL = plot
- TT = wait (default)

ujn = The user job name of the queued job.

lfn = The name assigned to the attached file (If lfn is not specified the file name will be the value of ujn (or jsn if ujn is also unspecified)).

Either the JSN or the UJN parameter (or both) must be specified.

7.0 Detaching Jobs

Detaching a job is a new capability which is available only to IAF users. You may detach your job at any time. After detaching a job the terminal will be assigned to a new job (new JSN) and will go through the recovery dialogue.

The detached job may continue to execute. Control over the detached job may be regained through the use of the RECOVER command (See section 8.0).

Before detaching the job you can use the SETJOB command to change the control parameters for the job. These parameters will take effect at the time of the detach. The format of the SETJOB command is:

```
SETJOB,UJN=ujn,DC=q,OP=op.
```

ujn = The user job name assigned to the job.

dc = The disposition code assigned to the job.

- TO = queue job output to wait queue
- NO = do not queue the output
- DF = Default (output disposition depends on the job's origin type)

(The initial value for q is DF).

op = The job processing option (takes effect when the job terminates).

- SU = Suspend the job for an installation defined period of time after all job steps have completed.
- TJ = Terminate the job after all job steps have completed.

(The initial value for op is SU).

The actual detaching of the job takes place when the Detach command (Control D) is issued from the terminal. Any commands which are entered following the Detach command will apply to the new terminal job and not the detached job.

8.0 Recovering Jobs

The RECOVER command can be used (by interactive users only) to recover detached jobs which belong to you. The formats of the RECOVER command are:

```
RECOVER.

RECOVER,jsn,T.

RECOVER,JSN=jsn,OP=T.
```

jsn = The job sequence name of the job to be recovered.

T = Terminate recovery processing if no recoverable jobs (if there are recoverable jobs the system will inform you).

If the JSN value is not specified (first format) then a list of recoverable jobs will be displayed and the JSN value requested by the system.

9.0 Controlling Jobs

9.1 Sending Data to an Executing Job

The new CFO command allows you to send data to an executing job. The data is placed into location RA+70B through RA+74B of the program's field length. The CFO flag, (bit 14 of RA) can be used by the program to indicate when it will accept data from the CFO command. The CFO command clears the PAUSE flag after the data is placed into the program's field length (see section 9.2).

The format of the CFO command is:

```
CFO,jsn.data
```

jsn = The job sequence name of the job to receive the data.

data = The data for the job (up to 36 char).

9.2 Control Over an Executing Job

The PAUSE command can be used to suspend execution of a job. Only jobs that belong to you may be suspended by this command. The PAUSE command has the following format:

```
PAUSE,jsn.
```

jsn = The job sequence name of the job to be suspended.

The GO command can be used to continue a job which has been suspended due to an error, a request for input or a PAUSE command being issued. Only jobs that belong to you may be continued by this command. The format for the GO command is:

```
GO,jsn.
```

jsn = The job sequence name of the job to be continued.

9.3 Sense Switch Control

The ONSW and OFFSW commands can be used to turn on and off sense switches for any jobs which belong to you. The format for the ONSW and OFFSW commands are:

```
ONSW,switch1,switch2,.......switchn,jsn.

OFFSW,switch1,switch2,.......switchn,jsn.
```

switch = Sense switch ordinal (0-6)

jsn = The job sequence name of the job which the sense switch is associated with.

NOS V2 USABILITY

GLOBAL LIBRARY SET

NOS Global Library Set at NOS 2.0

The Global Library Set feature is part of the NOS 2.0 operating system. A user library can be declared a global library with the LIBRARY command. From that point on any program or or procedure contained in the library can be executed by providing the entry point name of the program or procedure name. A program can also load overlays randomly by name from this library.

In conjunction with support of the Global Library feature LIBGEN was modified to allow processing of several new record types:

- absolute
- overlay
- procedure

relocatables, capsules, and overlay capsules are still processed by LIBGEN.

Segmented programs are not supported.

1 LIBRARY Command

The LIBRARY command specifies one or more file names or system library names as a set of global libraries. This command also controls the initial CYBER Loader search order. The command format is:

```
LIBRARY,libname(1),...libname(n)/parm.
```

where:

libname(i) The user library which is to be made part of a global library set The user library must either be a local file or a system library.

parm Specifies one of the following options:

- A Add the specified library or libraries to the global library set.
- D Delete the specified library or libraries from the global library set.
- R Replace the current global library set with the library set specified on the command.

The default is R which is the way the LIBRARY command behaved before NOS Version 2.0.

Entering:

LIBRARY.

will clear global library set processing.

Notes:

1. The search order is specified by the order of user libraries on the LIBRARY command. That is, if a duplicate name exists in multiple libraries, the FIRST occurrence is used.

   If library search order is important, you should use the CYBER Loader commands LIBLOAD and SATISFY, described in the CYBER Loader Reference Manual.

2. The number of user libraries allowed has not changed, the limit is still two non-system libraries. This limit may be circumvented by having a user library with a local file name the same as a SYSTEM library name, for example, BIT8LIB. This technique has its limitations, as you should not use a SYSTEM library with the same name as one from which you want to satisfy external references.

3. You should consider that relocatable load time increases considerably as the number of entry points to search increases.

   Grouping routines that are likely to be used together in libraries is far more efficient than placing many disassociated routines in one library.

2 Special Entry Points

Special entry points which are honored for global libraries are:

RFL= field length for the program

NPC= NOS parameter cracking. This entry point indicates command parameters are to be cracked in operating system format rather than the default product set format. Thus, you no longer have to use a leading slash if this entry point is defined.

3 How It Works

LIBRARY constructs a four word entry on a local file, ZZZZZLD, for each entry point in the global library set which can be executed through a command. The entry contains information concerning:

- entry point name and record type
- field length and NPC= entry point indicator
- user library name and location

Entry points of routines that do not contain transfer addresses (e.g., FORTRAN subroutines and functions) are not placed in this file.

When a command is entered, 1AJ searches the ZZZZZLD file only if the global library bit is set (via LIBRARY). If a global library set is not declared, loading works exactly as before.

IAF no longer prefixes commands entered in the BATCH subsystem with a dollar sign. This permits entry points such as REWIND to be executed from user libraries rather than the SYSTEM. The dollar sign can still be used to force routines to run from the SYSTEM.

The ZZZZZLD file, as well as the file(s) specified on the LIBRARY command, have NO AUTO-DROP status (new at Version 2.0) which protects it from:

OLD,file. NEW,file. CLEAR,*. RETURN,*.

and the like. A RETURN or UNLOAD command that explicitly names a NO AUTO-DROP file causes that file to be released.

Note that the CRM error file, ZZZZZEG, and the FILE command file, ZZZZZDG, also have NO AUTO-DROP status at NOS Version 2.0.

4 Overlay Loading

Overlays can now be part of a global library set. The following are some things to consider:

1. Overlays with the same number (e.g., two (1,0) overlays) are differentiated through the overlay name. In this case the CALL OVERLAY subroutine should then indicate the overlay name and not the file name of the file containing the (i,j) overlay.

2. Overlay search is now done in the following fashion:

1. If there is a FOL (fast overlay loader) directory, it is used to locate the overlay. The FOL is documented in the CYBER Loader Reference Manual. This is the fastest method to load overlays.
2. If it is a local file load, 1AJ checks for an OPLD record. If the OPLD record is present, the overlay is accessed randomly.
3. Otherwise, an end around search for the overlay is performed, as before.

5 Example 1

File CCLPROC contains:

```
.PROC,CCLSHOW,PARM1.
NOTE,OUTPUT.+THIS IS CCLSHOW WITH PARAMETER PARM1.
```

File RELPRGM contains:

```
      PROGRAM RELPRGM
      PRINT *,'THIS IS RELPRGM'
      END
```

from which LGO is created:

```
FTN5,I=RELPRGM,L=0.
```

LIBEDIT can be used to create a user library through the new U parameter which automatically invokes the LIBGEN utility. LIBEDIT, by default, automatically inserts non-replaced records at the end of the new file.

```
/libedit,p=0,u,n=mylib.
ENTER DIRECTIVES -
? *file,cclshow
?
      RECORDS WRITTEN ON FILE MYLIB
         RECORD    TYPE      FILE       DATE      COMMENT

 ADDED    RELPRGM   REL       LGO       81/12/14.
 ADDED    CCLSHOW   PROC      CCLSHOW
 LIBRARY GENERATION COMPLETE.
```

The global library set, MYLIB, is declared through:

```
LIBRARY,MYLIB.
```

NOS Global Library Set at NOS 2.0

This can be verified via ENQUIRE:

```
/enquire,l.

 LOADER INFORMATION.
    MAP OPTIONS  =  DEFAULT
    DEBUG = OFF
    GLOBAL LIBRARY SET IS -
 ABSLIB
```

Programs can now be loaded from MYLIB, as in:

```
/cclshow,option.
THIS IS CCLSHOW WITH PARAMETER OPTION.
$REVERT.CCL
/relprgm.
 THIS IS RELPRGM
     0.005 CP SECONDS EXECUTION TIME.
```

6 Example 2

You can combine the user prologue and global library set features of NOS Version 2.0 to provide an 'environment' upon login.

For example, you could use these capabilities for a user name where such things as printing, modifying, and adding information to a file is done. Suppose procedures PRINT, MOD, and ADD invoked routines to perform this. You could do the following to automate this process:

- construct a user library, say, EDITLIB, containing the PRINT, MOD, and ADD procedures

- write a procedure, say, START, including commands to declare EDITLIB a global library set, as in:

```
GET,EDITLIB.
LIBRARY,EDITLIB/A.
```

- declare the START procedure as your user prologue:

```
UPROC,FN=START.
```

- login and enter:

```
PRINT   to print
MOD     to modify
ADD     to add
```

7 Conclusion

Global Library Set is one of the major usability features of NOS Version 2.0. With this new feature, you will be able to execute a series of program steps by name, rather than by the several steps required before. Also, by using global libraries many programs can be consolidated on one file, easing the usage of various programs which formerly had to be kept as separate files.

NOS V2 USABILITY

LIBEDIT ENHANCEMENTS

LIBEDIT ENHANCEMENTS

* There have been substantial changes and additions to both the command parameters and the directives (see below`

* The capability of invoking LIBGEN from LIBEDIT has been developed as part of the GLOBAL LIBRARY SET feature.

* LIBEDIT will now automatically insert any "new" (a named record present on file "LGO", which is not present on file "OLD") just before the end-of-file on file "NEW".

* LIBEDIT directive processing has been changed during interactive sessions. Now, if a directive error is detected, LIBEDIT will prompt you to re-enter the directive (instead of aborting).

Parameter Changes

1. L — The output file name parameter has been changed from"LO" to "L".

2. LO — The list option parameter has been changed from "L" to "LO". Also, there are now five separate list options:

   LO=E - list errors
   LO=M - list modifications
   LO=C - list directives
   LO=N - full list of records writen to file "NEW"
   LO=F - full list

   The new defaults are: LO=EM, for interactive output, and LO=F for non-interactive output.

Parameter Additions

1. NA — No Abort on directive errors, same as the old "D" parameter ("D" has been retained for compatibility).

2. NI — No new record will be inserted at the end of file.

3. NR — Files "OLD" and "NEW" will not be rewound after processing. Same as the old "R" parameter.

4. BRIEF — When used under IAF, the printing of title lines on output will be suppressed.

5. C — After processing has been completed, file "NEW" will be copied over file "OLD".

New Directives

1. LIST,FN,EMCNF — Changes the output list file to "FN", and the list options to E, M, C, N, or F.

2. NEW,FN — Sets file "FN" as the new file.

3. OLD,FN — Sets file "FN" as the old file.

4. NOINS — No new records will be inserted at the end of file.

5. NOREW — Files "NEW" and "OLD" will not be rewound after processing.

6. VFYLIB — After processing is complete, file "NEW" will be verified against file "OLD".

7. DEBUG — This directive causes errors to be reported, but ignored. Processing continues normally with the next directive. It is equivalent to the NA command parameter. The directive is automatically selected when output is to a terminal.

GLOBAL LIBRARY SET Changes

1. U Parameter — This control card parameter will cause LIBEDIT to call LIBGEN, when normal processing is complete. LIBGEN will place the ultimate user library on file "NEW".

2. NX Parameter — The normal LIBGEN parameter will be accepted by LIBEDIT, and will be passed to LIBGEN.

3. LIBGEN,FN — This new directive will call LIBGEN to create a user library on file "FN". It is equivalent to the "U" parameter.

SORT 5

A GLIMPSE OF SORT 5 DESIGN AND INTERNALS

I. Brief Introduction to Sort 5

Sort/Merge 5 is a new product being released for the first time under the NOS/BE 1 operating system at level 552, and under the NOS 2.0 operating system. This paper will refer to it as Sort 5.

Sort 5 contains all the features of Sort/Merge 4 with some exceptions. Sort 5 does not support Exit 6 owncode, does not take checkpoint dumps, and does not use tape scratch files. Sort 5 does have some new features as described below.

II. New Features

1) Sort 5 can be called with a command powerful enough for 99% of typical sorts. The simplest form is "SORT5. A B" which will sort file A using the entire record as the key and put it on file B. The example "SORT5. A A 11..15" will reorder file A according to characters 11 through 15. Multiple keys and a variety of key types can also be specified, as can other options. A directive file can also be specified.

2) Sort 5 can be called with an interactive dialogue. Novice users need only remember "SORT5.DIALOG=YES" to invoke an interactive dialog that will eventually sort their file. This dialog offers a HELP facility.

3) Sort 5 can "sum" records. This is the ability to combine two or more records with equal key values into a single record, while summing values from a specified field and leaving the sum in the new record. For example, the headquarters of a chain of stores could have one file per store, each containing records with a product identifier and the amount ordered. If all files are fed into Sort 5, summing will generate a file with only one record per product and having the total amount of that product to be ordered for all stores.

4) Sort 5 can sort signed integers written according to the FORTRAN "I" FORMAT, such as " 123" or " -456". This cannot be done by defining a special collating sequence because it fails for negative numbers.

5) Sort 5 suppresses voluminous dayfile messages. This is particularly useful for sites with limits on the size of dayfiles. Even when repeatedly called from a user program Sort 5

does not issue dayfile messages; it instead passes various information back to the caller thereby giving it the option of writing some or all of the information to the dayfile. When called from a command Sort 5 finishes with a dayfile line saying "n RECORDS SORTED".

III. Design Goals

Sort 5 is designed to be reliable, usable, maintainable and to perform well.

Its reliability is assured by several factors. The main factor is that Sort 5 was thoroughly analyzed and designed in a structured manner before any code was written. Another important factor is that it was written in a high-level language except for machine-dependent functions or interfaces to the operating system. All complex data structures are in the high-level language and only necessary values are given to COMPASS subroutines. One advantage of designing the product before it was coded was that memory is managed better: as a result, much more code is devoted to the preparation of the sort phase. Emphasis was placed on easy-to-understand code even if it was a little more wordy. The final factor in Sort 5's reliability is that it was very thoroughly tested.

Sort 5 is designed to be usable. It can be called with a command powerful enough for 99% of typical sorts. Novice or forgetful users can invoke the interactive dialog to help them sort files. Sort 5 can also be called from programs using a set of procedures whose names and parameters closely follow the format of the command. For simplicity, Sort 5 deals only with the standard character set, although many collating sequences are available.

Sort 5 is also usable in that conversion from Sort 4 is easy. The most difficult part of conversion would have been converting the calls to Sort 4-FORTRAN interface routine to the new Sort 5 interface. Fortunately Sort 5 completely supports the old interface. We also have the policy that if someone finds an incompatibility at execution time between Sort 4's interface and Sort 5's interface then we will change Sort 5 rather than changing Sort 4 or the reference manuals. It is true that Sort 5 does not support Sort 4's directive format, but most of such conversion involves deleting the directive file rather than creating a new directive file.

Sort 5 is designed to be more easily maintained. Its structured design makes it much easier for maintenance programmers to understand its overall organization and logic. Its structured implementation makes it much easier to understand and possibly change data structures and interfaces between modules. Use of a high-level language makes it much easier to understand the detailed logic and meaning of data items, and eliminates most errors in picking up data or calling other modules. Sort 5 can also be built as a debug version. This debug version has much code of the form:

```
IF condition-that-should-not-occur THEN abort-with-message,
or, IF list-debug-messages THEN print-useful-items,
or simply, print-useful-items
```

The debug version also writes a trace file with useful information including the common form of the specifications and a list of generated code. Tools are also kept on the PL that list comdecks and build and update working files.

Sort 5 is designed for better performance. One far-reaching technique is to compile all code that will do the actual sorting. This means that execution will be faster, more space will be available for sorting and special cases can be optimized without impacting the general case. Special case code is generated for fixed-length records, summing, RETAIN option, combinations of owncode, and fast I/O for BT=C, RT=F or BT=I, RT=W records. Sort 5 manages its memory so it has a lot of room for intelligent code before the actual sorting and merging phases. The compiled code ensures that the actual sorting and merging phases have enough intelligence for each particular sort while using a minimum amount of memory for code. Sort 5 reformats each user record into a more efficient form by keeping the minimum number of bits of miscellaneous information followed by key fields in a form ready for a quick compare, followed by the non-key fields of the record. Sort 5 also has an intelligent merge phase to help it choose the fastest sequence of intermediate merges, considering available memory, the lengths of the intermediate files, and where the intermediate files reside.

IV. Implementation of Sort 5

Sort 5 has been implemented to achieve the above design goals Most aspects of implementation closely follow the design guidelines. I will discuss some aspects of implementation not obvious from the above design goals.

Sort 5 has a number of phases: Parameter Gathering, Parameter Analysis, Code Generation, and Execution. The Parameter Gathering phase gathers parameters according to the method used to call Sort 5 - command, directive file, interactive dialogue or procedure calls. The Parameter Analysis phase takes these raw parameters, analyses them, issues diagnostics and saves the parameters in a common format for the next phase. The Code Generation phase generates code from the digested parameters, and the Execution phase executes this code. Long sorts then have a Merge Design phase, another Code Generation Phase and another Execution phase.

Figure 1 shows a map of memory during the various phases of a sort. Almost all of the code is in capsules that are loaded into memory only when needed. The main capsule, S$MAIN, is loaded into memory either by the SORT5 command overlay or by the SM5END or SMEND procedure call. S$MAIN ensures that permanent CMM blocks such as the CRM group directory or FITs are in low memory. Capsules are then loaded, executed and unloaded to gather and analyze parameters. Finally the sort phase controller is loaded and takes over. This controller directs generation of sort code. At this point a fraction of a second has been spent, and the actual sort execution is about to begin. Memory space is very valuable during execution so only the controllers and the generated code are kept in memory. The sort execution phase uses all available memory and a lot of the time to get user records and sort them onto strings of records on files. The actual sort process is explained below. The average string length is about twice the size of the working storage area. For short sorts (files that fit in the working storage area) the sort is now complete. For long sorts the Merge Design phase is loaded, executed and unloaded to tell the Execution phase how to merge the internal files. Code is generated and executed to produce just a few internal files. Then the final code is generated and executed to merge these internal files and revert the internal records back to their original format.

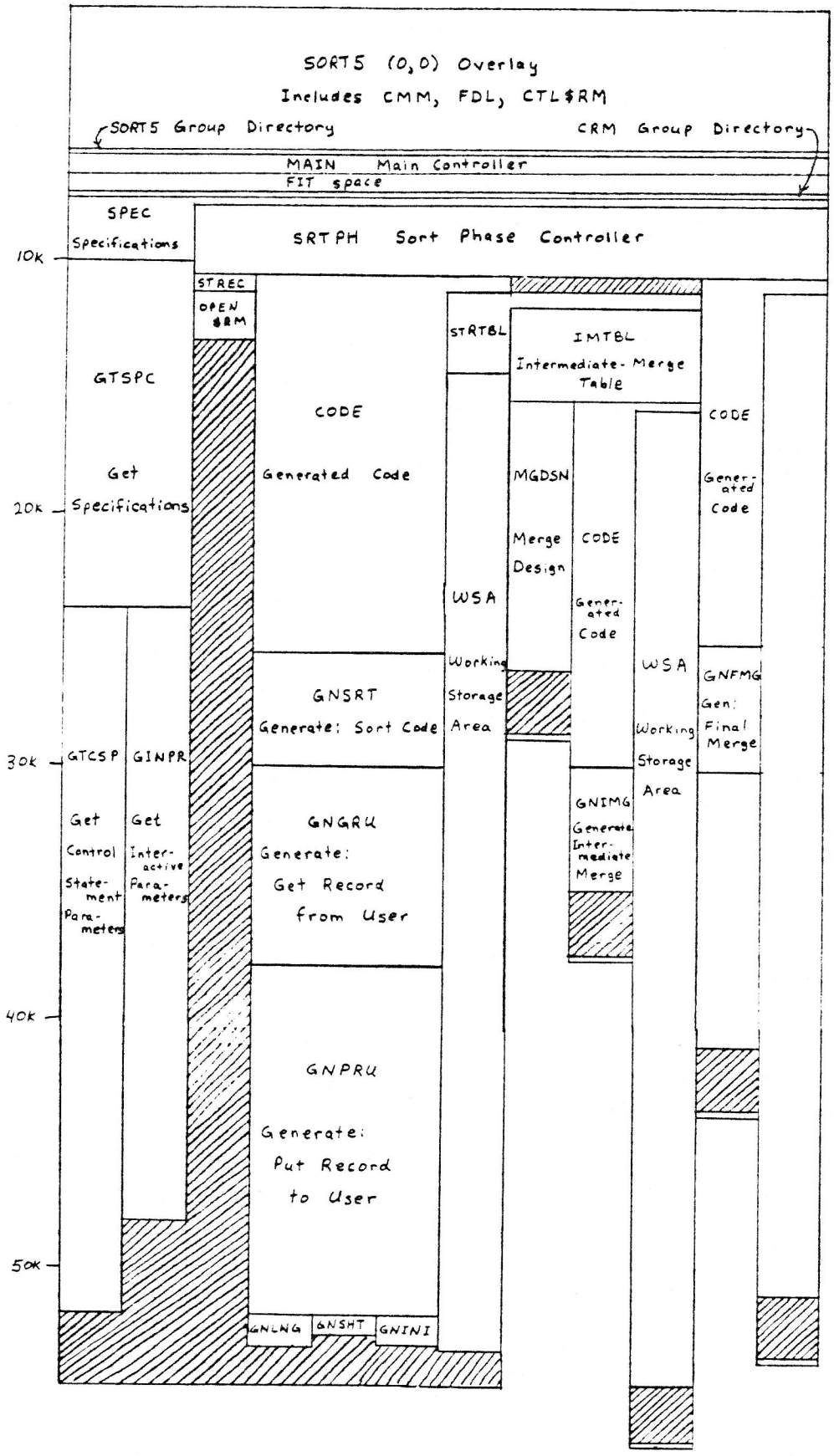
SORT5 (0,0) Overlay
Includes CMM, FDL, CTL$RM
SORT5 Group Directory
CRM Group Directory
MAIN Main Controller
FIT space
SPEC
Specifications
SRTPH Sort Phase Controller
10K
STREC
OPEN $RM
STRTBL
IMTBL
Intermediate-Merge Table
GTSPC
Get Specifications
CODE
Generated Code
MGDSN
Merge Design
CODE
Generated Code
CODE
Generated Code
20K
WSA
Working Storage Area
GNSRT
Generate: Sort Code
WSA
Working Storage Area
GNFMG
Gen: Final Merge
30K
GTCSP
Get Control Statement Parameters
GINPR
Get Interactive Parameters
GNGRU
Generate: Get Record from User
GNIMG
Generate Intermediate Merge
40K
GNPRU
Generate: Put Record to User
50K
GNLNG
GNSHT
GNINI

Figure 1

The Sort Execution Phase manages the resources of fast-but-limited central memory and slow-but-voluminous disk space so as to sort all records given it. Only central memory is suited for shuffling records around to actually sort them - any technique to use disk space in a similar manner would involve at least one disk access per record sorted. The disk space is used to save whole strings of already-sorted records. Central memory is used to either produce these strings, or to read several strings at once and merge them to a single long string. The former process is known internally as sorting, and the latter as merging. Merging is further broken down as intermediate merges where strings are merged to form fewer strings, or the final merge where strings are merged and records given to the user's file and/or owncode.

The sort technique is determined by the fact that strings will be merged and that elapsed time is roughly proportional to the amount of I/O done. If the strings for a sort (before any intermediate or final merging) are twice as long as another sort with the same input then there will be half as many strings. The maximum number of strings that can be efficiently merged at one time is about 15, due to buffer sizes and operating system performance. Suppose a sort has 30 units of records that are internally sorted one unit per string. This produces 30 strings which cannot be merged directly to output. One or more intermediate merges must be done, so there will be 90 units of I/O volume. On the other hand, if these 30 units of records were originally sorted two units per string then there would be 15 strings. These 15 strings could then be merged for final output with only 30 units of I/O volume. Thus, initially having two units per string results in a tremendous savings in performance in this case. For shorter sorts it may not make a difference. A file size of 6 units would have 6 units of I/O volume regardless of initial string size.

Since the initial string size is so critically important for the performance of large sorts the sort technique must produce strings as long as possible. Most sort techniques can sort no more records than can fit in central memory. The exception is a replacement sort. As soon as the smallest record is determined it is written to a file and is replaced by a new record, and the process repeated. Each new record may or may not be able to fit on the current string. Eventually central memory will fill up with records that cannot fit on the current string, and a new string will have to be started.

For randomly ordered input records a replacement sort will produce strings twice as big as central memory. The worst case is reverse ordered input records, producing strings just as big as central memory. The best case is input records

ordered well enough that all out-of-order records will fit in central memory, producing one giant string.

Several replacement sorts are possible. Sort 5 uses a variant of a tournament replacement sort. Figure 2 shows 12 records competing in a tournament. At this point the tournament resembles a tennis tournament having players a through k with a being the best. Each pair of players/records compete and each winner goes on to play other winners, until the overall winner is determined.

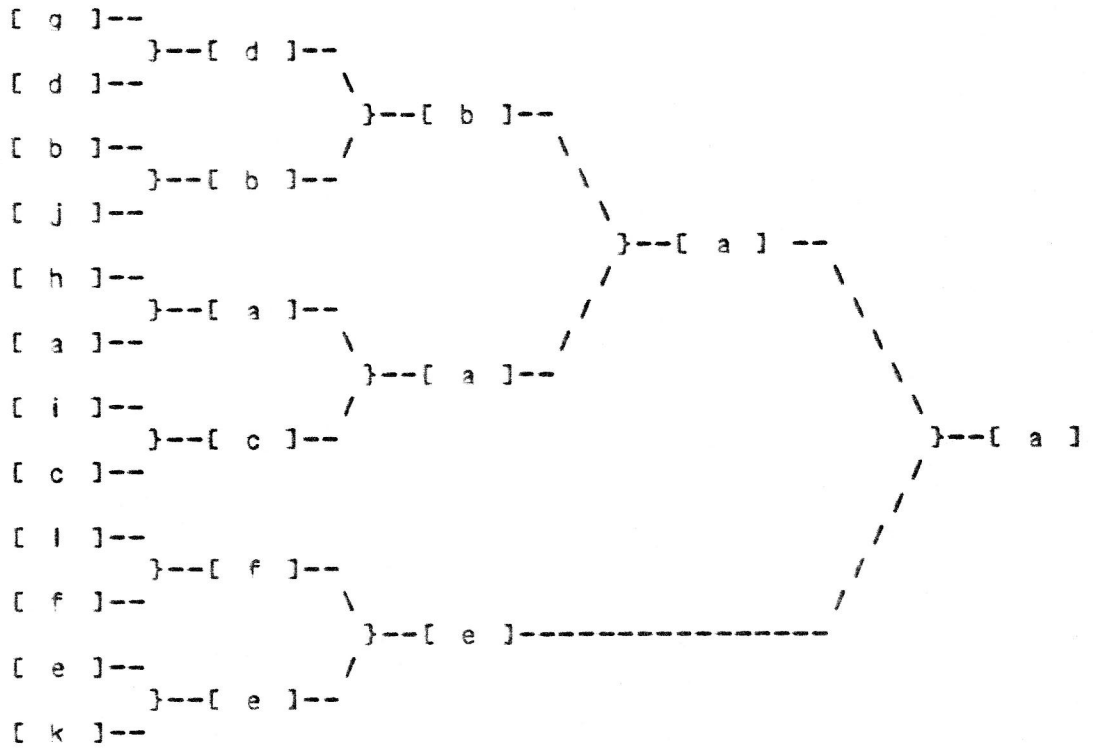

Figure 2

When the winning record of a pair advances to the next level we do not want to actually move the entire record. To save time we use indexes to records in the tournament and keep most of each record in a separate record storage area, RSA. This tournament is shown in Figure 3a and the Record Storage Area is shown in Figure 3b. For example, "h(5)" in the tournament means this has a comparison value of "h" and the rest of the record is in slot 5 of the RSA.

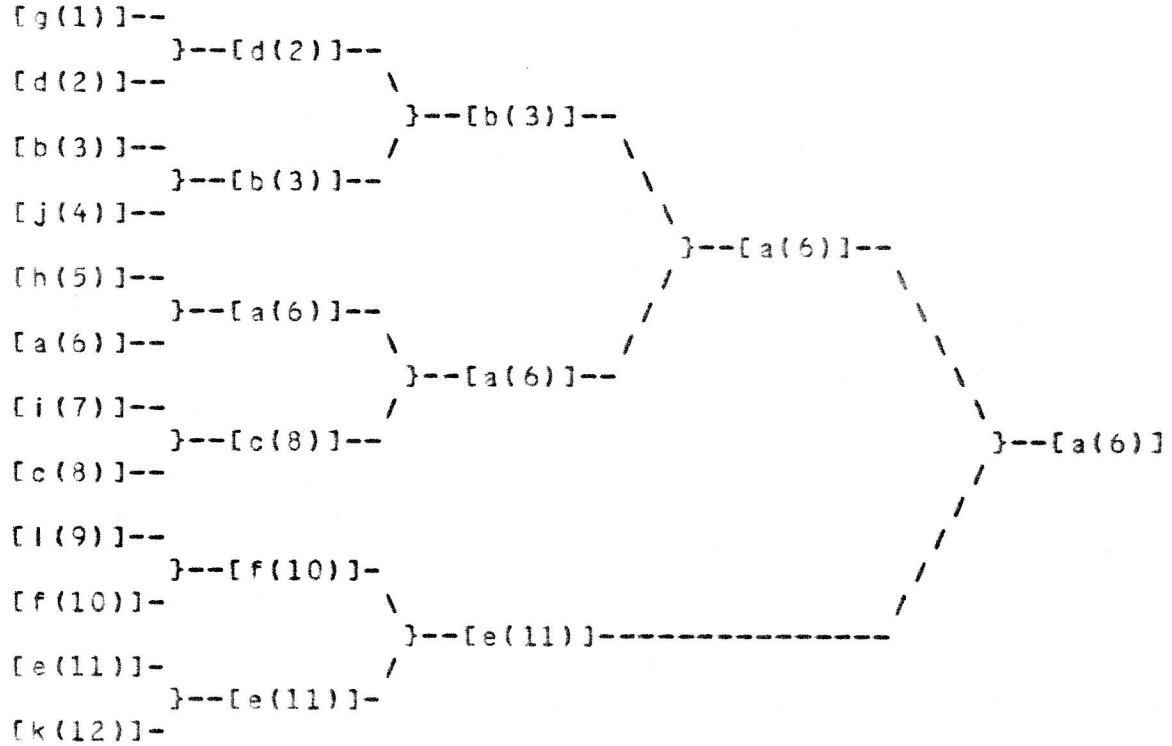

Figure 3a

```
 1: rest of record g
 2: rest of record d
 3: rest of record b
 4: rest of record j
 5: rest of record h
...
11: rest of record e
12: rest of record k
```

Figure 3b

Let us try to reorganize the tournament to save space. It is clearly wasteful to use more than one box for each winner. If we keep track of only the last winners' boxes we get figure 4. This tournament may look more confusing than Figure 3a but it contains the same information while using less space.

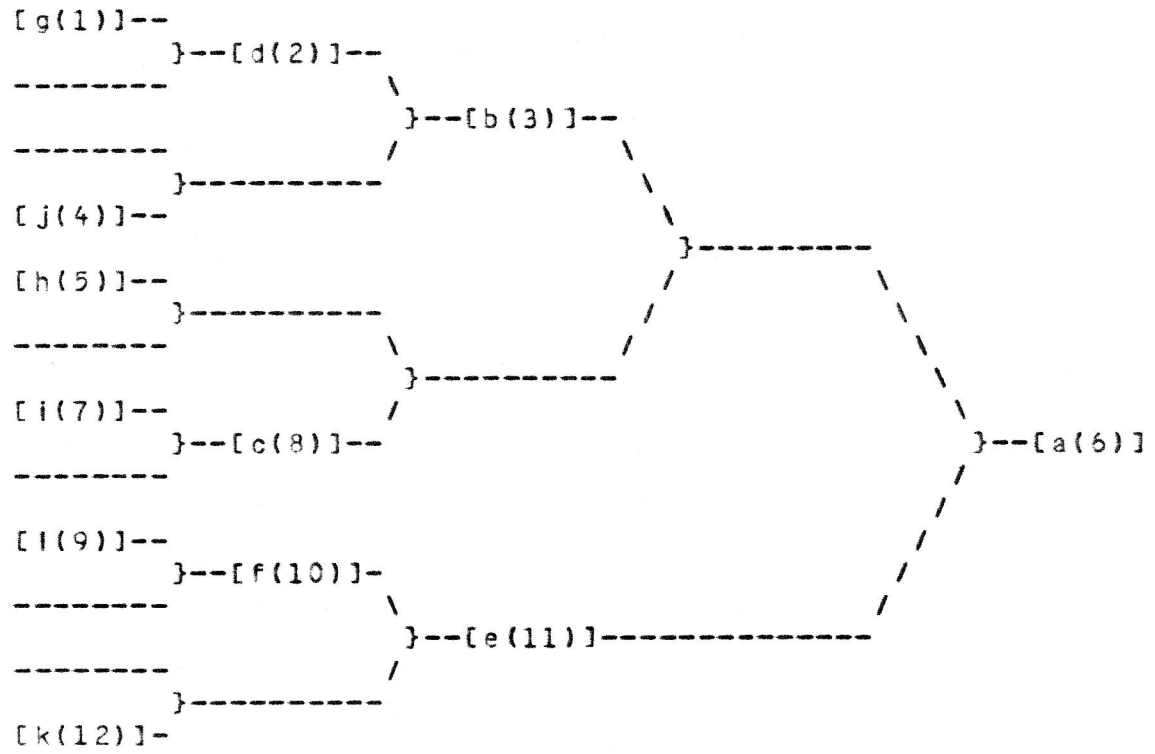

Figure 4

Now each box holds the loser of a competition. This is its only real significance. It does not matter now whether the loser came from the upper branch or the lower branch. So let us put each box in the middle of each pair. This gives Figure 5 which is the final form of the tournament. The boxes are numbered in this figure the same way Sort 5 numbers them. We will discuss the significance of this numbering system later.

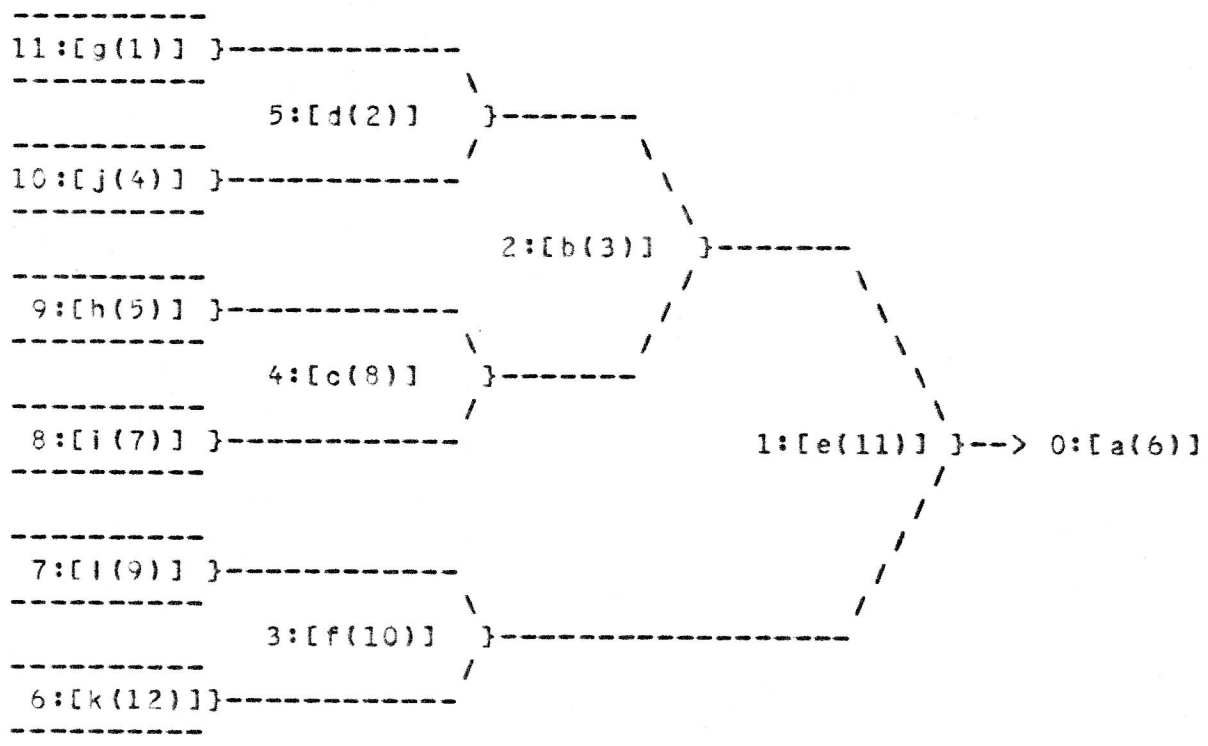

Figure 5

The tournament shows record a(6) won. So we write record a (from slot 6 in the RSA) to a string file. We then get a replacement record from the user and put it in slot 6 of the RSA. Suppose this record is m(6). We know the next winner is the replacement record or one of the records that lost to the current record: m(6), h(5), c(8), b(3) or e(11). We also want to put in the box of the next winner an appropriate loser so the tournament is ready for a subsequent winner. We do this by holding the replacement record, m(6), in our hand and comparing it with the first level loser, h(5). H(5) wins so we leave m(6) in box 9 and hold h(5). We next compare h(5) against the second-level loser, c(8). C(8) wins so we leave h(5) in box 4 and hold c(8). We next compare c(8) against the third-level loser, b(3). B(3) wins so we leave c(8) in box 4 and hold b(3). We next compare b(3) against the fourth-level

loser, e(11). This time the winner is b(3) so we leave e(11) in box 1 and continue to hold b(3). We have compared all appropriate records, so the winner is b(3) and the structure of the tournament is the same as we started. Figure 6 shows how each record has been shuffled after this last step.

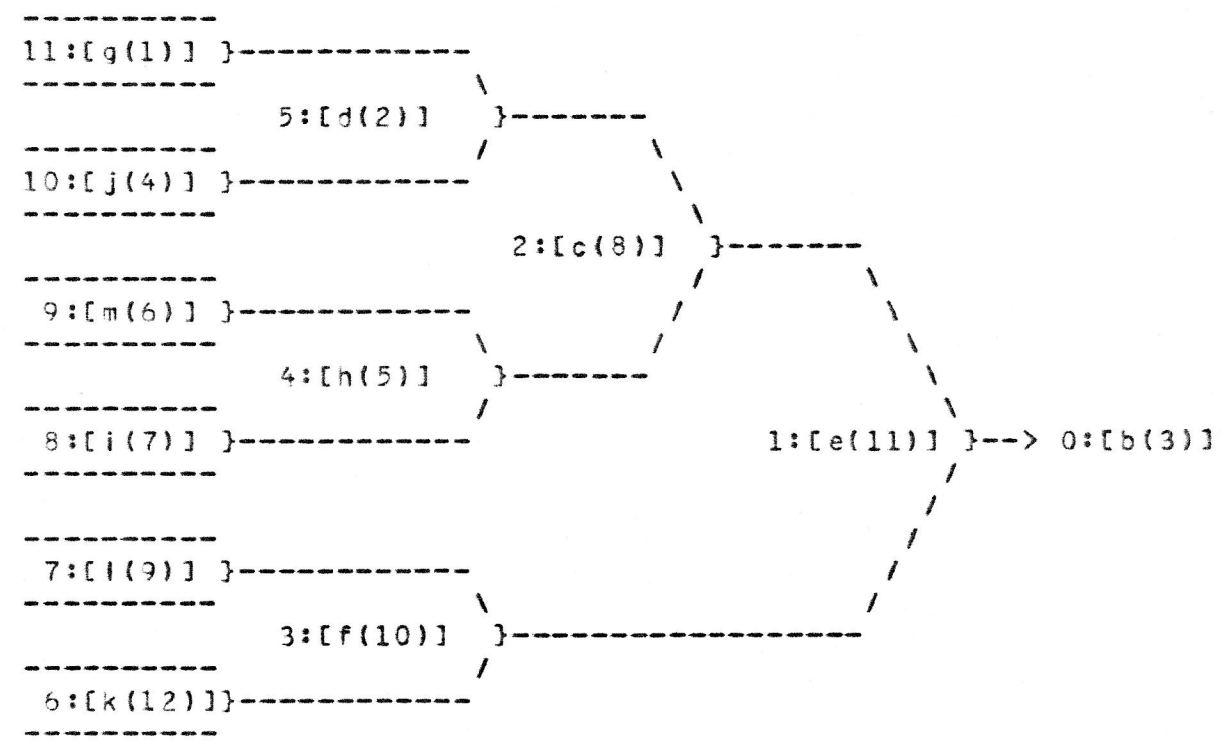

Figure 6

Suppose the next replacement record is aa(3). This record will never be able to fit on the current string because it should go before a record, b, that has already been written out. It is marked as a "next-string" record. It competes normally with other records except that a "next-string" record always loses to a normal "current-string" record. Eventually the tournament will fill up with "next-string" records and a "next-string" record will win the tournament. At that point all the "next-string" records will be considered "current-string" records and the process will continue.

Eventually we will run out of replacement records. Then we introduce "end-of-data" records. These also compete, with any normal record winning over an "end-of-data" record. The process continues until an "end-of-data" record wins the tournament. The sorting phase is then complete. It will have produced one or more sorted strings, which will then be merged and eventually be given back to the user.

At this point you can see the logic behind the numbers assigned to each box by Sort 5. The sort process goes from a box at one level to the box on the next level by dividing the first box number by two. Also, the first-level box number can be determined from the winning record number by the formula

<tournament length> - (<record number> + 1)/2

For example the box that a(6) originally competed against was 12 - (6 + 1)/2 = 9, which is where its replacement record will start competing.

I will now talk about some other aspects of Sort 5 implementation.

Sort 5 is maintained on an UPDATE program library. The project followed specific naming and ordering conventions to make it easy to find their way around the PL. Decks are grouped into sections consisting of a comment deck, one or more normal decks and an optional end-of-record deck. The comment deck is just a parenthesized name such as "(TEXT)". Two or more normal decks are ordered alphabetically, as befits a sort product! The end-of-record is alone in a deck named "*WEORn". The above description applies for comdecks as well as decks. Comdecks named abc$ contain all definitions for symbols named abc$xyz, making it easy to find these definitions. Figure 7 is a brief description of the various sections of the PL. Problems in the PROCS or TOOLS sections may not be PSR'd.

COPYRIGHT	copyright notice
*HELP	introduction to internals
(COMDKS)...	ordinary comdecks
(COMDKS2)...	comdecks that call other comdecks
(TEXT)...	text used in building Sort 5
(SKELGEN)...	routine to generate skeleton
(SORT5)...	routine for SORT5 (0,0) overlay
(MERGE)...	routine for MERGE (0,0) overlay
(SM5SORT)...	SM5 routine for SYSLIB
(SM5MERG)...	SM5 routine for SYSLIB
(SYMPL)...	miscellaneous SYMPL routines
(COMPASS)...	miscellaneous COMPASS routines
(PROCS)...	CCL procedures for development
(TOOLS)...	jobs to create tools: * Check for empty record * List common decks from PL * Append needed UPDATE directives * Generate report from performance file

Figure 7

Executable Sort 5 consists of overlays, relocatables and capsules. Comdeck SKELETON describes the exact content of each overlay and capsule, and which routines are relocatable. Route SKELGEN is assembled with SKELETON. When executed, SKELGEN produces a skeleton for all overlays, relocatables and capsules. Standard product COPYLM is used to replace the skeleton routines with the actual routines, producing a file easily built into the final library and overlays.

ST5TEXT is a COMPASS text used only in building Sort 5. ST5TEXT defines macros for structured programming, code generation, conditional assembly, understanding SYMPL comdecks and generating PASSLOC tables. Although ST5TEXT is created with IPTEXT it only uses IPTEXT to determine which operating system it is being built on, and thereby define the IFNOS and IFNOSBE macros accordingly. These macros are used to conditionally assemble code. Other macros are used to conditionally assemble debug code, memory mapping code and performance measurement code. Macros IFTHEN, ANDIF, ELSE- and ENDIF. are used to conditionally execute code. Many subroutines using these macros are programmed entirely without labels, making it easier to understand them. These and the other macros are fully explained in ST5TEXT.

This concludes our discussion of the implementation of Sort 5.

NOS V2 TAF FEATURES

TAF FEATURES FOR NOS VERSION 2

I. INTRODUCTION

This article describes the new TAF features for NOS version 2. It is intended to be a brief overview of the external as well as internal changes to the transaction subsystem. Reading this overview will generate many thoughts and questions. The TAF and TAF/CRM reference manuals should be consulted as they contain much more detail concerning application programming as well as information important to the system programmer and/or data base administrator.

II. TAF AUTOMATIC RECOVERY

Introduction

TAF Automatic Recovery provides for detection of failures, confinement of failures, and recovery to a correct and consistent state. To confine failures and establish a point of correct and consistent information, the recovery design uses the idea of commitment units. An implicit "begin" starts the commitment unit and an explicit "commit" ends the commitment unit. The implicit "begin" occurs when initial input is received, and the explicit "commit" occurs at the end of the transaction via a new task request called SECURE. All changes bounded by "begin" and "commit" are treated as one change. Either all changes are complete ("commit" executed) or all changes are rolled back ("commit" failed).

A recoverable transaction is defined via a new LIBTASK directive. This directive defines the tasks that make up the recoverable transaction as well as the data manager (TAF/CRM or CDCS) to be used by this transaction. A recoverable transaction can be thought of as a commitment unit.

A new file called the Communication Recovery File (CRF) is used to contain the information needed for recovery. This file is a preallocated direct access file and contains terminal/transaction status and terminal input/output. There are four sources for the data (messages) that is kept on the CRF:

1. Initial terminal input.
2. Task issued SECURE request.
3. Task issued RPUT request.
4. System task issued WSTAT request.

Thus, for correct implementation at the application level of TAF Automatic Recovery, recoverable transactions must be defined and, they must contain a SECURE request. This will allow all the necessary information for recovery to be logged on the Communication Recovery File by the transaction executive. Additional application level requests are available to facilitate easy recognition within a transaction of the exact point of failure.

General Processing Flow

A new control statement, TAFREC, has been added to the procedure file that initializes TAF. TAFREC performs the following initialization and recovery processes for TAF:

1. Creates/processes the CRF's as designated via the K-display command K.INT from information contained on the TAF

Configuration File (TCF) and the network files (NCTFi).
2. Determines which users were active and sets the recovery flag in the Terminal Status Table (TST).
3. Reformats the CRF's if any of the following were changed:
   . Maximum size of recovery terminal messages
   . Maximum number of recovery terminal messages
   . Number of terminals
4. Prepares a recovery report containing the:
   . Contents of the TCF
   . Initial K-display
   . Status of terminals/users/jobs affected by recovery
   . Errors that prevent a correct and consistent recovery

After TAFREC completes, TAF1 executes as before, except that the initialization values and terminal recovery status are obtained from a file prepared by TAFREC instead of the K-display.

Recovery data for each recoverable transaction is saved on the CRF when the initial input is entered. This is done by TAF when processing the new request, CALLTRN, that ITASK now uses to initiate a transaction.

If a system failure occurred before the transaction completed, the recovery field would have been set in the TST. Upon terminal login a new system task, RTASK, would be called rather than ITASK. RTASK determines whether the transaction should be rerun. If so, RTASK calls another new system task, CTASK, to rerun the transaction.

New Task Requests

The following requests are for use within an application task.

- RERUN - Restarts a transaction using the initial terminal input
- RGET - Retrieves data from the CRF
- RPUT - Places data on the CRF
- RSECURE - Retrieves a message previously saved on the CRF by a SECURE request
- SECURE - Signals the completion of a transaction's work and saves a message on the CRF to be sent to the terminal when the transaction ceases
- TSTAT - Returns the status of one or more variables that change through out the life of a transaction

The following requests are intended for use within system tasks.

- CALLTRN - Initiates a transaction (this request can only be used by ITASK)
- SRERUN - Resubmits initial terminal input for a rerunnable transaction
- TINVOKE - Returns or specifies a system identifier (transaction sequence number) for begin-commit sequences in this job

. TMSG - Issues a message to the TAF dayfile
. WSTAT - Records information on the CRF needed for recovery processing

New System Tasks

The following new system tasks have to be present for the TAF Automatic Recovery feature:

BTASK	RCTASK
CTASK	RTASK

BTASK recovers rerunnable BTRAN (batch originated) transactions. BTASK reads the entire CRF using TSTAT requests to obtain information about each "BTRAN user". If the TAF/CRM current begin-commit sequence for each "BTRAN user" is not successful, BTASK will call CTASK to rerun this transaction.

CTASK handles the following recovery cases.

. Interactive users
. BTRAN users
. CRM data base
. Terminal failures

For both interactive and BTRAN users, CTASK issues a TINVOKE request to obtain a new transaction sequence number. Then, CTASK will issue an RSTDBI request (a new TAF/CRM request) to restore the begin identifiers to TAF/CRM for a recoverable transaction. An SRERUN request is issued to cause the initial input to be resubmitted to the transaction system.

To recover TAF/CRM data bases, TAF/CRM internal table information is obtained via the CRMSTAT request (another new TAF/CRM request). With this information CTASK issues WSTAT requests to record TAF/CRM data base begin-commit history on the CRF. CTASK will then roll back the TAF/CRM data bases by issuing TRMREC requests (yet another new TAF/CRM request).

Recovery from terminal failures is handled in a similar fashion. The transaction sequence number from CRF is checked against the information obtained via the CRMSTAT request. A WSTAT request is issued to record the TAF/CRM begin-commit history and, then, a TRMREC request is issued to roll back the data base.

RCTASK recovers rerunnable CDCS transactions. RCTASK reads the entire CRF by issuing TSTAT requests to obtain information about each user. For those users able to use CDCS and CDCS was down while they were active, an SRERUN request will be issued to rerun their transaction. This will occur if the user is still logged in when CDCS comes up again. BTRAN CDCS transactions will also be rerun by RCTASK.

RCTASK gets scheduled whenever CDCS becomes active.

Data base recovery for TAF/CDCS transactions is not implemented in this release. Only the interface to CDCS Automatic Recovery has been implemented for TAF transactions. The CDCS Automatic Recovery feature will be contained in a future release of NOS Version 2.

RTASK is scheduled at user login time when the recovery bit in the TST is set. RTASK recovers rerunnable transactions after terminal disconnects, network failures, TAF failures, and system failures. The process is as follows.

1. If a recoverable transaction is no longer rerunnable, RTASK informs the terminal operator that recovery has occurred, that the recoverable transaction will not be rerun automatically, and to enter the next input.
2. If the transaction is not recoverable, RTASK sends "ready" to indicate the terminal operator should enter the next input.
3. If the transaction is "secure" and the data base commit unit is committed, RTASK sends the "secure" message as recorded on the CRF.
4. Otherwise, RTASK calls CTASK to rerun this transaction using a new transaction sequence number.

TAF Configuration File and K-display Changes

Three new directives have been added to the TCF.

```
DISPLAY,status.
NETWORK,ID=i,FM=family,UN=username.
RECOVER,ID=i,MS=ms,NM=nm.
```

The DISPLAY directive controls whether or not the initialization K-display will be used. If status is specified as ON, the K-display will be selected. If OFF, the K-display will not be used, unless an error is present in one of the directives that may be in the TCF.

The NETWORK directives declare the network files (NCTFi) TAF will use in determining which users are allowed to access TAF. There may be from one to eight network files. "i" may be a number from 0 to 7. This file then resides under the family and user name specified on the FM and UN parameters.

The RECOVER directive declares which set of files TAF is to use in recovery mode. "i" is the recovery file id (CRFi). This id must match an id specified in a NETWORK directive. "ms" is the maximum message size in words that TAF will record on the CRF. "nm" is the maximum number of user messages that can be recorded on the CRF via the RPUT request.

In addition to these new directives all the K-display commands

may now be placed on the TCF. If DISPLAY,ON. is specified, the initial K-display will contain the values entered via the TCF.

Four new K-display commands are available to the operator to perform recovery related functions.

```
K.INT = ftp,id.
K.ERO = CRF,opt.
K.GO.
K.STOP.
```

The K.INT command defines which TAF recovery file(s) are to be initialized. This is the only means of initializing a CRF. Those files specified on a RECOVER statement and selected via the K.INT command will be initialized. Those files specified on a RECOVER statement and not selected via the K.INT command will be used for recovery. The K.INT command also defines whether TAF/CRM data base recovery files are to be initialized or recovered. "ftp" is the file type, either CRF or CRM and "id" can be specified as a numeral from 0 to 7 or ALL or NONE.

The K.ERO command defines whether to override certain i/o or logical errors encountered during procesing of the TAF recovery files (CRF's).

The K.STOP command aborts TAFREC and the K.GO command is identical to the K.END command.

New TAF Installation Parameters

TAF Automatic Recovery has introduced three new installation parameters, IPTAR, IPTST, MAXMS, and NTSB. The first three parameters as well as most of the previous installation parameters are defined in a new common deck, COMKIPR. The fourth is defined in deck, TAF.

IPTAR enables or disables the TAF Automatic Recovery feature. If set equal to zero, TAF Automatic Recovery is disabled and the following requests will not function in a recovery mode.

TINVOKE	SECURE
TSTAT	RSECURE
RERUN	WSTAT
RPUT	CALLTRN
RGET	

Setting IPTAR equal to one enables the TAF Automatic Recovery feature. Default is enabled.

IPTST sets the maximum number of terminals in all active network files. Default is 500 and the maximum value this parameter may be set to is 4095.

MAXMS sets the maximum number of words for the RPUT and RGET requests. Default is equal to installation parameter MAXWS (410D).

NTSB sets the number of entries for a new internal table within TAF called TAF's Work Table. This table is used to hold information for which work is queued. The default value is 40.

Compatibility Issues

1. A new control statement, TAFREC, must be added to TAF's initialization procedure file. This control statement must appear in the procedure file even if TAF Automatic Recovery is not enabled.

2. TAF can now use from one to eight network files. The name of the files, NCTFn (n = 0,1,...,7), and the user name and family where they reside is specified on the NETWORK directive in the TCF. At least one NETWORK directive is needed in the TCF.

3. LIBTASK can now define a transaction which may have the attribute of being recoverable. The directive specifies the first task to be executed in the transaction and allows specification of up to four additional tasks which may be part of the transaction, to be executed one after the other.

   A task may be part of more than one transaction and system tasks may be included in the transaction definition.

4. Initial terminal input results in the execution of a named transaction as follows:

   . If the first three characters of input are recognized by ITASK as a transaction code which is associated with a transaction, then ITASK will attempt to schedule the transaction. Transaction codes can be associated with the transaction through the STRAN macro in ITASK.

   . If no transaction was associated with the transaction code, then ITASK will use as a transaction name the first seven characters of input.

   In either case, if the transaction name does not exist (either associated with the user task library or the system task library), then an error message will be issued via MSABT.

5. BTRAN request changes:

   . The user name parameter is no longer specified in the BTRAN header word. Rather the user name with which the request is associated is the user name of the submitting batch job.

- The user validation required to use this request has changed from the CSTP bit to the CUCP bit in the access word.

- Status values returned to the user have changed slightly-some new ones have been added, and status 11B has been deleted.

- A BTRAN initiated transaction cannot run under a user name assigned to another batch or transaction terminal user.

- A SEND to the originating user name is not allowed in a BTRAN initiated transaction.

- A SEND is no longer required of a BTRAN initiated transaction.

6. Three new libraries must be built, called TRANC5, TRANF4, and TRANF5. These provide the language dependent (COBOL 5, FTN 4, FTN 5, respectively) interface routines for the recovery requests.

7. The new system tasks must be added to TASKLIB.

8. The following LIBTASK input directive is needed when building TASKLIB, if XTASK will be used to initiate transactions:

```
/XTRAN,T1=XTASK.
```

9. The programs, TAFNAM1 and TAFNAM2, are now named, TAF1 and TAF2, respectively.

III. TAF/CRM AUTOMATIC RECOVERY

Introduction

TAF/CRM Automatic Recovery maintains data bases in a consistent state across failures. It does this by using the idea of a begin-commit sequence (much like TAF Automatic Recovery) and ensuring that either all or none of the data base updates within a begin-commit sequence are performed. Recovery can be selected on a file by file basis. Within a single TAF/CRM data base some files may be recoverable while others are not.

Recoverable files are indicated by the recovery parameter on the CRM statement in the xxJ file. However, a data base may have recoverable files only if there is one or more before image recovery files (BRF) specified for that data base. This is done by including a BRF,n. statement in the xxJ file, with n being greater than zero.

To support automatic recovery, before images of modified records are saved on a before image recovery file (BRF). To support batch recovery, after images of modified records are saved on an after image recovery file (ARF).

No before or after images are kept for non-recoverable files, so no recovery is supported, even if the non-recoverable file is modified inside a begin-commit sequence.

General Processing Flow

During TAF's initialization, the TAF/CRM recovery files may or may not be initialized via the K.INT K-display command. If initialized, all recovery information on all BRF's and ARF's will be lost. If not initialized, TAF will initialize in a recovery mode.

In recovery mode, TAF/CRM will reconstruct all internal tables from information on the BRF(s) concerning abnormally terminated recoverable tasks. This involves making sure the operational environment (which files open, records locked, etc) is reset for the TAF/CRM recovery task, CTASK. CTASK is initiated at the end of TAF's initialization to recover TAF/CRM data bases (see the description of CTASK in section II).

When a user, who was running a recoverable transaction logs back in, TAF will schedule RTASK to determine whether the transaction can be rerun. If so, RTASK will call CTASK to obtain a new transaction sequence number (TINVOKE request), to restore the begin-id's (RSTDBI request) and restart the transaction. A task in the recovered transaction may then issue a DBSTAT request to identify

the point of failure and continue processing.

All requests that modify recoverable data base files (DELETE, REWRITE,WRITE) must be made from within a begin-commit sequence. Processing these requests involve the following steps.

1. TAF/CRM checks to see if a DBEGIN request had been issued, if not, an error code is returned.

2. TAF/CRM will lock the record if not already locked.

3. The before image of the record is written to the BRF (an empty before image is written on a WRITE).

4. DELETE/REWRITE/WRITE the record.

5. Write the after image of the record to the ARF (an empty after image is written on a DELETE).

External Changes

An additional parameter has been added to the various TAF/CRM read requests (READ,READN,READM). This parameter will tell TAF/CRM to return the lock status of the record being read.

The xxJ file has an additional statement specifying the number of before image recovery files associated with this data base. The format of the new statement is:

BRF,n.

Where n is a number from 0 to 64 (see installation parameter, BMAX). This statement is required for all TAF/CRM data bases even if they contain no recoverable files.

Two additional parameters have been added to the CRM statement in the xxJ file, also. There is a parameter stating whether the file described in this CRM statement is recoverable or not recoverable. The two values are either R or N.

R - Recoverable file
N - Non-recoverable file

The other parameter is a forced write indicator. If specified as "on" for this file , all file buffers which are modified in core are written to disk immediately. If specified as "off", the modified file buffers are written to disk when space is needed. The two values are Y or N.

Y - Forced write is set "on" for this file
N - Forced write is set "off" for this file

New Task Requests

To support TAF/CRM Automatic Recovery, application tasks must update their data base files from within a begin-commit sequence. The following application task requests are used to easily facilitate recovery from within the task.

- DBEGIN - Designate the start of a begin-commit sequence.
- DBCOMIT - Designate the end of a begin-commit sequence.
- DBFREE - Terminates a begin-commit sequence and cancels all updates made to recoverable files since the preceding DBEGIN request.
- DBSTAT - Returns the begin-commit identifiers for the current begin-commit sequence and the most recent successfully completed begin-commit sequence.

The following requests are intended for use within system tasks. They are only available to system tasks written in compass.

- DBUP - Places a data base or data base file in "up" status. TAF/CRM will attach the file(s) and make them available for use by the application.
- DBDOWN - Places a data base or data base file in "down" status. TAF/CRM will return the file(s) when there are no more outstanding requests. The file(s) will be unavailable for use by the application.
- CRMSTAT - Returns various TAF/CRM internal table information.
- TRMREC - Causes a DBFREE to be executed, a CEASE stamp to be written on the BRF, and all resources held by the task to be released.
- CRMSIC - Passes information from TAF to TAF/CRM about a batch recovery job that has completed successfully.
- RSTDBI - Restores the data base begin identifiers for a recovered transaction.

TAF/CRM Interface To Batch Recovery

TAF/CRM will submit a batch job containing a call to the batch recovery utility program, DMREC, when any of the following situations occur.

- An after image recovery file must be dumped to tape.
- TAF/CRM detects a defective data base file.
- TAF/CRM detects a defective before image recovery file.

This batch job will run under the username, password, and family of the owner of the file being dumped or recovered. The utility,

DMREC, will send a reply to TAF via an SIC call. TAF will initiate a system task, CRMTASK, to process the response from the batch job. CRMTASK uses the CRMSIC request to notify TAF/CRM of the successful completion of a batch recovery job.

New System Task - CRMTASK

CRMTASK is a task that enables the data base administrator to determine and/or change the status of all TAF/CRM data bases or any particular data base or data base file. Only those files and data bases described to TAF in the xxJ files can be specified on the K-display commands that initiate CRMTASK. This task is also used by TAF to inform TAF/CRM of responses from batch recovery jobs. CRMTASK commands can also be made available to terminal users. To accomplish this, a transaction must be defined with CRMTASK as the only task. CRMTASK is initiated by the K-display command, K.DIS,CRMTASK. When the K-display is assigned, the following commands are available.

- K.CRMSTAT.
- K.CRMSTAT,xx.
- K.CRMSTAT,xxpfn.
- K.DBUP,xx.
- K.DBUP,xxpfn.
- K.DBDOWN,xx.
- K.DBDOWN,xxpfn.
- K.END.
- K.MENU.

The first command displays status of all data bases. Parameters xx and xxpfn specify specific data bases and data base file names. K.END. will terminate CRMTASK and K.MENU. will display the various commands available.

New Installation Parameters

The following are new installation parameters related to the TAF/CRM Automatic Recovery feature. They can be found in deck, COMKIPR.

- CRMARB - The maximum number of after image records which will be buffered in a central memory buffer for a file before being written to disk. Default is 15.
- CRMARFN - The length of the after image recovery files in pru's. Default is 35000.
- CRMUPM - The maximum number of updates allowed from within a single begin-commit sequence. Default is 15.
- BMAX - The maximum number of before image recovery files per data base. Default is 8.
- RMDM - The maximum number of mainframes running TAF/CRM.

Needs to be greater than one if this mainframe may be used to recover TAF/CRM data bases from one or all of the other mainframes. Default is one.

IV. TAF/CRM BATCH RECOVERY

Introduction

TAF/CRM Batch Recovery provides a data base administrator with recovery capabilities beyond those provided by TAF/CRM Automatic Recovery. It is used to recover defective data base files, dump and load data base files and alter image recovery files, preallocate data base files, and maintain a directory of data base file and recovery file dump tapes.

The basic mechanism in the design of TAF/CRM Batch Recovery is the complete control over its library of backup tapes in order to recover with as little operator intervention as possible. To do this, a utility program called DMREC creates and maintains a file backup directory, ZZdbDIR (db is the data base name). This file is a direct access index sequential file resident under the same user name and family as the data base files. It contains information about all data base files, which are dumped to backup tapes and the after image log tapes currently active.

DMREC will process on a data base basis. This means that there will be one file backup directory per data base and DMREC, during any one execution, will know only about the files and backup tapes for this data base.

DMREC will process both data base file dumps and after image recovery file dumps. After image recovery files are written during normal TAF/CRM processing. This mass storage file will be dumped to tape via a TAF/CRM initiated batch (DMREC) job when it becomes full.

All backup tapes managed by DMREC will be labeled and, in order to provide data security, only accessible thru DMREC under the creator's user name.

DMREC Processes

The following is a list of directives that cause DMREC to perform specific recovery related functions.

- CREATE - Generates initial copies of recovery files (ARF's or BRF's).
- DUMP - Dumps a data base file within a TAF/CRM data base or dumps an after image recovery file (ARF) onto a backup tape.
- EDIT - Edits the backup directory (ZZdbDIR). This directive performs cleanup processing on the backup directory. VSN's of dump tapes can be added or deleted from

the directory. This directive may also be used to discard inactive dump tapes.

- EXPAND - Preallocates a data base file by a given percentage.
- LIST - Lists the contents of the backup directory.
- LOAD - Loads a data base file from a backup tape to mass storage.
- RECOVER - Recovers a data base file within a TAF/CRM data base. This directive is a combination of the LOAD and UPDATE directives.
- UPDATE - Applies updates (after images) from the ARF dump tapes to a given data base file.

During the UPDATE process (with subdirective IGNORE), the user is able to direct DMREC to ignore updates done by certain transactions (using the transaction sequence number) or by certain tasks.

Installation Parameters

The following installation parameters can be found in deck, COMKIPR.

- DTTP - Dump device type. Default is nine track.
- TDEN - Dump device density. Default is the system default.

The following installation parameters can be found in deck, DMREC.

- NUMARF - Number of copies of the same ARF dump. Default is one.
- NDUMP - Maximum number of file names specified on one dump directive. Default is 100.
- EXPCT - Default expand percentage. Default is 10 percent.
- NCOPY - Default number of dump copies to keep. This value is used during the EDIT process if not changed via the CYCLE subdirective. Default is 2.
- TTIGL - Maximum number of task names and transactions that can be specified via IGNORE subdirectives. Default is 5000.
- TLOGL - Maximum number of files in the data base. Default is 100.
- TVSNL - Maximum number of vsn's that can be used during this execution of DMREC.
- WBUFL - Size of DMREC's internal working buffer. This buffer is used to contain the data read in from a block on the ARF dump tape or ARF. The size of the block is dependent on the MRL specified on the data base files in the xxJ and on installation parameters, CMDM and CRMARB.
- TDTR - Tape Format Definition. Default is labeled, nine track.
- AAICL - Maximum number of IGNORE entries. This is the table constructed after determining whether the

transaction's or task's after images fall in a specified date/time window. Default is 200.

. FTABL - Maximum size of the intermediate table used to process the IGNORE directives. Default size is 5000 entries.

Compatibility Issues

To use the TAF/CRM Batch Recovery feature, the user specified in the xxJ file must be given read permission to the xxJ file. This file still resides under TAF's user name, but DMREC executes under the user name as specified in the xxJ file and must be able to get a copy of the xxJ file.

V. TAF/CRM MIP/AK SUPPORT

This feature implements both multiple index processing (MIP) and actual key (AK) capabilities for TAF/CRM users. As with index sequential (IS) and direct access (DA) files under TAF/CRM, the the MIP and AK related files must be created in a batch environment.

For MIP'd files, a TAF/CRM user specifies an alternate key using the key-identifier parameter on an appropriate request. The key-identifier is the ordinal of the alternate key description statement (AKY) in the xxJ file. An alternate key can be deleted from the index file by using the MIPGEN utility. To maintain the correct ordinals for the other alternate keys, the data base administrator must define a deleted alternate key in the xxJ file.

External changes for the feature include the following.

- AK is now a valid file organization for TAF/CRM data base files and can be specifed on the CRM statement in the xxJ file.

- Two new statements must be included in the xxJ file to support MIP'd files. The IXN statement describes the index file and the number of alternate keys to TAF/CRM. The AKY statement describes the various alternate keys for MIP'd files.

- A new task request called START allows postioning of a file at or after a given key. It performs the same function as the CRM macro, START.

- Optional parameters have been added to the various read requests in order to fully take advantage of multiple index processing.

VI. TAF/CRM BATCH CONCURRENCY

Introduction

TAF/CRM Batch Concurrency provides a mechanism whereby a batch job may access files currently assigned to TAF. This batch access to TAF/CRM data base files is identical in capabilities and interface to that which exists for a transaction under TAF/CRM. All of the TAF/CRM Automatic Recovery features are also supported by this new feature.

A series of requests is available to the batch user wishing to interface to TAF/CRM data base files. These requests are identical in format to those used in TAF transactions. Each request is satisfied via a call to object time routines located on a new library, BCLIB (Batch Concurrency Library).

TAF/CRM Batch Concurrency is available to all users who are validated to use TAF interactively, and who have the SCP/UCP permission (CUCP bit) specified in their access word. As with interactive transactions, only one batch job can be active under a given user name at a given time. Also, an interactive transaction and a batch job cannot be running simultaneously under the same user name.

Data Flow

The basic data flow of this feature is the passing of TAF/CRM data base records between a batch job and TAF. The batch job (a UCP) will communicate with TAF (a SCP) by issuing TAF/CRM data manager requests. These requests will cause the execution of object time routines, which have been loaded into the users field length. The routines will establish the UCP/SCP connection to TAF via the CALLSS macro, pass data in request packets, and receive data from TAF/CRM.

The TAF executive will be monitoring words RA+50B (RA.SSID) and RA+51B (RA.SSC) looking for requests from batch jobs. Upon receipt of a request, TAF will validate each request both for content and access permission. In the case of an initial request, TAF will create table entries for the batch job, assign it a transaction sequence number, and issue a CALLSF request to establish a long-term connection with a batch job. When a connection is established, TAF will place the request in the TAF/CRM input queue for processing.

Internal Changes/Installation Parameters

A new table has been added to TAF. This Batch Communication Table

(BCT) will enable TAF to keep track of batch jobs with requests for TAF/CRM. The table is built at initialization time and the number of entries is based on the installation defined option, TBCON.

Subcontrol point areas are utilized by batch concurrency to hold requests for TAF/CRM. The initial request by a batch job to TAF results in job validation and assignment of a subcontrol point. When assigned, the subcontrol point will be large enough to contain the argument array, the TAF/CRM parameters passed by the batch job, and a buffer to contain the largest possible record. The subcontrol point area is associated with a particular batch job until that job terminates or aborts.

The installation option, TBCON, specifies the maximum number of active batch jobs with concurrent access to TAF/CRM. The option is placed in the TAF Configuration File (TCF) as a directive. The run-time K-display command, K.DBCON,n., can be used to alter the maximum number of concurrent batch TAF/CRM users within the limits specified by the TBCON directive in the TCF.

NOS V2

SHARED ROTATING MASS STORAGE

Introduction

This article describes the NOS Version 2 feature known as Shared Mass Storage. The sharing between mainframes of mass storage devices has been a feature of NOS Version 1 for some time. In order to share devices, however, all mainframes accessing shared devices were required to be connected to a common ECS or ESM device. ECS and ESM (Extended Core Storage and Extended Semiconductor Memory) are functionally equivalent forms of extended memory and are available as options on most Cyber mainframes. NOS Version 2 Shared Mass Storage is a new approach to sharing devices which removes the extended memory requirement and adds new capabilities. In order to distinguish between the old and the new method of sharing devices, the old method will be referred to as ECS Multi-Mainframe or ECS MMF. The new method will be referred to as Shared Mass Storage. Devices shared under Shared Mass Storage will be referred to as independent shared devices because they do not require a common link device.

1.1 History

The Shared Mass Storage software was originally written by CDC as the result of a Quote for Special Software (QSS). The package was written to be compatible with NOS Version 1.4 (PSR Level 518). It was incorporated as a standard feature available in the level 552 release of NOS 1.4.

The Shared Mass Storage package which appears in NOS Version 2 is basically this same package, but it has undergone substantial modification. Some modifications were necessary to make the package compatible with the changes made to the rest of NOS for Version 2. Other modifications were in the interest of performance and feature enhancements.

This article presents the Shared Mass Storage feature to an audience which presumably has no familiarity with the NOS 1.4 version of Shared Mass Storage. The feature will be presented as it changes a system which supports only ECS MMF operations. For those sites that intend to run NOS Version 2, there is minimal advantage in installing the NOS Version 1 Shared Mass Storage as an intermediate step.

2 Overview

The purpose of the Shared Mass Storage feature is to eliminate some of the restrictions imposed by the ECS MMF method of sharing mass storage devices. These restrictions are:

1. All mainframes sharing mass storage devices must be connected to a common ECS or ESM device.

2. The maximum number of mainframes in a single MMF complex is limited to four.

3. Only mainframes which support external extended memory may share mass storage devices. This excludes the Cyber 176 and the Cyber 835 for example.

Shared Mass Storage removes these restrictions in the following way:

1. Extended memory is not required in the multi-mainframe complex.

2. There need be no device which is common to all mainframes in the complex.

3. Internal tables will allow up to 16 mainframes to simultaneously access a single device. The total number of mainframes in a multi-mainframe complex may, however, be greater than 16.

4. All Cyber 170, Cyber 70, 6000 and 800 series mainframes which can run NOS may now access common permanent files and data bases in a multi-mainframe environment. The supported shared devices under Shared Mass Storage are the 844 and the 885 disks.

Shared Mass Storage is able to remove the ECS requirement by maintaining the tables necessary for device sharing on each independent shared device. There is no longer a global device accessible by all mainframes within a complex. The following differences will be noticed when comparing ECS MMF to Shared Mass Storage:

1. Removing ECS as a central device in multi-mainframe operations decreases the performance which was achieved by using ECS tables to interlock shared devices. Shared Mass Storage relies on the device hardware interlock and a new software interlock. Tables must be rewritten to disk each time they are updated. This process is considerably more time consuming than the updating of tables in ECS with periodic checkpointing.

2. Global fast attach files are not accessed as quickly as they are with either ECS MMF or stand alone operations. The system sector of the file must be used to control access by each mainframe.

3. Shared Mass Storage only supports the 844 and 885 disks as shared devices. ECS may not be shared.

4. ECS MMF currently allows Mass Storage Subsystem (MSS) files to be accessible to all mainframes in an MMF complex through a master-slave communication scheme between the mainframes. Shared Mass Storage does not support this scheme. Only the mainframe physically accessing the Mass Storage Facility has access to MSS files.

5. On-line reconfiguration of shared devices is not supported by Shared Mass Storage.

2.1 Site Impact

Sites not using this feature will notice no impact. ECS MMF and stand alone modes will continue to function as before. The impact to sites choosing to run Shared Mass Storage is mostly in the area of system operation. Computer users should only be aware that files are now available from a number of mainframes. The site operations will be affected with the following considerations:

1. Mass storage allocation must be configured to minimize the negative performance impact of maintaining certain file types on shared devices. The following file types should not be assigned to independent shared devices:

   . Any file which is local to only one mainframe, such as temporary or rollout files.

   . Fast attach files which are non-global such the resource executive files. If the default family is shared, these files will normally reside on shared devices. The following procedure may be saved as SYSPROC under the System user index (377777) to move these files at deadstart:

```
.PROC,SYSPROC.
.*
.* SYSTEM PROCEDURE WHICH RUNS
.* AT EVERY LEVEL 0 DEADSTART.
.*
```

```
IFE,HID=$AA$,MIDAA.
.*
.* NOS REPLACES THE *HID* SYMBOL WITH
.* THE VALUE SPECIFIED IN THE CMRDECK
.* *MID* ENTRY AT THE LAST DEADSTART.
.*
.* THE FOLLOWING EXAMPLE ASSUMES THAT
.* EQ01 IS A NON-SHARED DEVICE ON
.* BOTH MAINFRAMES AA AND AB.
.*
MVEPROC,AA,01.
ENDIF,MIDAA.
IFE,HID=$AB$,MIDAB.
MVEPROC,AB,01.
ENDIF,MIDAB.
.*
.* CALLS TO MVEPROC FOR OTHER
.* MAINFRAMES MAY BE INSERTED HERE.
.*
.* ADDITIONAL COMMANDS MAY ALSO
.* BE INSERTED IN SYSPROC.
.*
REVERT.
.DATA,MVEPROC
.PROC,MVEPROC,MID,NONSHDEV.
.*
.* THIS PROCEDURE MOVES THE RESEX FAST
.* ATTACH FILES TO DEVICE NONSHDEV.
.*
ISF,R=RSXV#_MID,SJ=0,SP=0.
PURGE,RSXV#_MID,RSXD#_MID/NA.
RETURN,RSXV#_MID,RSXD#_MID.
ASSIGN,NONSHDEV,RSXV#_MID.
ASSIGN,NONSHDEV,RSXD#_MID.
DEFINE,RSXV#_MID,RSXD#_MID.
RETURN,RSXV#_MID,RSXD#_MID.
ISF,SJ=0,SP=0.
REVERT.
```

- The System file.
- The System and Account dayfile, Error Log and Binary Maintenance Log.

2. The CMRDECK entry ISHARE has been added to identify independent shared devices.

3. The CMRDECK PRESET entry contains a list of devices to be preset if the machine is running in Shared Mass Storage mode.

4. The procedure for dealing with a downed machine using MREC has been modified.

5. Shared devices may not be reconfigured on line.

6. The DSD MOUNT command contains an optional *P* parameter to indicate the presetting of an independent shared device.

7. New operator messages and appropriate responses have been documented.

3 Design Notes

3.1 Device Index Table

A Device Index Table (DIT) is present on each independent shared device. This table determines the index each machine will use when accessing tables on the device. The index is used as an offset into those tables that have an entry per mainframe. This index is called the machine index and is determined separately for each independent shared device. When an independent shared device is first associated with a mainframe, the DIT on that device is searched for an available entry. The Machine Identification (MID) of this machine, a two character value which is unique within the complex, is written into that slot in the DIT. The ordinal of the slot determines the machine index that will be used by this machine on that device. The process is repeated for all independent shared devices associated with this machine. A machine may be assigned a different index for each independent shared device. A site must insure that MIDs are indeed unique within their complex since there is no way for for NOS to verify this automatically.

Note that the machine mask, which is used in ECS MMF operations, is not used by the Shared Mass Storage software.

3.2 PP Resident

PP resident routine FTN has been restructured and two new programs, 1RU and 1FA, have been provided to assist FTN in processing certain monitor functions.

FTN checks the function that is to be processed. If the function involves access to tables on an independent shared device, FTN will load program 1RU or 1FA to continue the processing for that function. Program 1RU handles the special processing needed to reserve a unit for functions involving access to tables on independent shared devices. Program 1FA handles the logic involved in accessing a fast attach file which resides on an independent shared device.

In order to simplify FTN's special processing check, the monitor functions requiring special processing have been renumbered and are now grouped together. This allows all these functions to be checked at once.
These functions are:

Function	Number	Description
AFAM	30	Access Fast Attach File
DLKM	31	Delink Track Chain
DTKM	32	Drop Tracks
RTCM	33	Reserve Track Chain
STBM	34	Set Track Bit
VMSM	35	Validate Mass Storage

Fast attach file processing had previously been accomplished with the the AFAS and RFAS subfunctions of the IAUM monitor function. These subfunctions are now part of a new monitor function, AFAM, Access Fast Attach File. This function is one of the functions FTN traps for special handling. Issuing this function for a file residing on an independent shared device will cause 1FA to to be loaded and executed.

1FA processes the AFAM function for independent shared devices completely, without calling CPUMTR. This involves updating the system sector for the fast attach files. Since there is no global table possible for fast attach files, the system sector must be used to control the access contention between mainframes. The advantage of a fast attach file over a normal permanent file on an independent shared device is limited to the elimination of the catalog search. Accessing a fast attach file under Shared Mass Storage is considerably slower than under ECS MMF or stand alone operations if that file resides on an independent shared device.

Programs 1RU and 1FA are executed in high PP memory. The area occupied by these routines and their overlays is saved in a CM buffer prior to their loading and restored upon completion. This process allows the execution of these routines to be invisible to the routine which originally called FTN. This is similar to the way 1DD is currently loaded and executed.

FTN executes a small bootstrap loader to read these routines into the PP from CM. The routines must therefore be CM resident. A 400B CM word buffer is allocated for each pool PP to hold the saved PP area while these routines are executing. NOS allocates the buffers and forces CM residency for 1RU and 1FA automatically if any independent shared devices are present.

Since 1RU and 1FA may perform I/O operations to read and write disk tables, the above functions must not be issued with a mass storage channel reserved. A PP program which was performing I/O must issue an ENDMS macro before calling FTN. It is not necessary to do another SETMS after the function completes since the driver area was restored by 1RU/1FA.

In order to achieve the goal of not impacting a system running stand alone or ECS MMF, PP resident is initialized at deadstart to perform these special checks only if independent shared devices are present in the system. Independent shared devices are identified by the ISHARE CMRDECK entry.

3.3 Updating the MST/TRT

1RU processes the monitor functions trapped by FTN by insuring the CM copy of the MST/TRT is current, then issuing the original monitor function to CPUMTR and rewriting the updated MST/TRT. There is some additional logic in the interest of performance, but we will consider a simplified version of 1RU first:

1. Obtain the hardware unit reserve of the device to be updated.
2. Read the MST/TRT to CM.
3. Issue the original monitor function to CPUMTR. CPUMTR updates the CM copy of the MST/TRT.
4. Write the updated MST/TRT to disk.
5. Release the hardware unit reserve.

Since this method involves considerable overhead, a scheme has been implemented to reduce this overhead when multiple consecutive accesses to the MST/TRT occur from the same mainframe. This involves an interlock called the software device reserve.

3.4 Software Device Reserve

The software device reserve consists of a field in the MST word SDGL. When the field is zero, the reserve is not set. When the field is non-zero, it contains the machine index of the machine holding the reserve. Any machine wishing to update these tables must wait until this reserve is released. The machine setting the reserve uses it to process multiple requests from several PPs within the same mainframe for the same device. The software reserve is set with the first request and released after the last request. The last request writes the updated tables to disk including the MST with the software reserve cleared. The hardware reserve is held until the software reserve is set. The hardware reserve is then released so that other accesses, not involving disk tables, may proceed.

The synchronization of the requests from PPs within the same mainframe is accomplished with the use of the unit reserve count in the MST.

When 1RU prepares to issue a function to update the MST/TRT, the IURS (Increment Unit Reserve count) subfunction of the STBM monitor function is issued to CPUMTR. CPUMTR returns the current value of this count to the calling copy of 1RU. If this count is 0, then this is the initial access to the MST/TRT and the MST will be read by 1RU from disk and written to CM. If the MID in the local portion of the MST matches this machine's MID, 1RU assumes the CM TRT is current and does not copy it from disk. This is the case when this machine was the last machine to update the MST/TRT. If this machine was not the last machine to update the MST/TRT, then 1RU copies the TRT to CM as well.

If another copy of 1RU has already copied the MST/TRT to CM and is in the process of updating it, the second 1RU need not read the disk copy. 1RU determines this by the non-zero count returned from the IURS subfunction. When each 1RU finishes processing the MST/TRT, the DURS (Decrement Unit Reserve count) subfunction is issued. CPUMTR returns the current value of this count to the calling copy of 1RU. If this count is now zero, 1RU rewrites the updated MST/TRT to disk with the software reserve cleared.

The copy of 1RU that initially read the MST/TRT from disk is the one that sets the software reserve, if necessary. If, after the original function is processed, the DURS does not return the count to zero, just the MST will be written with the software reserve set. Since the MST is part of the label sector, this involves a rewrite of this sector.

Whenever the MST indicates that another mainframe owns the software reserve, 1RU delays and rereads the MST, waiting for the reserve to be clear.

3.5 PP Output Register Processing

When PP resident routine FTN is initially passed a function from a calling PP program, the function is written to the output register in CM. This is done before FTN checks to see if this function is one of the functions requiring special processing. If it is one of these functions, the output register and message buffer are saved until 1RU can perform the required independent shared device processing.

3.6 Shared Device Preset

Independent shared devices must be preset much like the way the link device is preset for ECS MMF operations. The first machine to access an independent shared device is responsible for presetting it. This involves initializing the DIT and the Machine Recovery Table (MRT) as well as releasing all device and file interlocks. Presetting is done on a per device basis and is controlled by the CMRDECK entry PRESET and the DSD MOUNT command. The PRESET entry now contains a field for the EST ordinals of independent shared devices to be preset. The MOUNT command contains a *P* parameter indicating that the removable device to be mounted must be initialized as an independent shared device. In ECS MMF, only the first machine to be deadstarted in a complex was required to specify preset. In a Shared Mass Storage environment, devices not accessible by the first mainframe to deadstart must be preset by the first mainframe to access them.

3.7 Deadstart Recovery

The machine recovery utility, MREC, is used in much the same way as it is for ECS MMF. If one machine in the complex goes down, MREC is run on one of the remaining machines to clear interlocks held by the down machine and to remove that machine from the DIT. Once MREC has been run, the downed machine may only be re-introduced through a level 0 deadstart. Since not all independent shared devices which were accessed by the failing machine may be accessed by any other single machine, MREC may have to be run on several remaining machines to process all the independent shared devices connected to the failing machine. The MREC DSD *K* display shows the current status of all independent shared devices connected to the machine on which it is being run. MREC can be directed to process a subset of the devices available.

Program RMS runs at deadstart to recover or initialize all on-line mass storage. Program CMS runs periodically during system operation to

verify the state of on line mass storage. Both programs use common subroutines and overlays and are part of the deck MSM. These programs have been heavily modified to support Shared Mass Storage. When a deadstart is performed, the following error conditions may be detected:

- This machine's ID is present in the DIT of a shared device on a level 0 deadstart. This is an error condition. An operator message is displayed and deadstart is aborted. One of the following courses of action should be followed.

    1. MREC should be run on a machine which is currently accessing the device to remove this MID from the DIT.

    2. If no other machines are accessing the device, PRESET must be specified for this device in our CMRDECK to initialize the DIT.

- This machine's ID is not present in the DIT of an on-line device and this is a level 1 or 3 deadstart. This condition indicates that either the configuration has changed or another machine has removed this machine via MREC or PRESET. In either case, a level 0 deadstart is required.

All checks are done on a per device basis. The action required may differ from device to device.

A situation similar to this may occur when initially associating a removable pack with the system on a device which has been designated as shared. If this machine's ID is already present in the DIT, the device must either be preset or another machine accessing the pack must run MREC to remove this machine ID.

3.8 Machine Failure Considerations

If a machine running Shared Mass Storage fails, one of the following conditions will be present on each independent shared device which it is accessing:

1. The failing machine may possess the hardware and software reserves.

2. The failing machine may possess only the hardware unit reserve.

3. The failing machine may possess only the software reserve.

4. No reserves may be set.

In addition to device reserves, shared controllers may also be reserved by a failing machine.

If the failing machine is to be re-introduced with a level 1 or 3 deadstart, no recovery procedures should be performed on any surviving machines. In the context of a level 1 deadstart, it is assumed that the machine which went down was checkpointed and deadstarted intentionally since a level 1 deadstart should never be used after a machine failure.

If the failing machine will be re-introduced to the complex with a level 0 deadstart, the following recovery procedures should be performed:

- Temporarily idle activity on the remaining machines.
- Clear hardware unit and controller reserves held by the failing machine.
- Run MREC to clear software reserves, track reservations, etc.

Note that MREC should always be run, even if neither hardware or software reserves are held by the failing machine. MREC must remove the DIT entry for the device or a level 0 deadstart by that machine will abort.

Before clearing hardware reserves, activity on the shared devices should be temporarily idled on the remaining machines. This decreases the possibility that a valid hardware reserve held by an operating machine will be inadvertently cleared.

Activating the deadstart button on the down machine (or initiating the deadstart sequence on the 825) will release all controller reservations. In addition, if the controllers involved are 7155 controllers, any units reserved by them will also be released. Remaining unit reserves may be cleared by turning the device off line, then on-line or may be released by running MREC on a surviving machine.

3.9 Shared Mass Storage Unload

CMS checks periodically for the local unload flag and, if set, clears this mainframe's entry from the DIT. When all entries in the DIT have been cleared, the global unload status is set. Care must be taken not to physically remove a pack until all machines accessing the device have logically unloaded it. Global unload status (*N*) must be displayed in the E,M display of all mainframes.

3.10 Checkpointing of Mass Storage Devices

Because the MST/TRT/MRTs residing on independent shared devices are updated each time they are modified, periodic checkpointing of these tables by 1CK is unnecessary. The checkpointing of the local areas is still required. The sector of local areas is present on the label track and contains pointers to the dayfile tracks.

3.11 PFM/PFLOAD/PFDUMP Considerations for Shared Mass Storage

Interlocks for the permanent file activities must be obtained on a per device basis. A new scheme that eliminates the PF interlocks in word PFNL has been implemented for stand alone, ECS MMF and for Shared Mass Storage. The new scheme uses an interlock and count in the MST of each device. PFLOAD obtains the device interlock (in TDGL) before proceeding. This locks out other PFM or PFU activity on that device. PFM and PFDUMP increment the PF activity count in TDGL. A non-zero PF activity count locks out PFLOAD but allows PFM to run when PFDUMP is executing. PFU may request the device interlock with PF activity present. If PF activity remains on the device, the device interlock is set, but a special status is returned to PFU indicating that the PF activity must be checked and that PFU operation must not begin until the PF activity count is zero.

3.12 Mass Storage Driver Considerations

Special care has been taken to prevent two machines from simultaneously updating tables or other information on the same shared device. The following I/O sequence must not release device or controller reserve on an independent shared device.

```
SETMS  READ          INITIALIZE DRIVER FOR READ
RJM    RDS           READ DISK SECTOR
SETMS  WRITE         INITIALIZE DRIVER FOR WRITE
RJM    WDS           WRITE DISK SECTOR
SETMS  READ
RJM    RDS
```

The device (unit) reserve prevents access to the unit from other controllers. The controller reserve prevents access to the controller from other mainframes. Both reserves are necessary to prevent another mainframe from accessing the device while one machine has an I/O operation in progress.

MTR controls the seek processing for all mass storage I/O requests. On non-shared devices, it is not necessary to retain the controller reserve during this operation. If the unit is being shared, however, accesses to the controller from other mainframes must be locked out during this time. If any outstanding seek operations are in progress on a particular controller for a shared device, the controller must not be released when another I/O operation completes on the same controller. MTR maintains a count of outstanding seeks to shared devices on each controller. If this count is non-zero, a drive release function is issued instead of an operation complete function at the end of the mass storage operation. An operation complete causes both the unit and controller reserves to be cleared. The drive release does not release the controller.

To prevent one machine from keeping the controller reserved for an unlimited period of time, a count of consecutive seeks issued without releasing the controller reserve is maintained by MTR. When this count reaches a certain threshhold, no new seeks are initiated for shared devices on that controller. When the last shared I/O operation is complete, the controller is released allowing another mainframe to access the controller.

3.13 Dayfile Message Processing

1RU and 1FA maintain the channel and unit reservation during function processing. They must not get into a situation which requires a dayfile buffer to be dumped. These programs (and 1DD) use the suppress message option (SM) on the SETMS macro. This causes a flag to be set such that when a dayfile message is issued, CPUMTR will insert the message into the dayfile buffer as long as there is room. If there is not sufficient room, the message is discarded and an informative message will be issued to the dayfile upon completion of the operation to indicate that dayfile messages have been lost. Previously, the SM option caused the driver to suppress issuing dayfile messages. Now, the driver will issue the messages and messages will be lost only if dayfile buffers are full.

4 Conclusion

Shared Mass Storage provides a new potential for multi-mainframe sites with a need to share a permanent file base or selected permanent file devices. Since Shared Mass Storage may impact a site's performance, the site should weigh the advantages of Shared Mass Storage against the performance impact. Sites should first consider their reasons for sharing devices. These reasons could be:

- A second machine accessing a set of files provides a backup machine in case of a failure of one of the machines.
- One machine may be used for interactive purposes while others perform mainly batch services.
- A site's workload may be to large for any one machine.
- Shared Mass Storage may be used to transfer files from one machine to another.

If a site has a need to share files, the two methods of sharing should be considered. The ECS MMF method provides superior performance but is not possible or practical at many sites. Sites should also consider other alternatives such as a Loosely Coupled Network serviced by the Remote Host Facility. This facility is available with NOS 1.4 at level 552 and will soon be available in NOS Version 2. If Shared Mass Storage is to be run, care should be taken to minimize the impact through proper mass storage allocation and configuration.

NOS V2

CYBER 176/819 SUPPORT

Cyber 176/819 Support

NOS Version 2.0 supports the 819 (single/double density) disk on the Cyber 176. Note that the 819 is only supported on the Cyber 176 due to the requirements of first level PPUs (FLPP) to drive the 819 hardware. The 819 data is transferred through Large Core Memory (LCM), also only available on a Cyber 176.

NOS uses LCM as a data cache for all 819 devices, permitting multiple access to a particular piece of data by multiple users.

1 819 Disk

The 819 disk has four separate surfaces which are read or written by four parallel heads. The 819 is available in both single and double density format. The single density 819 has:

- 620B sectors/track
- 3134B tracks

The double density 819 has:

- 1440B sectors/track
- 3144B tracks

10B tracks are reserved for customer engineer use on both types of devices.

There are no restrictions on file types that may reside on the disk (i.e., it may contain the SYSTEM file, queued files, permanent files, etc.). Only one copy of the SYSTEM file is recommended, as a second SYSTEM device defeats the cache algorithm.

The 819 disk is capable of sustaining a 5580 kilo-character transfer rate, which is approximately five times the speed of a full tracked 885 disk.

Note that NOS will run in a mass storage environment of only 819 disks, but an 844 or 885 disk is required for maintenance.

2 Notes and Cautions

Deadstart from an 819 is not available, and there are no on-line diagnostics for the 819. You must execute off-line diagnostics by dumping and reloading the device. The maximum 819 configuration is 12 drives. The minimum amount of LCM required to run is 512K words, with the LCM buffering requiring a minimum of 40,000B words.

NOS first places data to be written to an 819 in LCM. This LCM data is flushed to the 819 only at certain times (e.g., job termination, job rollout, or device checkpointing). This factor can be critical in some transaction logging and recovery applications.

3 Error Reporting

Error reporting for the 819 is somewhat different in that you will seldom see any errors reported in your dayfile. If an error occurs during a physical read of the 819 disk, you are not notified of the error until you actually try to read the data that was determined to be in error.

If an error occurs during a physical write of the 819 disk, that error is not reported to you, since your job may have already terminated.

Both read and write errors are logged in the binary maintenance log, but not in the system error log or dayfile. Such read or write errors also cause a flashing message to appear at the system control point. The binary maintenace log can then be dumped and interpreted through HPA to extract detailed information regarding the error.

4 CMRDECK Entries

All 819 CMRDECK entries are described in the NOS Installation Handbook (60459320). The following sections describe some of the entries which are critical to 819 performance.

819 performance statistics on hash table and buffer utilization are kept in the buffer manager statistics table. The format of this table is described in the common deck COMS176. Its location in LCM can be obtained from a listing of CPUMTR, symbol BMST in common deck COMS176.

4.1 PP I/O Buffer Definition

The PPB=x. entry specifies the number of CM buffers to be allocated in low core for use by CPUMTR and PPs to pass data back and forth from LCM and therefore also the 819s. A PP cannot read LCM directly and must request CPUMTR (via the PIOM monitor function) to copy the data into a CM buffer so the PP can read it. This method (with a few embellishments) is also used for PP access to the 819 I/O handler. Therefore, enough buffers are needed so that PP I/O is not bottlenecked because there are more PPs wanting CM buffers than are available.

4.2 819 Buffer Space in LCM

The value of the 'IOB=n.' entry is very important to system performance. Testing by CDC has indicated that as this value increases, so does performance. Thus, the best value to use is the amount of LCM remaining after the alternate SYSTEM library and USER ECS are allocated. It is not advised to use LCM as a secondary rollout device.

Specifying an IOB value that is too small could cause the system to thrash and even have data re-read several times before it is actually used.

4.3 Hash Buffer Size

The hash buffer size that is specified on the 819 equipment entry is also important to performance. This buffer is used to track a device's LCM resident data. Each device has entries for non-continuous pieces of data. If the hash buffer is too small, the hash table will become full, and the record of data in LCM is lost to make room for new entries. Note that no data is lost, just the fact that the data is already in LCM. Hence, too small a value for this parameter could also cause thrashing.

Due to the high usage of the SYSTEM device for short pieces of data, the hash buffer size for this device should be larger than that of other devices.

5 Overall Design

The maximum transfer rate of an 819 device could not be maintained by use of the standard I/O request processing through a PP. Therefore, a majority of 819 I/O is performed by modules in CPUMTR. CPUMTR acts much like a regular PP, directing the rest of the system to process actual 819 I/O requests, and copying the data to and from the LCM cache into and out of the user's I/O buffer. This process, although substantially different, still maintains the same interface to the user program as the one always used for disk I/O in NOS. In addition to the increased speed of I/O, PP utilization in most systems is noticeably decreased, or at least PP's are made available for other purposes.

CPUMTR has been modified to support the 819 disk, which consequently required support of both FLPPs and the buffering of the larger 819 disk sectors (1000B words each) in LCM. The new modules in CPUMTR include:

- 819 request processor
- LCM Buffer Manager
- I/O control
- interrupt handler

Both PP and RA+1 requests are processed by the LCM Buffer Manager. The function of this module is to allocate and control the buffers which are required to block/deblock the NOS 100B word PRUs into the 819's 1000B word sectors.

The I/O control module processes the initiation of I/O requests and queues requests if the selected unit is currently busy. The Interrupt Handler processes the transfer of data to/from the LCM buffers from/to the CM I/O buffers which in turn feed the FLPP driver. The FLPP driver processes all disk functions, error recovery, seek overlap, and disk transfers. This driver is the same one used in the NOS/BE and Scope 2 operating systems to support the 819.

5.1 819 Request Processor

The 819 request processor is called by CPUCIO for all CIO requests for 819 devices which are not processed directly by CPUCIO (e.g., REWIND).

This processor performs most of the common I/O requests, and passes any that it cannot do on to a PP. A PP is also called to do any error processing.

5.1.1 Read Request

The LCM Buffer Manager is called on a read request to determine if the desired data is already in an LCM buffer. If it is, the address of the buffer is returned and the data is copied from the LCM buffer into the user's buffer.

If the desired data is not in an LCM data buffer, a recall condition is returned and the request is placed in the recall table for that control point. The reason for the recall condition is that disk I/O is required to obtain the data.

5.1.2 Write Request

This is quite similar to a read request. The LCM Buffer Manager either returns a data buffer address or the recall condition.

5.2 LCM Buffer Manager

The LCM Buffer Manager controls the allocation of the LCM data buffers (cache) and maps 819 I/O requests to the buffer which contains the data. The requests contain the equipment, logical track, logical sector, and I/O function. The logical track is mapped into an entry in a hash table for that equipment. This entry is used to determine if the request is already in a data buffer. If so, the address of the data buffer is returned to the caller.

If the information is not in a data buffer:

- a free buffer is allocated
- recall request status is returned
- I/O is initiated by calling the I/O control module

The LCM Buffer Manager also insures that write buffers (e.g., a 100B word logical sector) are correctly positioned within the 819's 1000B word sector.

The number of LCM buffers the LCM Buffer Manager can use is controlled by the previously mentioned CMRDECK entry IOB.

5.3 I/O Control

The function of I/O control is to find the mass storage table (MST) for the requested 819 unit and initiate the I/O request on the unit. If the unit is busy, the request is attempted on a possible alternate path. If both paths are busy (or there is only one path) the request is entered in the queue for that unit.

5.4 Interrupt Handler

The interrupt handler copies data between the CM FLPP I/O buffers and the LCM data buffers. When the current I/O operation nears completion, the interrupt handler determines if the next request for this unit can be handled without repositioning; if so, the FLPP driver is directed to

process this request. If repositioning is required the interrupt handler:

- waits until the current request is completed
- issues the repositioning request and any other requests that are queued for idle units
- waits for the FLPP driver to interrupt the CPU when a drive is on position for a transfer

NOS V2

FNT REORGANIZATION

FNT Reorganization Feature Notes

1 Introduction

This article provides a general overview of the FNT Reorganization project. It is intended to be a primer to the FNT Reorganization Design Notes, where detailed implementation information may be obtained.

A general discussion of the reorganization of the FNT will be presented. This will be followed by a discussion of some limitations that have been eliminated as a result of the project. Finally, new capabilities arising from the project will be presented.

2 Division of System FNT

Files in the system FNT have been divided into four groups: local files, rollout files, queue files and system files. Common files have been eliminated. New table structures have been developed to accomodate each of these four groups.

FNT entries for files assigned to a job (local files) have been removed from the system FNT and placed in an FNT local to a job. The field length for a job has been expanded to include an area immediately preceeding the job's RA which contains this local FNT, along with the dayfile buffer and pointers, command buffer, DMP= parameter area, and an installation area. This new area is known as "Negative Field Length", or NFL. NFL is contained within the 131K field length limit for any job. NFL is also included within a user's field length validation limit. See appendix B.4 of the FNT Reorganization Design Notes for a description of NFL.

Rollout files have been removed from the system FNT. Entries in a new table called an "Executing Job Table", or EJT, keep information pertaining to all executing jobs. An EJT entry is present for each job regardless of whether the job is rolled out or executing at a control point. The EJT entry for a particular job remains positionally fixed for the life of the job. See section 3.2 of the FNT Reorganization Design Notes for a description of the EJT.

Queue files have been removed from the system FNT. All active queue files have an entry in a new table called the "Queue File Table", or QFT. See section 3.1 of the FNT Reorganization Design Notes for a description of the QFT.

The only files that remain in the system FNT are fast-attach files and the SYSTEM file. See section 3.4 of the FNT Reorganization Design Notes for a description of the system FNT.

The format of all four types of table entries have been changed from the familiar two-word FNT format. The sections referred to above in the FNT Reorganization Design Notes have detailed descriptions of each table and its entries.

3 Eliminated Limitations

The following is a list of well-known limitations in NOS Version 1 that have been eliminated as a result of the FNT Reorganization project.

1. Almost all CMR space restrictions have been eliminated. Among those that remain are the following.

   . The EST must reside before 7777B.

   . All MSTs must reside before 77771B.

2. A system sector access is not required to obtain information for queue file or job selection. All information required for selection has been moved to the table entries.

3. Many interfaces to file queueing functions have been eliminated. Queue file functions are handled by either DSP or QAC.

4. The DB validation parameter, which controls queue file flooding, operates correctly for identical user indices on multi-family systems.

4 Resultant Capabilities

The following is a list of the capabilities that have arisen from the FNT Reorganization project. Sites migrating from NOS Version 1 systems should carefully review these capabilities in light of any local modifications that have been previously developed.

4.1 Operational Capabilities

1. Level 3 deadstarts will recover jobs in core, under most conditions. This recovery amounts to continuing execution at the next EXIT command in the job stream with either a DRET (deadstart rerun) or RAET (recovery abort) error flag set. See sections 3.2.10, 3.32.8 and 3.32.9 of the FNT Reorganization Design Notes.

2. A generalized mechanism for operator notification of system events has been developed. A new *A,OPERATOR* display posts messages indicating the events. Currently, table full conditions, track limits, and thresholds exceeded for DAYFILE/ERRLOG/ACCOUNT/BML size are reported through this mechanism. See sections 3.15.2.6 and 3.20 of the FNT Reorganization Design Notes.

3. Operator control of jobs and queue files is consistent regardless of execution state. The operator need not be concerned over whether a job is rolled out or executing at a control point, in order to refer to it. See sections 3.15.4.5, 3.15.4.7 and 3.15.4.8 of the FNT Reorganization Design Notes.

4. A new L-display enhances operator/console interaction. See the L-display Feature Notes and section 3.18 of the FNT Reorganization Design Notes for more information.

4.2 System Capabilities

1. A mechanism has been developed which handles multiple PPs in recall per job, up to a maximum of 17B. For example, it is now possible to have a PP recall itself and the job rolled out on a "track limit" condition.

2. CPUMTR preprocesses all CIO RA+1 requests. A portion of CPUMTR, known as CPUCIO, prevalidates all CIO RA+1 requests and passes them off to the appropriate driver, which may be either a PP or CP program, depending on the equipment type. The PP program CIO no longer exists. A new PP program 1MS drives 844s and 885s. See appendix I of the FNT Reorganization Design Notes for more information.

3. Priority evaluation and job scheduling is performed by service class, not job origin. All QUEUE and SERVICE parameters are grouped by service class. See section 3.7 of the FNT Reorganization Design Notes for a description of service classes. See section 3.8 of the FNT Reorganization Design Notes for a description of priority evaluation.

4. All jobs and queue files are uniquely identified by a "Job Sequence Name", or JSN. See section 3.19 of the FNT Reorganization Design Notes for more information.

5. Accounting messages have been added for complete job and queue file tracking and charging. See sections 3.7.5 and 3.28.4 of the FNT Reorganization Design Notes for more information.

6. Job termination is common to batch and interactive jobs. IAF no longer performs job termination. See section 3.13 of the FNT Reorganization Design Notes for more information.

7. The NJ SERVICE parameter is enforced for each service class.

8. The maximum number of control points is increased to 27 (octal).

4.3 IAF Capabilities

Most of the following capabilities have arisen from the "transferring of ownership" of a job from IAF to the system through the implementation of the EJT.

1. IAF jobs can access System Control Points (SCP) (e.g. CDCS, MCS, TAF, etc.) as User Control Points (UCP).

2. User break processing sets error flags for IAF jobs. This allows EXIT processing and programs using REPRIEVE to process both user breaks 1 and 2.

3. Jobs may be detached from a terminal. See the usability feature note entitled "User Control over Submitted Jobs" and section 4.11 of the FNT Reorganization Design Notes for more information.

4. Job suspension and recovery has been enhanced. See sections 3.12 and 4.4 of the FNT Reorganization Design Notes for more information.

4.4 Installation Capabilities

1. New table entries have areas "Reserved for Installation".

2. New table entries lengths are variable at installation time. In most cases, table entries may be increased to 8 words, where 2 of these words are reserved for installation use. without adverse effects.

3. Predefined symbols have been provided for installation defined service classes, queue file types, and subsystems. It is the responsibility of the site to supply supporting code.

4. Space in several space critical decks have been reserved for installation use. This space is made available to sites by deleting a symbol in NOSTEXT. See section 3.34.2 of the FNT Reorganization Design Notes for more information.

NOS V2

L - DISPLAY

The L-DISPLAY

There is a need for additional DSD displays, more complex displays, easier methods of creating new displays and more operator interaction with some of the displays. The new L-display addresses these needs. DSD gets the data for the L-display from a CMR buffer. The CMR buffer is filled by specially formatted data from a CPU program that runs at a control point. The execution of the CPU program is initiated by a DSD command. The L-display comes automatically to the left screen when this command is entered. However, rather than executing a DSD overlay to format the data, DSD causes a program to be run at a control point to do the formatting. Thus the L-display overlay can be used for any number of commands without significantly changing DSD.

The L-display may interact with the operator, returning values entered by the operator to the CPU program for further processing.

The L-display is similar to the K-display in the way the operator interacts with the display and in the macros used to format the screen and accept input. It is different from the K-display in the way it is initiated and the location of the display buffer.

Three commands use the new L-display. They are:

COMMAND	DESCRIPTION
QDSPLAY(JSN)	Display the contents of a file in the Queued File Table (QFT)
FOID.	Display the Family Ordinal Table (FOT)
SUBSYST.	Display Subsystem Information

After entering the L-display command, if interaction is necessary, the operator enters commands to the program in the following format:

L.command.

The 'L.' is entered automatically whenever the program requests L-display input.

The CPU program that formats and interacts with the display is initiated by DSD and 1DS as a result of an operator command. The job is put, by 1DS, into the INPUT queue as a System Origin, System Service class job. The L-display is automatically brought to the left screen.

The formatted display data is moved to the CMR buffer as a result of a CPM RA+1 request. New Macros have been written to issue the requests. The display buffer is in CMR rather than in the programs field length so that the program can be rolled out during the time that no interaction or updating of the display is required. As a matter of fact, the job may even terminate if interaction and dynamic updating are not required while the display is being reviewed. This can be done by leaving the data for the display in the CMR buffer until another L-display is called or this one is called again.

Because the L-display uses a single CMR buffer, only one L-display program can be active at one time.

DSDOUT/DSDINP Macros

The CPU program initiated by the DSD command communicates with the system with two new macros: DSDOUT & DSDINP. These macros are defined in COMCMAC.

The utility formats the display in exactly the same way as for the K-display in a CM buffer in it's field length. (See the CONSOLE macro in the NOS V.1 IMS section 28) It then executes the macro DSDOUT. This macro calls CPM. CPM will then move the data from the programs field length to the buffer in CMR. DSD will then display it from the CMR buffer for the operator to see. When the L-display is called to either the left or right screen, DSD continuously displays the current contents of the CMR buffer.

The macro call has the following format:

```
DSDOUT          addr
```

"addr" is the address of the buffer that contains control information and the data to be displayed.

The CPU program may request input from DSD using the macro DSDINP.

The macro call has the following format:

```
DSDINP          addr,C
```

Where "addr" is the address of a buffer to hold the input from DSD, "C", if present, indicates that DSD should not accept input from the operator. For example, it is used when the program is going to drop out. To receive more lines, another DSDIN should be issued.

If the program has made a DSDINP request and the L-display is not on the screen, DSD will display the intensified message on the right screen name area:

```
see *L* display
```

All operator entries prefixed by "L." will be transferred to the CMR INPUT buffer by DSD, which CPM will then transfer to the field length to the CPU program.

The program can make a DSDINP request before a DSDOUT, but "L." will not appear on the screen until the program makes a DSDOUT request. "L." commands are allowed when the L-display is not assigned to the screen.