USER'S GUIDE

MAN-MACHINE INFORMATION SYSTEM

Stanford Research Institute
Menlo Park, California

MEMO ON NEW USER GUIDES
DGC 7/8/69
Distribution -- All AHI

The User's Guide notebooks are being put through a major revision (which will be followed by a series of minor revisions as individual sections are changed).

Some of the numbered sections are now subdivided by plain (un-numbered) dividers. These have been put in with their tabs at the top edge.

Also, at the end of each numbered section you will find a plain divider with the tab at the bottom edge.

> If you have material of your own which is relevant to a particular numbered section, place it behind this plain divider -- it will then be safe from the revision process.

As before, Section 0 is reserved for your own use.

The section on NLS commands anticipates the next version of NLS in three respects:

> "Jump to Identity" is called "Jump to Item".
>
> "Execute File Check" is described, although it is not yet available.
>
> "Execute File Cleanup" is called "Execute File Grope".

U S E R G U I D E

CONTENTS

Note on this User's Guide

This edition of the User's Guide uses a new, tentative kind of organization -- it is planned as a group of independent but cross-referenced documents.

Wherever a reference is made to another "document", the reference is to another section of the User's Guide.

Please report inaccuracies and confusions to Dave Casseres.

Note on On-Line NLS Documentation

NLS is also Documented in an on-line file with some on-line aids built into it. The on-line version may be found in KDF as (Casseres)NLIST.

NLIST contains links to (Casseres)CONAN, (Casseres)LINKS, and (Casseres)INFOR, which describe the content analyzer, the link system, and the information-retrieval system respectively. These files are also given as hard copy in the User's Guide.

The origin statement of NLIST contains some useful names to jump to.

NLIST is designed to be viewed with statement numbers off. The branch-only feature may also be useful.

A number of trails are built into NLIST; they may be entered from Statement (trailset).

Mouse Buttons

Right-Hand Button

When pushed and released without any intervening input, this button gives a CA (command accept).

When it is held down while a LIT is entered from keyboard or keyset, this button causes the LIT to be interpreted as a reference to a pointer.

Center Button

When pushed and released without any intervening input, this button gives a CD (command delete).

When it is held down while a LIT is entered from keyset, this button causes the LIT to be interpreted as Case 1 input (i.e., letters come out upper-case).

Left-Hand Button

When pushed and released without any intervening input, this button gives a backspace, causing the last input character (in a literal type-in) to be thrown away.

A backspace made during the process of a bug selection causes the last selection made to be cancelled. This only works in certain commands; please see entry on "bug selection" in document ODDS AND ENDS.

When it is held down while a LIT is entered from keyset, this button causes the LIT to be interpreted as Case 2 input.

Left-Hand and Center Buttons Together

When pushed and released without any intervening input, this combination gives a backspace-word, causing the last input word (in a literal type-in) to be thrown away.

When it is held down while a LIT is entered from keyset, this combination causes the LIT to be interpreted as Case 3 input.

Notes on Syntax Equations

The syntax equations in this User's Guide are not rigorous. They are not completely consisent as to notation, nor are they guaranteed to be accurate.

The letters at the beginning of each equation are the control letters entered from keyboard or keyset to specify the command.

CA means "command accept;" this is done by pressing either of the two CA keys on the keyboard, or the right-hand button on the mouse.

Square brackets are used to indicate selections made with the bug. Thus [c] means that a character is selected, [v] means that a visible string is selected, and so forth. If two characters are to be selected, they are shown as [c]1 and [c]2.

Please see "bug selection" entry in document ODDS AND ENDS.

"LIT" means any string of characters input from the keyboard or keyset.

"NUMBER" means any number entered from the keyboard or keyset.

The slash (/) means "or."

The dollar sign ($) by itself means "any number of" (from 0 to 1000). The construct m$ means "any number equal to or greater than m;" the construct $n means "any number equal to or less than n;" the construct m$n means "any number from m to n."

Examples: $3(NUMBER) means "up to three numbers"; 3$5(char) would mean "three to five characters".

The term VIEWSPEC in a syntax equation means that VIEWSPECs may be set.

For example, the syntax equation for the Replace Text command (see rt) is r t [c]1 [c]2 LIT CA. This means "type r; type t; select a character; select a second character; type a string of characters; do a command accept."

SYSTEM STARTUP 10/3/68

1 This document gives details on how to restart the system after a crash.

2 First, see who else is around.

2A If there is someone who knows more than you do, enlist his aid.

2B Otherwise, proceed with the following procedures. Each of the procedures should be tried in order until one of them produces satisfactory results. The procedures are:

2B1 Breakpoint 4

.2B1A Leaves user entered and RAD files OK.

2B2 BRU 74 with Breakpoint 4

.2B2A Leaves user entered and RAD files OK.

2B3 Fill from Paper Tape

.2B3A Leaves RAD files OK, but user must reenter.

2B4 Recovery from Mag Tape

.2B4A Leaves RAD files OK, but user must reenter.

.2B4B NOTE: Just now there are irregularities which make this procedure highly inadvisable unless you are really sure what you are doing. The steps given in this document for this procedure will not give you a working NLS.

2B5 Initialization from Mag Tape

.2B5A Destroys RAD files, and user must reenter.

.2B5B NOTE: Just now there are irregularities which make this procedure highly inadvisable unless you are really sure what you are doing. The steps given in this document for this procedure will not give you a working NLS.

2C If anyone else is using the system, make sure they know you are going to attempt a restart.

2D Before starting the procedure, record the crash in the notebook marked "AHI Information," which will be found on or near the console. To obtain register contents, turn the dial marked REGISTER to the desired register designation, then read off the lights.

3 Breakpoint 4

3A Momentarily depress Breakpoint Switch 4 on the console and then return it to the up (RESET) position.

3B If this succeeds, Teletype 1 should respond SYSTEM RESTARTED PROCEED WITH CAUTION, and then the logging Teletype should type F.D. CLEANUP STARTED followed by F.D. COMPLETED.

3C If it fails, try a BRU 74 with Breakpoint 4.

4 BRU 74 with Breakpoint 4

4A Stop the computer by moving the RUN/IDLE/STEP switch to IDLE.

4B Press START.

4C Turn the register dial to C, then use the pushbuttons below the register display to enter BRU 74 (00100074 octal).

4C1 The binary code for this (corresponding to lights showing in the register) is 000,000,001,000,000,000,111,100.

4D Push Breakpoint Switch 4 down.

4E Move the RUN/IDLE/STEP switch to RUN.

4F Return Breakpoint Switch 4 to the up (RESET) position.

4G If this succeeds, Teletype 1 should respond SYSTEM RESTARTED PROCEED WITH CAUTION, and then the logging Teletype should type F.D. CLEANUP STARTED followed by F.D. COMPLETED.

4H If it fails, try a Fill from Paper Tape.

5 Fill from Paper Tape

5A Load the System Recovery Tape into the paper tape reader.

5A1 This tape is a loop and should be found on top of the realder/punch unit. It is labeled CRASH RECOVERY.

5A2 After sliding the tape into the reader unit, move the RUN/LOAD switch on the reader to RUN.

5B Stop the computer by moving the RUN/IDLE/STEP switch to IDLE.

5C Push the START button.

5D Move the RUN/IDLE/STEP switch to RUN.

5E Move the FILL switch to PAPER TAPE and release.

5F Wait for the tape to fill and the HALT light to appear.

5F1 Move the RUN/LOAD switch on the reader to LOAD and remove the tape.

5G Check the blackboard for a note as to whether Breakpoint Switch 1 is to be used.

5G1 If it is, depress Breakpoint Switch 1 and leave it down.

5H Move the RUN/IDLE/STEP switch to IDLE and then back to RUN, to clear the HALT light.

5I If Breakpoint Switch 1 has been depressed, move it to the up (RESET) position.

5J You must now bring the system up, as follows:

5J1 Type a RUBOUT on Teletype 1.

5J1A If the response is WAIT, go back to the beginning of the ifill procedure.

5J1B If you have succeeded, the response will be TSS 1.91 IS IUF SET DATE AND TIME. Set the date and time in the correct iformat.

5J2 The system will type INITIALIZE OR RECOVER. Respond by typing R.

5J3 F.D. cleanup will be started and completed.

5J4 Enter SYSTEM as a user at Teletype 1.

5J5 Set executivity to -1.

5J6 The tapes on the drives must be remounted. Check the UNIT SELECT switches on the drives and use the MOUNT command to mount the tapes. (See Note on Tape Mounting at the end of this document.)

5K Move Breakpoint Switch 1 to the up (RESET) position, if you set it earlier.

5L Reenter yourself as a user from your own console.

5M If this process fails, try a Recovery from Mag Tape.

6 Recovery from Mag Tape

6A Check the blackboard to find the reel number of the recovery tape. The tape is on a MAC reel labeled RECOVERY and stored on the second shelf of the tape cabinet.

6B Use the MOUNT command on tape unit 0. When the system types MOUNT NEW TAPE, remove the tape that is on the drive, set up the recovery tape in its place (using 556 BPI), and hit the period on the Teletype to execute MOUNT NEW TAPE. (See Note on Tape Mounting at the end of this document.)

6B1 Set the other tape drives to MANUAL.

6C Use the same procedure as Fill from Paper Tape, except that the FILL switch is set to MAG TAPE.

6D After bringing up the system, restore the original tape to tape unit 0, and put away the recovery tape.

6E If this procedure fails, try initializing from mag tape.

KDF Commands

(See end of document for definitions of terms.)

Abandon File

Syntax: a <KDF filename> .

Semantics: This command is used when repeated KDF read or write errors indicate that the file is written on a bad disc spot somewhere. The file is deleted and the space is not used again for other files. It will be listed under the "status" command as a bad spot.

Brief List

Syntax: b (. / sp username .)

Semantics: A list of file names in KDF is presented, in a format that allows many files to be listed without filling the screen.

If the "b" is followed immediately by a period, the current user's files are listed.

If a space is typed instead, KDF responds with "FOR USER" and expects a user name. Then when the period is typed, the specified user's files are listed.

Change File Name

Syntax: c <old KDF filename> . <new KDF filename> .

Semantics: The old name is changed to the new.

Delete File

Syntax: d <KDF filename> .

Semantics: The named file is deleted and its name thrown away.

Examine File

Syntax: e <KDF filename> .

Semantics: A line of information about the named file is displayed. This includes the accessibility, length, type, etc. See note on "accessibility", below.

Where a file belonging to the current user has a password associated with it, the password is displayed in parentheses.

Where a file belonging to another user has a password associated with it, a pair of parentheses is displayed but the password itself does not appear.

Finished

Syntax: f .

Semantics: KDF is terminated and control returns to the Exec.

General Access

Syntax: g <KDF filename> . r/w/n .

Semantics: This sets the accessibility of the named file with respect to a "general" user -- i.e. one who is not the file owner and does not use a password (if applicable).

Only the file owner may execute this command.

"r" sets accessibility to read-only.

"w" sets accessibility to read-write.

"n" sets accessibility to no access.

See note on "accessibility", below.

Initialize

Syntax: i n i t i a l i z e .

Semantics: The user's file space is reinitialized, and all existing files are deleted.

List

Syntax: l (. / sp username .)

Semantics: A list of files in KDF is presented, along with information on each file (accessibility, length, type, etc.). See note on "accessibility", below.

If the "l" is followed immediately by a period, the current user's files are listed.

Where a file has a password associated with it, the password is displayed in parentheses.

If a space is typed instead, KDF responds with "FOR USER" and expects a user name. Then when the period is typed, the specified user's files are listed.

Where a file has a password associated with it, a pair of parentheses is displayed but the password itself does not appear.

Owner Access

Syntax: o <KDF filename> . r/w/n .

Semantics: This sets the accessibility of the named file with respect to the file owner.

Only the file owner may execute this command.

"r" sets accessibility to read-only.

"w" sets accessibility to read-write.

"n" sets accessibility to no access.

See note on "accessibility", below.

Password Access

Syntax: p <KDF filename> . <new password> <rparen> r/w/n .

(After the first period is typed, the system responds with "IS (". The password is typed and the parenthesis closed, then an r, w, or n is typed, and finally another period.)

Semantics: This sets the accessibility of the named file with respect to a general user (i.e. not the file owner) who uses the given password in association with the named file.

Only the file owner may execute this command.

"r" sets accessibility to read-only.

"w" sets accessibility to read-write.

"n" sets accessibility to no access.

See note on "accessibility", below.

Read Disc File

Syntax: r <KDF filename> . <RAD filename> .

Semantics: The named KDF file is copied to the named RAD file.

Status

Syntax: s (. / SP username .)

Semantics: Lists a user's file space by 2K blocks, 8 blocks to the line. Each entry is either *NG* (bad spot), -NU- (not used), or a file name followed by a file block number or by (E) if it is the last block of the file.

If the "s" is followed immediately by a period, the current user's files are listed.

If a space is typed instead, KDF responds with "FOR USER" and expects a user name. Then when the period is typed, the specified user's files are listed.

Verify Data

Syntax: v (. / SP username .)

Semantics: Reads all the data in a user's KDF files. A message is displayed if hardware indicates that any data are unreadable.

If the "v" is followed immediately by a period, the current user's file data are read.

If a space is typed instead, KDF responds with "FOR USER" and expects a user name. Then when the period is typed, the specified user's file data are read.

Write Disc File

Syntax: w <KDF filename> . <RAD filename> .

Semantics: The named RAD file is copied to the named KDF file.

Explanation of Terms Used in this Document

RAD file: As used here, this means either a drum file or a non-KDF disc file.

KDF filename: As used here, this includes a username and a password, if needed.

username: The username is given in parentheses, just as it is for a RAD file. If the file being named is the current user's own file, the username and its parentheses are omitted.

password: The password is only used where a username is also used (because the owner of a file never needs to give the password). The username is followed immediately by a comma, then the password, and finally the closing parenthesis. The comma and the password are not echoed to the display or TTY.

NOTE: Names of KDF files are not recognized until the entire name has been typed. However, user names and names of non-KDF files are recognized as soon as enough characters have been typed.

Accordingly, it is not necessary to use apostrophes in defining a new KDF file name. If apostrophes are typed, they are considered to be characters in the file name. However, in defining a new RAD-file name from KDF, it is still necessary to use apostrophes.

KDF recognizes the BACKSPACE key (or CONTROL A), but does not delete the character from the display. Instead, it displays an up-arrow.

accessibility: Three completely separate categories of file accessibility are defined: owner, general, and password.

The "owner" is the user in whose KDF space the file is stored. "Owner access refers to access by the owner, unless he gives a password; in this case it is "password access".

Owner access is initialized as read-write.

"General access" refers to access by anyone who is not the owner and does not give a password.

General access is initialized as read-only.

"Password access" refers to access by anyone who gives a password.

Password access is not initialized; it is set by the owner when he defines the password.

Each category may be set independently to any one of three modes:

W means read-write

R means read-only

N means no access.

Notice that illogical situations can result: if, for example, the user defines a password for a file and sets password access to read-only, but forgets to change the general access, then the password access will be the same as the general access. To keep general users from reading the file, general access must be set to "no access."

It is also possible for a user to set his own access to one of his files to "no access." Fortunately he can change it back to read-write with the "Owner Access" command.

7 Initialization from Mag Tape

7A Check the blackboard to find the reel number of the initialize tape. The tape is on a MAC reel labeled INITIALIZE and stored on the second shelf of the tape cabinet.

7B Use the MOUNT command on tape unit 0. When the system types MOUNT 1 NEW TAPE, remove the tape that is on the drive, set up the initialize tape in its place (using 556 BPI), and hit the period on the Teletype to execute MOUNT NEW TAPE. (See Note on Tape Mounting at the end of this document.)

7B1 Set the other tape drives to MANUAL.

7C Use the same procedure as Fill from Paper Tape, except that the FILL switch is set to MAG TAPE and the response to INITIALIZE OR RECOVER is I instead of R.

7D After bringing up the system, restore the original tape to tape unit 0, and put away the recovery tape.

7E After the fill procedure is finished, SYSTEM is entered as a user. If tapes are remounted, etc., it is necessary to copy several files from the disc to the RAD. This should be done from Teletype 1, as follows:

```
@KDF.

&READ DISK FILE CCP.

TO .*/CCP* (NEW FILE).

&READ DISK FILE CCPLD.

TO */X/* (NEW FILE).

&&FINISHED.

@CCP.

TELETYPE # 2.      (or any number not in use)

ARGUMENTS ()

INPUT FILE /X/   (CCP will run, then return to exec.)

@DELETE NAME /X/.
```

7F If this initialization from mag tape fails, go home.

8 Note on Tape Mounting

8A The term MOUNT refers strictly to a software process, not to the physical placement of a tape on a drive. The latter will be called "setting up" a tape.

8B The MOUNT command is used to rewind a tape that is on a drive, doing the necessary things to the file directory, etc., either to get it recognized after a restart or in preparation for setting up a new tape.

8BJ After this has been done, the system will type MOUNT NEW TAPE. If a new tape is not to be set up, type a period and the tape on the drive will be made ready for use. If a new tape is to be set up, remove the old tape at this point, set up the new one (as described below) and then type the period. This will cause the new tape to be made ready for use.

8C Setting Up a Tape

8C1 (not written yet -- see notes in the "AHI Information" notebook at the console.)

/STARTNLS, 08/28/68 0928:27 DIA ;

! How to start up NLS

!A Enter as NLS.

!B Put disk file (HAY)C-NLS on the rad and give it duribility P.

!C Get files (HAY)G-NLS and (HAY)G2NLS from the disk also.

!D PLace /G-NLS and do a SYS-DUmp 0 to 7 band 4 page 1.

!E RESet and PLace /G2NLS and do a SYS-DUmp 0 to 3 band 16.

!F RESet and do a SAve 0 to 37777 on '/CRASH'.

!G Make the file /CRASH public access RW, and duribility P.

!H Since NLS is read only and shared, if the monitor thinks an NLS page is already in core, it will not read it from the rad when someone calls NLS, but will use the copy in core. So in order to make sure that the code for NLS has really been changed, everyone must get out of NLS. Note that the monitor does not know the difference between real code and junk, so if someone calls NLS and blows up, the monitor may still think that part of NLS is in core.

:DOCUMNTATN-FILE UTILITY, 03/24/69 H3E2:F7 CHI ; .DSN=1; .LSP=0;

Description

Functions

SAVE disc files on KDF

LOAD disc files from KDF

PRINT files

COMPILE files

Control File

A QED file used for input data

Contents

Use semi-colons as delimitors, with a period as a final delimitor(any thing after the first period will be ignored)

The first line must be: user-name;password;initials;user-name abreviation.

One line for each file to be procesed

File-name;(kdf user-name);compilation data if needed.

Compilation data consists of an ordered set of ordered pairs of commands and source statment numbers

the commands are MOL, SPL, ARPAS, and ARPDMP

For example, SPL;1;MOL;2.

The last line must be: END.

For an example of a control file, see (NLS):NLSFILES

Usage

Use teletype or nls station

A logged-out nls station must be available if you wish to print or compile

From executive level go to SNOBOL(push SN.)

READ FROM (NLS):FILE UTILITY(QED type commands are used in SNOBOL)

GO.--OK.(push G..)

Respond to "CONTROL FILE:" by typing the name of your control file, followed by a return

If the desired control file is :NLSFILES, just type a return

The program will then type that the file was opened and the number of files to be processed

(QUERY) The following will then be asked: "LOAD, SAVE, PRINT, COMPILE, OR DONE?"

Type the first letter of the function you wish to utilize

You will then be asked if you want all of the files in the control file proccessed

Yes, type "Y", and proccessing will begin

No, type "N"

Do you wish the files listed so that you can sellect those to be proccessed?

Yes, type "Y". Each file-name will then be typed, after which you should respond "Y" or "N"

No, type "N". You will then be requested to type the file-names of those files you wish proccessed, each followed by a return, with an additional return to terminate the list

The file-names so entered must correspond to an entry in the control file

If so, a colon will be typed for the next file-name

If not, an error message will be typed, and you will be allowed to re-enter a file-name(those previously entered are not lost)

If the process you sellected was PRINT or COMPILE, you will be asked to specify the number of an available logged-out nls station, followed by a return

When processing is complete, the program will return to its query: "LOAD, SAVE, PRINT, COMPILE, OR DONE?". see (QUERY)

Special Notes

This SNOBOL1 program logs in on another teletype. If none are available, an error message will be typed, and the program will be terminated(currently SNOBOL tries to log in on tty 15 or 8)

If the control file cannot be opened, the program will terminate with an error condition

If a syntax error is detected in the compilation of any part of a file, compilation will stop for that file and will begin for the next(an error message is output)

If printing files, be sure teletype 1 is not being used(leave a note on it so that no one will use it while you are in PRINT proccessing)

THE ON-LINE SYSTEM (NLS)

Introduction

NLS, as currently implemented, is essentially a highly sophisticated text-manipulation system oriented primarily toward on-line use; i.e., it is not primarily oriented toward production of hard copy, although fairly sophisticated hard-copy formatting and output are included in the system.

NLS is intended to be used on a regular, more or less full-time basis in a time-sharing environment, by users who are not necessarily computer professionals. The users are, however, assumed to be "trained" as opposed to "naive." Thus the system is not designed for extreme simplicity, nor for self-explanatory features, nor for compatibility with "normal" working procedures.

Rather, it is assumed that the user has spent considerable time in learning the operation of the system; that he uses it for a major portion of his work; and that he is consequently willing to adapt his working procedures to exploit the possibilities of full-time, interactive computer assistance.

Thus the practices and techniques developed by users for exploiting NLS are as much a subject of research interest as the development of NLS itself.

Work-Station Console

The user sits at a console whose main elements are a display screen, a typewriter keyboard, a cursor device called the "mouse," and a set of five keys operated by the left hand, called the "keyset.".

The screen is used for displaying text, in various formats. The top portion of the screen (approximately 1/5 of the total area) is reserved for feedback information of various kinds: the name of the user command mode currently in effect, a "register" area used for various kinds of feedback, an "echo register" which displays the last six characters typed by the user, and other items which are explained below.

The keyboard closely resembles a conventional typewriter keyboard, with a few extra keys for special characters and control functions. It is used for typing text as content for a file and for specifying commands, which are given as two- or three-character mnemonics.

The mouse is a roughly box-shaped object, about four inches on its longest side, which is moved by the right hand. It is mounted on wheels, and rolls on any flat surface. The wheels drive potentiometers which are read by an A/D converter, and

the system causes a tracking spot ("bug") to move on the screen in correspondence to the motion of the mouse.

The user specifies locations in the displayed text by pointing with the mouse/bug combination. This eliminates the need for specifying a location by entering a code of some kind. Use of the mouse is very easily learned and soon becomes unconscious.

On top of the mouse are three special control buttons, whose uses are described below.

The keyset has one key for each finger of the left hand. The keys are struck in combinations called "chords," and each chord corresponds to a character or combination of characters from the keyboard. There are 31 possible chords; beyond this, two of the buttons on the mouse may be used to control the "case" of the keyset, giving alternate meanings to each chord. There are four possible cases, for a total of 124 possible combinations.

A simple binary code is used, and has proved remarkably easy to learn. Two or three hours' practice are usually sufficient to learn the most commonly used chords and develop reasonable speed.

The keyset was developed to increase the user's speed and smoothness in operating NLS. It was found that users normally keep the right hand on the mouse, because the great majority of command operations involve a pointing action; efficient use of the keyboard, however, requires the use of both hands, and shifting the right hand (and the user's attention) to the keyboard is distracting and annoying if it must be done for each two- or three-letter command mnemonic.

Use of the keyset permits the user to keep his right hand on the mouse and his left on the keyset, reverting to the keyboard only for entry of long strings of text (typically five or more characters).

Originally, the keyset exactly duplicated the keyboard in function; in the development of NLS, however, certain control functions have been made two-stroke operations from the keyset where they would be three- or four-stroke operations from the keyboard. Nevertheless, it is still possible to operate all of the features of NLS without using the keyset; thus the beginner may defer learning the keyset code until he has gained some degree of mastery over the rest of the system.

Structured Text

"Text" is used here as a very general term. A "file" of text (corresponding roughly to a "document" in hard copy) may consist of English or some other natural language, numerical data, computer-program statements, or anything else that can be expressed as a structure of character strings. Simple line drawings can also be included in a file.

All text handled by NLS is in "structured-statement" form. This special format is simply a hierarchical arrangement of "statements," resembling a conventional "outline" form.

Each statement in a file may be considered to possess a "statement number," which shows its position and level in the structure. Thus the first statement in a file is Statement 1; its first substatement is 1A, and its next substatement is 1B; the next statement at the same level as the first is Statement 2; and so forth. Statement numbers have been suppressed in printing out most of this document, but are printed out for the remainder of this section as an example.

1c2b A statement is simply a string of text, of any length; this serves as the basic unit in the construction of the hierarchy. In English text, statements are normally equivalent to paragraphs, section and subsection headings, or items in a list. In other types of text, statements may be data items, program statements, etc.

1c2b1 Each paragraph and heading in this document is an NLS statement. Each statement is indented according to its "level" in the hierarchy; this paragraph is a substatement of the one above, which is in turn a substatement of another statement. A statement may have any number of substatements, and the overall structure may have any number of levels.

1c3 Note that when a user creates a file, he may let all of his statements be first-level ones, i.e. 1, 2, 3, etc. In this case he will not have to consider a hierarchical structure but simply a linear list, as is found in conventional text.

1c3a However, many of the features of NLS are oriented to make use of hierarchy, and the benefits of these features are lost if hierarchy is not exploited.

1c3b This is an example of an NLS feature to which the user must accomodate his methods; however, the experience of users has been that hierarchical structure very rapidly becomes a

completely "natural" way of organizing text. Many automatic features of NLS make the structure easy to use: for example, statement numbers are created automatically at all times and the user need not even be aware of them. It is sufficient, when the user creates a statement, to specify its level relative to the preceding statement.

Use of the System

Text manipulation is considered to involve three basic types of activity by the user: composition, study, and modification. In practice, the three activities are so intermingled as to be indistinguishable.

Composition

Composition is simply the creation of new text material as content for a file.

In the simplest case, the user gives the command "Insert Statement" by typing "is". He then points (with the mouse) to an existing statement; the system displays a new statement number which is the logical successor, at the same level, as the statement pointed to. The user may change the level of this number upward by typing a "u" or downward by typing a "d".

NOTE: Even if no previous statement has benn created, the system displays a "dummy" statement at the top of the text-display area, and the user points to this dummy.

The user then types the text of the new statement from the keyboard. On the screen, the top part of the text-display area is cleared and characters are displayed here as they are typed. When the statement is finished, the user hits a CA (command accept) button on the keyboard or mouse, and the system recreates the display with the new statement following the one that was pointed to.

New material may also be added to existing statements by means of commands such as Insert Word, Insert Text, and others. Properly speaking, these operations are modification rather than composition, and are discussed below.

Simple line drawings may be composed and added to the file by means of the "vector package." This is discussed in another section of this report.

Study

The study capabilities of NLS constitute its most powerful and unusual features. The following is only a brief, condensed description of the operations that are possible.

Jumping

NLS files may, of course, contain a great deal more text than can be displayed on the screen, just as a document may contain more than one page of text. An NLS file is thought of as a long "scroll." The process of moving from one point in the scroll to another, which corresponds to turning pages in hard copy, is called "jumping." There is a very large family of Jump commands.

The basic Jump command is Jump to Item. The user specifies it by entering "ji", and then points to some statement with the mouse. The selected statement is moved to the top of the screen, as if the scroll had been rolled forward.

Most of the Jump commands reference the hierarchical structure of the text. Thus Jump to Successor brings to the top of the display the next statement at the same level as the selected statement; Jump to Predecessor does the reverse; Jump to Up starts the display with the statement of which the selected statement is a substatement, and so forth.

The Jump to Name command uses a different way of addressing statements. If the first word of any statement is enclosed in parentheses, the system will recognize it as the "name" of the statement. Then, if this word appears somewhere else in the text, the user may jump to the named statement by pointing to the occurrence of the name, or by typing the name.

This provides a cross-referencing capability which is very smooth and flexible; the command Jump to Return will always restore the previous display, so that the user may follow name references without losing his place.

It is also possible to jump to a statement by typing its statement number.

View Control

If a file is long, it may be impossible for the user to orient himself to its content and structure or to find

specific sections by jumping through it. The principal solution to this problem is provided by level control and line truncation.

Level control permits the user to specify some number of levels; the system will then display only statements of the specified level or higher. Thus if three levels are specified, only first-, second-, and third-level statements are displayed.

Line truncation permits specification of how many lines of each statement are to be displayed. Thus if one line is specified, only the first line of each statement will be displayed.

Common usage is to use the first two or three levels in a file as headings describing the material contained under each heading in the form of substatements. Thus the user may start by looking at a display showing only the first-level statements in the file, one line of each. This amounts to a table of contents.

He may then select one of these statements and jump to it, specifying one more level. He will then see more details of the content of that part of the file. This process of "expanding the view" may be repeated until the user has found what he is looking for, at which point he may specify a full display of the text.

Users soon develop a habit of structuring files in such a way that this process will work well. As it happens, such a structure is usually a good, logical arrangement of the material, reflecting the relationships inherent in the content.

The level and truncation controls are designed so that the necessary specifications may be made with only one or two strokes of the keyboard or keyset. These controls are only the most important of a large set of view-control parameters called "VIEWSPECs." Other VIEWSPECs control a number of special NLS features affecting the display format.

Content Analysis

The NLS content analyzer permits automatic searching of a file for statements satisfying some content pattern specified made by the user. The pattern is written in a special language as part of the file text.

Content patterns may be simple, specifying the occurrence of some word, for example. They may also be highly complex, specifying the order of occurrence of two or more strings, the absence of some text construct, conditional specifications, etc. Simple patterns are extremely easy to write; complex ones are correspondingly more difficult.

Link Jumping

A "link" is a string of text, occurring in an ordinary file statement, which indicates a cross-reference of some kind. It may refer to another statement in the file, or to a statement in some other file, possibly belonging to another NLS user. The text of the link is both human-readable and machine-readable, and the command Jump to Link permits the user to point to the link with the mouse and immediately see the material referred to.

An example of a link is (Smith, Plans, Longrange:ebtng).

The first item in the link indicates that the referenced file belongs to a user named Smith; the second is the name of the file; the third is the name of a statement in the file (a statement number may also be used); and the string of characters following the colon controls the VIEWSPECs to set up a particular view of the material.

The use of interfile links permits the construction of large linked structures made up of many files, and study of these files as if they were all sections of a single document.

Modification

A large repertoire of editing commands is provided for modification of files. The basic functions are Insert, Delete, Move, and Copy.

These functions operate upon various kinds of text entities. Within statements, they may operate upon single characters, words, and arbitrary strings of text defined by pointing to the first and last characters.

This set of commands is not restricted to operation within one statement at a time; for example, a word may be moved or copied from one statement to another.

The editing functions also operate at the structural level, taking statements or sets of statements as operands. A number

of special entities have been defined for this purpose: for example, a "branch" consists of some specified statement, plus all of its substatements, plus all of their substatements, etc. A branch can be deleted, moved to a new position in the structure, etc.

As noted above, the modification activity tends to merge, in practice, with study and composition.

Summary

It must be noted that NLS is not a system designed for general usage, but a specialized tool designed for a group of people working on the development of computer aids to human intellectual processes. It is for this reason, for example, that NLS is not really a text-editing system oriented toward hard-copy production, but rather something simultaneously more general and more specialized.

It is in the process of manipulating a file -- studying it, making modifications, adding new material as an integrated process lasting for minutes or hours at a time and having a continuity extending for days, weeks, or even years -- that the real benefit of NLS appears.

An NLS file tends to become an evolving entity, subject to constant modification, updating, and reevaluation. Its development may have no clearly defined endpoint. It may cease to exist as a file by being incorporated in another file, or it may eventually be abandoned; however, it will probably never be "finished" in the usual sense of the word.

Continuous use of NLS to store ideas, study them, relate them structurally, and cross-reference them results in a superior organization of ideas and a greater ability to manipulate them further for special purposes, as the need arises -- whether the "ideas" are expressed as natural language, as data, as programming, or as graphic information.

CODES FOR KEYSET AND VIEW-CONTROL PARAMETERS

The keyset has four cases called 0, 1, 2, and 3. The center and left-hand buttons on the mouse are used for specifying case as follows: Case 0, neither button; Case 1, center button; Case 2, left button; Case 3, both buttons. The buttons are held down while striking chords on the keyset.

Case 0 contains lower-case letters, comma, period, semicolon, question-mark, and space.

Case 1 contains upper-case letters, less-than, greater-than, colon, backslash, and tab.

Case 2 contains various punctuation characters for chords 1-15, digits from 0 to 9 for chords 16-25, more punctuation for chords 26-29, and ALTMODE and carriage-return for chords 30 and 31.

Case 3 contains VIEWSPECs and centerdot.

Codes for Keyset and VIEWSPECs

Keyset Codes

CODE	Case 0	Case 1	Case 2	Case 3
00001	a	A	!	L=L-1
00010	b	B	"	L=L+1
00011	c	C	#	L=ALL
00100	d	D	$	L=1
00101	e	E	%	L relative
00110	f	F	&	recreate display
00111	g	G	'	branch-only on
01000	h	H	(	branch-only off
01001	i	I	)	content-analyzer on
01010	j	J	@	content-analyzer off
01011	k	K	+	trail on
01100	l	L	-	trail off
01101	m	M	*	statement numbers on
01110	n	N	/	statement numbers off
01111	o	O	↑	frozen statements on

CODE	Case 0	Case 1	Case 2	Case 3
10000	p	P	0	frozen statements off
10001	q	Q	1	T=T-1
10010	r	R	2	T=T+1
10011	s	S	3	T=ALL
10100	t	T	4	T=1
10101	u	U	5	pointers on
10110	v	V	6	pointers off
10111	w	W	7	L=T=ALL
11000	x	X	8	L=T=1
11001	y	Y	9	blank lines on
11010	z	Z	=	blank lines off
11011	,	<	[	(nothing)
11100	.	>	]	(nothing)
11101	;	:	←	(nothing)
11110	?	\	ALT	centerdot
11111	SP	TAB	CR	(nothing)

Capital-Letter VIEWSPECs

A -- indenting on

B -- indenting off

C -- names on

D -- names off

E -- clip picture and show (see vectors)

F -- show picture only if it fits (see vectors)

G -- display file as tree structure (see tree)

H -- display file as normal text (see tree)

I --keyword reordering on (see document on information-retrieval system)

J --keyword reordering off (see document on information-retrieval system)

K -- display of statement signatures on

L -- display of statement signatures off

Control of VIEWSPECs

The VIEWSPECs are used as parameters to control the way in which statements are displayed.

VIEWSPECs may be controlled in three ways: during certain commands such as Jump or Load, with the View Set command, or from the keyset in Case 3. (The viewspecs may also be set from the keyboard with the right-hand and center buttons on the mouse down, i.e. in Case 3 position.)

During the Jump and Load commands (and a few others), there is a point where the VIEWSPECs in the upper left-hand corner of the display become large and are accessible to change. They may then be changed by typing them in from the keyboard or keyset as upper- or lower-case letters.

The View Set command (q) may be used to achieve exactly the same effect.

Case 3 may be used to set all of the VIEWSPECs that are not capital letters, as shown in the table of keyset codes.

This may be done at any time.

After VIEWSPECs have been given in this fashion, it is necessary to hit Chord 00110, Case 3, for "new View," before the new VIEWSPECs will become effective.

Relative Level Control

The code "e" causes the subsequent setting of L to be interpreted relative to the level of the first statement in the new display.

For example, suppose that a Jump is being made to a third-level statement. If "e" is given with no subsequent codes for control of L, the result is L=3. If "eb" is given, the result is L=4; if "e3ba" is given, the result is L=5.

Thus it is possible to get an appropriate setting of L without knowing the appropriate absolute value, simply by specifying a relative value.

Six VIEWSPECs that are not self-evident are displayed as two lines in the upper left-hand corner of the screen.

The top line shows "L" and "T," which appear either as numbers or as the word "all." "L" determines how many levels of statements will be displayed and "T" determines how many lines of each statement will be displayed.

The second line shows four VIEWSPECs: g or h for the branch-only parameter on or off; i or j for content-analyzer on or off; k or l for trail feature on or off; and u or v for pointers displayed or not displayed.

View-Control Commands

Commands Affecting Display Start

Jump Commands

(jo) Jump to Origin

Syntax: j o VIEWSPEC CA

Semantics: The display start is positioned to the first statement.

(ji) Jump to Item

Syntax: j (i/null) [s] VIEWSPEC CA

Semantics: The display start is positioned to the selected statement. Note that the i in the command specification may be omitted.

(ju) Jump to Up

Syntax: j u [s] VIEWSPEC CA

Semantics: The display start is positioned to the source statement of the selected statement.

(jd) Jump to Down

Syntax: j d [s] VIEWSPEC CA

Semantics: The display start is positioned to the first substatement of the selected statement.

(js) Jump to Successor

Syntax: j s [s] VIEWSPEC CA

Semantics: The display start is positioned to the successor of the selected statement.

(jp) Jump to P redecessor

Syntax: j p [s] VIEWSPEC CA

Semantics: The display start is positioned to the predecessor of the selected statement.

(jh) Jump to Head

Syntax: j h [s] VIEWSPEC CA

Semantics: The display start is positioned to the first statement in the plex where the selected statement is found.

(jt) Jump to Tail

Syntax: j t [s] VIEWSPEC CA

Semantics: The display start is positioned to the last statement in the plex where the selected statement is found.

(je) Jump to E nd

This command expects a third letter. In each case, the command is similar to one of the other Jump commands, except that it considers a branch instead of a statement. The last statement of the branch is placed at the top of the display.

(jei) Jump to End of Item

Syntax: j e i [s] VIEWSPEC CA

Semantics: The selected statement determines a branch, and the last statement in that branch is placed at the top of the display.

(jen) Jump to End of Name

Syntax: j e n ([w]/SPACE LIT CA) VIEWSPEC CA

Semantics: The selected name determines a statement; the statement determines a branch, and the last statement in that branch is placed at the top of the display.

(jes) Jump to End of S uccessor

Syntax: j e s [s] VIEWSPEC CA

Semantics: The successor of the selected statement determines a branch, and the last statement in that branch is placed at the top of the display.

(jep) Jump to End of Predecessor

Syntax: j e p [s] VIEWSPEC CA

Semantics: The predecessor of the selected statement determines a branch, and the last statement in that branch is placed at the top of the display.

(jeu) Jump to End of Up

Syntax: j e u [s] VIEWSPEC CA

Semantics: The source of the selected statement determines a branch, and the last statement in that branch is placed at the top of the display.

(jed) Jump to End of Down

Syntax: j e d [s] VIEWSPEC CA

Semantics: The first substatement of the selected statement determines a branch, and the last statement in that branch is placed at the top of the display.

(jeh) Jump to End of Head

Syntax: j e h [s] VIEWSPEC CA

Semantics: The head of the plex containing the selected statement determines a branch, and the last statement in that branch is placed at the top of the display.

(jet) Jump to End of Tail

Syntax: j e t [s] VIEWSPEC CA

Semantics: The tail of the plex containing the selected statement determines a branch, and the last statement in that branch is placed at the top of the display.

(jeo) Jump to End of Origin

Syntax: j e o VIEWSPEC CA

Semantics: The last statement in the file is placed at the top of the display.

(jev) Jump to End of Vector Label

Syntax: j e v ([v]/SPACE LIT CA) VIEWSPEC CA

Semantics: This is identical to jen (Jump to End of Name) except that a vector label is selected instead of a word in text. The selected label or the LIT is used as a name and the end of the branch determined by the named statement is placed at the top of the display.

(jb) Jump to Back

Syntax: j b CA VIEWSPEC CA

Semantics: The display start is positioned to the statement immediately preceding the current display start.

(jn) Jump to Name

Syntax: j n ([w]/SPACE LIT CA) VIEWSPEC CA

Semantics: A statement name is specified by either a word-selection or a literal entry from the keyboard or keyset. When the command is executed, the statement with the specified name is placed at the top of the display.

If the specified name does not exist, the command is aborted with the message "no such name." If more than one statement with the specified name exists, the command is aborted with the message "duplicate name."

(jv) Jump to Vector Label

Syntax: j v ([v]/SPACE LIT CA) VIEWSPEC CA

Semantics: A statement name is specified by either a vector-label selection or a literal entry from the keyboard or keyset. When the command is executed, the statement with the specified name is placed at the top of the display.

If the specified name does not exist, the command is aborted with the message "no such name." If more than one statement with the specified name exists, the command is aborted with the message "duplicate name."

(jl) Jump to Link

NOTE: For important background information, see document LINKS AND RETURNS.

There are two cases of this command, depending on whether the link refers to a location in the current file or in another file (which must be a scratch file).

Syntax (within file): j 1 [1] VIEWSPEC CA

Semantics: The statement defined by the link is placed at the top of the display, and the VIEWSPECs given in the link are placed in effect, unless they are changed by manual input during the command. The new view is entered in the next location in the intrafile ring.

Syntax (out of file): j 1 [1] CA CA

Semantics: The statement defined by the link (in the file defined by the link) is placed at the top of the display, and the VIEWSPECs given in the link are placed in effect; they cannot be changed by manual input during the command. The new view is entered in the next location in the interfile stack.

(jr) Jump to Return

NOTE: For important background information, see document LINKS AND RETURNS.

Syntax: j r CA

Semantics: This command causes a return to the previous view; no change is made in the intrafile ring.

(ja) Jump to Ahead

NOTE: For important background information, see document LINKS AND RETURNS.

Syntax: j a CA

Semantics: This command (which can only be used meaningfully after jr has been used at least once) causes a move "forward" along the ring; no change is made in the ring itself.

(jf) Jump to File

Expects a third letter to specify jfl, jfr, jfa, jfw, or jfc.

NOTE: For important background information, see document LINKS AND RETURNS.

(jfl) Jump to File Link

Syntax: (intrafile link) j f l [1] VIEWSPEC CA

Syntax: (interfile link) j f l [1] CA CA

Semantics: The operation of this command is the same as Jump to Link.

(jfr) Jump to File Return

Syntax: j f r (CA/character) CA

Semantics: This command causes a move backward along the interfile stack, to the file previously viewed. No new entry is made in the stack.

When the characters j f r have been typed, the system will display the name of the file to be jumped to. If any character is typed instead of the CA, the system will go back one more step on the stack and display another filename.

(jfa) Jump to File Ahead

Syntax: j f a (CA/character) CA

Semantics: This command (which cannot be used meaningfully unless jfr has been used at least once) causes a move forward along the stack. No new entry is made in the stack.

When the characters j f a have been typed, the system will display the name of the file to be jumped to. If any character is typed instead of the CA, the system will go forward one more step on the stack and display another filename.

(jfw) Jump to File Working Copy

Syntax: j f w CA

Semantics: The WORKING COPY file is opened and displayed. The view will be the same as the last view of the WORKING COPY file. No new entry is made in the interfile stack.

(jfc) Jump to File Current

Syntax: j f c CA

Semantics: The "current" file is the one currently indicated by the pointer in the interfile stack. Usually the pointer indicates the file being displayed and this command is meaningless. However, when the working copy file is being displayed, this command is the means to return to the file previously displayed. No new entry is made in the stack.

Commands A ffecting Display List

(ec) Execute Content Analyzer

For a discussion of the content analyzer, see the document on the analyzer. This command is also used for the trail feature: see "trails" entry in document ODDS AND ENDS.

Syntax: e c [p] CA

Semantics: P is a pattern which specifies a content requirement for statements to be displayed. The pattern is compiled to produce a content-analyzer program which will cause only the statements meeting the requirement to appear on the display.

NOTE: This command does not put the content analyzer into action; it merely causes a pattern to be compiled. If the pattern has been incorrectly specified, the message "syntax error" will appear; otherwise, the message "successful compilation" will appear.

To put the content analyzer into effect, use the VIEWSPEC "i"; to turn it off use the VIEWSPEC "j". See document on Keyset and VIEWSPECs.

Freeze Commands

(fs) Freeze Statement

Syntax: f s [s] VIEWSPEC CA

Semantics: The selected statement is frozen, with the specified view. It will appear at the top of the screen whenever frozen statements are being shown.

(fr) Release Statement

Syntax: f r [s] CA

Semantics: The selected statement is unfrozen. The selection may be in the frozen area of the display or in the normal viewing area.

(fa) Release All

Syntax: f a CA

Semantics: All frozen statements are unfrozen.

Other View Control Commands

(ev) Execute Viewchange

NOTE: The syntax and semantics are given in a separate section, because of their complexity.

(q) View Set Commands

Syntax: q (a/ ... /z/A/ ... /L) CA

Semantics: The command makes the VIEWSPECs accessible for change, just as they are during a Jump command, for example.

sm (set mode)
(c/l/.....)

Set Commands

(sc) Set Character

Syntax: s c (c/l/i/r/b/n/f/s/u/w) CA [c] CA

Semantics: The selected character becomes upper-case, lower-case, etc. according to the following codes:

C -- capital

L -- lower-case

I -- italic

R -- roman

B -- boldface

N -- no boldface

F -- flickering

S -- solid (nonflickering)

U -- underline

W -- no underline

(sw) Set Word

Syntax: s w (c/l/i/r/b/n/f/s/u/w) CA [w] CA

Semantics: The selected word becomes upper-case, lower-case, etc. according to the codes given in sc.

(sv) Set Visible

Syntax: s v (c/l/i/r/b/n/f/s/u/w) CA [v] CA

Semantics: The selected visible becomes upper-case, lower-case, etc. according to the codes given in sc.

(si) Set Invisible

Syntax: s i (c/l/i/r/b/n/f/s/u/w) CA [i] CA

Semantics: The selected invisible becomes upper-case, lower-case, etc. according to the codes given in sc.

(ss) Set Statement

Syntax: s s (c/l/i/r/b/n/f/s/u/w) CA [s] CA

Semantics: The selected statement becomes upper-case, lower-case, etc. according to the codes given in sc.

(st) Set Text

Syntax: s t (c/l/i/r/b/n/f/s/u/w) CA [t]1 [t]2 CA

Semantics: The selected text becomes upper-case, lower-case, etc. according to the codes given in sc.

Delete Commands

(dc) Delete Character

Syntax: d c [c] CA

Semantics: The selected character is deleted.

(dw) Delete Word

Syntax: d w [w] CA

Semantics: The selected word is deleted.

(dv) Delete Visible

Syntax: d v [v] CA

Semantics: The selected visible string is deleted.

(di) Delete Invisible

Syntax: d i [i] CA

Semantics: The selected invisible string is deleted.

(dt) Delete Text

Syntax: d t [c]1 [c]2 CA

Semantics: The selected text string is deleted from c1 to c2.

(ds) Delete Statement

Syntax: d s [s] CA

Semantics: The selected statement is deleted. If it has any substatements, the deletion is illegal; however, the statement and its substructure may be deleted with Delete Branch.

(db) Delete Branch

Syntax: d b [s] CA

Semantics: The selected branch (defined by Statement s) is deleted.

(dp) Delete Plex

Syntax: d p [s] CA

Semantics: The selected plex (defined by Statement s) is deleted.

(dg) Delete Group

Syntax: d g [s]1 [s]2 CA

Semantics: The selected group (defined by Statements s1 and s2) is deleted.

(dd) Delete Drawing

Syntax: d d [s]1 CA

Semantics: The drawing associated with the selected statement is deleted.

Insert Commands

(ic) I nsert Character

Syntax: i c [c] LIT CA

Semantics: LIT is inserted immediately after the selected character.

(iw) Insert Word

Syntax: i w [w] LIT CA

Semantics: LIT is inserted after the selected word, with a SPACE between.

(iv) Insert Visible

Syntax: i v [v] LIT CA

Semantics: LIT is inserted after the selected visible, with a SPACE between.

(it) Insert Text

Syntax: i t [c] LIT CA

Semantics: LIT is inserted after the selected character. The action is identical to (ic).

(ii) Insert I nvisible

Syntax: i i [I] LIT CA

Semantics: LIT is inserted immediately after the selected invisible string.

(is) Insert Statement

Syntax: i s [s] LEVADJ SPACE LIT CA

Semantics: LIT becomes the text of a new statement (or set of statements ---see "centerdot" entry in document ODDS AND ENDS), following the selected statement at a level determined by the LEVADJ.

NOTE: The new statement does not necessarily follow the selected statement directly. See "LEVADJ" entry in document ODDS AND ENDS.

(ib) Insert Branch

Syntax: i b [s] LEVADJ SPACE LIT CA

Semantics: The action of this command is identical to that of Insert Statement.

(ip) Insert Plex

Syntax: i p [s] LEVADJ SPACE LIT CA

Semantics: The action of this command is identical to that of Insert Statement.

(ig) Insert Group

Syntax: i g [s] LEVADJ SPACE LIT CA

Semantics: The action of this command is identical to that of Insert Statement.

(iq) Insert QED Branch

Syntax: i q [s] (Y/(not Y)/CA) FILENAME CA

Semantics: Note that no LEVADJ is possible.

In response to the feedback message CONVERT CASE, a Y or CA means YES; any other character means NO.

If YES, all characters will be read as lower-case except those immediately preceded by a backslash: these will be upper-case and the backslashes will disappear.

If NO, all letters will be read by NLS as upper-case letters; backslashes will be taken literally.

The text of the QED file becomes a branch following the selected branch, at the same level. Normally, if the QED file was not created from NLS with the odq command, there will be some garbage in the new branch.

Replace Commands

(rc) Replace Character

Syntax: r c [c] LIT CA

Semantics: The selected character is replaced by LIT.

(rw) Replace Word

Syntax: r w [w] LIT CA

Semantics: The selected word is replaced by LIT.

(rv) Replace Visible

Syntax: r v [v] LIT CA

Semantics: The selected visible string is replaced by LIT.

(rt) Replace Text

Syntax: r t [c]1 [c]2 LIT CA

Semantics: The selected text is replaced by LIT.

(ri) Replace Invisible

Syntax: r i [i] LIT CA

Semantics: The selected invisible string is replaced by LIT.

(rs) Replace Statement

Syntax: r s [s] LIT CA

Semantics: The text of the selected statement is replaced by LIT (see "centerdot" entry in document ODDS AND ENDS).

(rb) Replace Branch

Syntax: r b [s] LIT CA

Semantics: The branch defined by the selected statement is replaced by LIT (see "centerdot" entry in document ODDS AND ENDS).

(rp) Replace Plex

Syntax: r p [s] LIT CA

Semantics: The plex defined by the selected statement is replaced by LIT (see "centerdot" entry in document ODDS AND ENDS).

(rg) Replace Group

Syntax: r g [s]1 [s]2 LIT CA

Semantics: The group defined by the selected statements is replaced by LIT (see "centerdot" entry in document ODDS AND ENDS).

Move Commands

(mc) Move Character

Syntax: m c [c]1 [c]2 CA

Semantics: Character c2 is moved so that it appears immediately after Character c1.

(mw) Move Word

Syntax: m w [w]1 [w]2 CA

Semantics: Word w2 is moved so that it appears immediately after Word w1, with spaces and punctuation as they should be.

(mv) Move Visible

Syntax: m v [v]1 [v]2 CA

Semantics: Visible v2 is moved so that it appears immediately after Visible v1, with spaces as they should be.

(mt) Move Text

Syntax: m t [c]1 [c]2 [c]3 CA

Semantics: Characters c2 and c3 are the first and last characters of a text string; the string is moved so that it immediately follows c1.

(mi) Move Invisible

Syntax: m i [i]1 [i]2 CA

Semantics: The string i2 is moved to follow i1.

(ms) Move Statement

Syntax: m s [s]1 [s]2 LEVADJ CA

Semantics: Statement s2 is moved so as to follow Statement s1, at a level determined by the LEVADJ. If Statement s2 has any substatements, the move is illegal; however, s2 and all of its substructure can be moved together with Move Branch.

NOTE: Statement s2 does not necessarily follow Statement s1 directly. See "LEVADJ" entry in document ODDS AND ENDS.

(mb) Move Branch

Syntax: m b [s]1 [s]2 LEVADJ CA

Semantics: The branch headed by Statement s2 is moved so as to follow the branch headed by Statement s1, at a level determined by the LEVADJ.

See note under Move Statement.

(mp) Move Plex

Syntax: m p [s]1 [s]2 LEVADJ CA

Semantics: The plex defined by Statement s2 is moved so as to follow Statement s1, at a level determined by the LEVADJ.

See note under Move Statement.

(mg) Move Group

Syntax: m g [s]1 [s]2 [s]3 LEVADJ CA

Semantics: The group of branches defined by Statements s2 and s3 is moved so as to follow Statement s1, at a level determined by the LEVADJ.

See note under Move Statement.

(md) Move Drawing

Syntax: m d [s]1 [s]2 CA

Semantics: The drawing associated with s2 is moved to s1.

Copy Commands

(cc) Copy Character

Syntax: c c [c]1 [c]2 CA

Semantics: Character c2 is copied immediately after Character c1.

(cw) Copy Word

Syntax: c w [w]1 [w]2 CA

Semantics: Word w2 is copied immediately after Word w1.

(cv) Copy Visible

Syntax: c v [v]1 [v]2 CA

Semantics: Visible v2 is copied immediately after Visible v1.

(ci) Copy Invisible

Syntax: c i [i]1 [i]2 CA

Semantics: Invisible v2 is copied immediately after Invisible v1.

(ct) Copy Text

Syntax: c t [c]1 [c]2 [c]3 CA

Semantics: Text from Character c2 to Character c3 is copied immediately after Character c1.

(cs) Copy S tatement

Syntax: c s [s]1 [s]2 LEVADJ CA

Semantics: Statement s2 is copied after Statement s1, at a level determined by the LEVADJ.

NOTE: Statement s2 does not necessarily follow Statement s1 directly. See "LEVADJ" entry in document ODDS AND ENDS.

(cb) Copy Branch

Syntax: c b [s]1 [s]2 LEVADJ CA

Semantics: The branch determined by Statement s2 is copied after Statement s1, at a level determined by the LEVADJ.

See note under Copy Statement.

(cp) Copy Plex

Syntax: c p [s]1 [s]2 LEVADJ CA

Semantics: The plex determined by Statement s2 is copied after Statement s1, at a level determined by the LEVADJ.

See note under Copy Statement.

(cg) Copy Group

Syntax: c g [s]1 [s]2 [s]3 LEVADJ CA

Semantics: The group determined by Statements s2 and s3 is copied after Statement s1, at a level determined by the LEVADJ.

See note under Copy Statement.

(cd) Copy Drawing

Syntax: c d [s]1 [s]2 CA

Semantics: The drawing associated with s2 is copied to s1.

Break and Join Commands

(bs) Break Statement

Syntax: b s [v] LEVADJ CA

Semantics: The statement is broken after the selected visible. The LEVADJ adjusts the level of the new statement made up of the last part of the original statement. This new statement follows the original statement.

NOTE: The second statement does not necessarily follow the first statement directly. See "LEVADJ" entry in document ODDS AND ENDS.

(bj) Join Statement

Append (as)

Syntax: b j [s]1 [s]2 CA

Semantics: The text of s2 is appended to s1, and s2 is deleted. If s2 has a sublist, the sublist is moved to follow s1.

Output Commands

(o) Output Checkpoint

Syntax: o CA

Semantics: The current file is written out on the "CHECKPOINT" file, which is automatically created if necessary.

(of) Output File

Syntax: o f ([filename]/null) CA CA

Semantics: The filename must be must be that of a scratch file; if it is an existing file, it must be sequential. The current file is written out on the specified file.

The system will offer the name of the last file the current file has been written on from NLS. If this is the desired filename, the user does not need to type anything except o f CA CA.

(od) Output Device

Expects a third letter to specify the "device."

The Output Device commands do not necessarily output the entire current file. Output begins with the display start and is subject to any VIEWSPECs that work at the statement-structure level.

The options are as follows:

(odq) Output QED File

Syntax: o d q (LIT/[filename]) CA

Semantics: The LIT or filename specifies a file. Material from the current file is output through Pass 4 in QED format. Note that lower-case letters will come through as garbage, and that directives will be executed.

Pass 4 is automatically initialized with the following directives: PSW=0, IND=0, RTJ=0, AND HSW=0.

The origin should not be on the display when ODQ is executed.

(odp) Output Printer File

Syntax: o d p (LIT/[filename]) CA

Semantics: The LIT or filename specifies a file. Material from the current file is output through Pass 4 in printer format.

This printer file may then be output to the printer itself by using the program PASS4 KLUDGE PRINT from a Teletype.

(odd) Output Dura File

Syntax: o d d (LIT/[filename])

Semantics: The filename "8-level" causes material from the current file to be output on punched paper tape.

(odf) Output Film File

Syntax: o d f (LIT/[filename])

Semantics: The LIT or filename specifies a file. Material from the current file is output through Pass 4 in correct format for CRT-to-film processing.

(om) Output MOL

Syntax: o m 2$2(Y/CA OR NOT Y/CA) LIT/[filename] CA

Semantics: MOL source code from the displayed file is output to the MOL compiler, which in turn writes object (assembly) code on a designated file.

Output begins with the display start and is subject to any VIEWSPECs that work at the statement-structure level.

Two optional choices are offered with this command.

When the M is typed, the word "reentrant" appears in the command Feedback Line. If reentrant code is desired, the word is accepted with a CA or a Y (for "yes"). Any other character causes the word "reentrant" to be rejected and disappear.

Next the word "temps" appears, referring to generation of temporary storage. If temporary storage is to be generated, the word is accepted with Y or CA; otherwise it is rejected with any other character.

Finally a file name is requested and entered either as a literal or by a bug selection.

(os) Output SPL

Syntax: o s LIT/[filename] CA

Semantics: SPL source code from the displayed file is output to the SPL compiler, which in turn writes object (assembly) code on the designated file.

Output begins with the display start and is subject to any VIEWSPECs that work at the statement-structure level.

(ot) Output Meta

Syntax: o t LIT/[filename] CA

Semantics: Tree Meta source code from the displayed file is output to the Tree Meta compiler, which in turn writes object (assembly) code on the designated file.

Output begins with the display start and is subject to any VIEWSPECs that work at the statement-structure level.

Load Commands

(l) Load Checkpoint

Syntax: l CA VIEWSPEC CA

Semantics: The CHECKPOINT file is copied to the WORKING COPY file and displayed, beginning at the first statement.

No change is made to the interfile stack.

Please see document LINKS AND RETURNS.

(lf) Load File

Syntax: l f [filename] CA VIEWSPEC CA

Semantics: The file specified by the filename is opened and displayed, beginning at the first statement.

A new entry is made in the interfile stack, which is used by the Jump to File Return, Jump to File Ahead, and Jump to File Current commands. For details, see document LINKS AND RETURNS.

(v) Vector P ackage

Syntax: v [s] CA

Semantics: The specified statement is placed at the top of the screen and the rest of the screen is cleared. The commands in the vector package may then be used to create or modify a picture attached to the statement. See document on Vector Package.

Keyword Commands

The keyword commands are described in the document on the Keyword Information--Retrieval System.

Alarm Commands

(ac) Alarm Clock Set

Syntax: a c NUM1 CA NUM2 CA NUM3 CA

Semantics: NUM1, NUM2, and NUM3 are the hour, minute, and second to which the alarm clock is set.

(at) Alarm Timer Set

Same as (ac) but relative to present time.

Pointer Commands

(pf) Pointer Fix

Syntax: p f [c] 1$3(CHAR) CA

Semantics: The pointer name (up to 3 characters) is attached to the specified character.

Please see entry on pointers in document ODDS AND ENDS.

(pl) Pointer List Show

Syntax: p l

Semantics: As soon as the command is specified, the display is replaced by a list of all the pointers that have been defined. To return to the display, enter a CA.

(pr) Pointer Release

Expects a third letter to specify an entity. The options are as follows:

(pra) Pointer Release All

Syntax: p r a CA

Semantics: All pointers are deleted throughout the file.

(prs) Pointer Release S tatement

Syntax: p r s [s] CA

Semantics: All pointers in the selected statement are deleted.

(prw) Pointer Release Word

Syntax: p r w [w] CA

Semantics: All pointers in the selected word are deleted.

(prt) Pointer Release Text

Syntax: p r t [c]1 [c]2 CA

Semantics: All pointers in the selected text are deleted.

(eo) Execute OOP S

Syntax: e o CA

Semantics: This command is for experimental use. It is a way of deliberately making NLS blow up on you.

Whenever an OOPS occurs (naturally or on command), a system file is automatically written which contains useful information for system programmers.

(ef) Execute File Check

Syntax: e f CA

This causes rapid checking of the file (by checksumming) to see if there are any data or structure errors. An appropriate message is displayed.

(eg) Execute File Grope

Syntax: e g CA

Semantics: This will clean out certain invisible problems in a file and produce a display of messages showing what kinds of errors have been found. File Grope takes much longer to run than File Check.

IMPORTANT NOTE: File Grope does not always correct all the errors it finds. Occasionally, it will lose file material in the process of attempting to correct a structural error. This command should not be used on a routine basis.

(es) Execute S tatus

Syntax: e s CA

Semantics: As soon as the s is typed, the display shows various items of information about the file. When the CA is hit, the normal display is restored.

(ed) Execute Declare File Ownership

Syntax: e d CA

Semantics: As soon as the CA is typed, the information in the file that gives the file owner's name is changed; the new name will be that under which the user entered NLS.

THE NLS CONTENT ANALYZER

1. Introduction

The content analyzer feature of NLS permits the user to specify (in a special language) a pattern of content. The analyzer is compiled in real time from the user's specification, and when it is turned on (by a VIEWSPEC parameter) only statements which meet the content specification will appear on the display.

The pattern specified may be a simple one -- e.g., it may specify a string of characters that must appear somewhere in each statement to be displayed; or it may be complex -- e.g., it may specify a string, to be followed within a given number of words by another specified string, in statements which were created after a certain date by a certain author, and not containing some third specified string.

The language for specifying content patterns is simple and easy to use for simple cases, but more exacting for complex cases.

2. Pattern-Specification Language

a. The Process of Searching a Statement

When the content analyzer is turned on, each statement in the file is searched, character by character, for the content specified in the pattern. Normally, the search begins with the first character, but it is possible to cause the search to proceed backwards.

The analyzer uses a pointer to keep track of the search. The pointer always indicates which character is to be examined next, unless something in the pattern causes the pointer to be moved first.

At any given moment in the search process, the analyzer is searching for one of four types of content entity:

A literal string of characters, such as "abcd" or "13-x" or "ed Mat" or "memory."

A string of character-class variables; these are explained in detail further on. A string of character-class variables might specify "three digits, one after another," or "two letters, followed by any number of spaces, followed by three to five letters or digits."

The date associated with the statement. (This is not normally displayed, but every statement bears the date on which it was created or most recently modified.)

The initials associated with the statement. (This is not normally displayed, but every statement bears the initials of the user by whom it was created or most recently modified.)

All of the more complex analysis is achieved by moving the pointer according to the logic of the pattern specification.

For example, if the analyzer is to start at a given point and find either String A or String B, it first looks for string A; if String A is not found, the pointer is returned to the starting point, and a search is made for String B.

b. Basic Elements

Every pattern ends with a semicolon.

Every pattern is made up of one or more of the basic entities listed above, combined by operators.

If the pattern (or some part of it) is to be found anywhere after the point in the statement where the search begins, it is enclosed in square brackets; otherwise it must be the first thing found.

A string of characters specified as content is enclosed in quotation marks. For convenience, if the string consists of only one character, it may be preceded by an apostrophe and the quotation marks omitted.

Examples

["memory"]; This pattern will cause display of only those statements containing the word "memory" at any point.

"inside"; This pattern will cause only statements beginning with the word "inside" to be displayed.

['3]; This pattern will cause display of only those statements containing the character "3" at any point.

Patterns like those shown in the examples above may be strung together; the significance of this is that one item is to be found after the one specified ahead of it.

Examples

["abc""def"]; This pattern specifies that the string abc

immediately followed by the string def must appear somewhere in each statement to be displayed. The pattern ["abcdef"]; is exactly equivalent.

["abc"]["def"]; This pattern specifies that the string abc is to be found anywhere in the statement, and anywhere after the "c" the string def is to be found.

c. Character-Class Variables

The character-class variables are as follows:

L means any letter

D means any digit

LD means any letter or digit

PT means any printing character (any character except space, tab, and carriage return)

SP means a space

TAB means a tab

CR means a carriage return

NP means any nonprinting character (space, tab, or carriage return)

CH means any character at all.

Examples

['.LLL'=D';]; This pattern will cause display of only those statements containing (anywhere) the following content: a period immediately followed by three letters, immediately followed by an equals sign, immediately followed by a digit, immediately followed by a semicolon.

"abcd"SPL D; This will cause display of only those statements beginning with the following content: the string abcd immediately followed by a space, immediately followed by any letter, immediately followed by any digit.

Note that a space is necessary between the L and the D because of a possible ambiguity: The pattern "abcd"SPLD; would mean "the string abcd immediately followed by a space, immediately followed by any letter or digit,"

because LD means any letter or digit.

d. The Dollar Sign (Arbitrary-Number Construct)

The arbitrary-number construct, in its most general form, is m$n. The meaning is "any number from m to n of occurrences of the following entity."

When the analyzer has found n occurrences of the specified entity, it also looks ahead to see if there is another occurrence. If there is, the test is considered to have failed. In other words, the limits m and n are absolute.

Example

The pattern 5$11LD; specifies that each staement to be displayed must begin with five to eleven letters and/or digits.

A statement beginning with twelve or more letters and/or digits would be rejected by this pattern.

The m or the n, or both, may be omitted; their assumed values in this case are m=0, n=1000. For all practical purposes, then, the default value of n is "any arbitrary number," since it is very unlikely that any entity will occur 1000 times consecutively.

E xamples

The pattern [7$D1$12L$5NP]; specifies that each statement to be displayed must contain the following: seven or more digits immediately followed by one to twelve letters, immediately followed by zero to five nonprinting characters.

The pattern 2$"abc"; specifies that each statement to be displayed must begin with two or more occurrences of the string abc, one after another.

e. Grouping by Parentheses

Parentheses may be used as they are in algebra to group elements. The specifications found within the parentheses are then treated as a single entity for logical purposes.

Example

[3$4(DSPL)1$2NP]; This pattern specifies that each

statement to be displayed must contain the following: three or four occurrences of the string (digit space letter), immediately followed by one or two nonprinting characters.

If the parentheses were not used, the 3$4 construct would apply only to the D.

The square brackets have the same grouping effect as parentheses; however, they are not interchangeable with parentheses because they also mean that the enclosed pattern may be found anywhere after the starting point.

f. Operators

The operators used for combining entities are as follows, in order of decreasing precedence (see note on precedence, below):

- (minus sign): This indicates negation. Thus -LD means a character which is not a letter or a digit.

Example: ["abc"-SP]; This pattern specifies that each statement to be displayed must contain the string abc immediately followed by some character which is not a space.

(space): This indicates concatenation. Thus "abc" "xyz"; specifies that the string abc must occur and must be immediately followed by the string xyz.

The space may be omitted unless it is necessary to prevent ambiguity. Thus "abc" "xyz"; could also be written "abc""xyz";

/ (slash): This indicates alternation. Thus SP/TAB means a character that may be either a space or a tab.

Example: 1$SP/2$3PT; This pattern specifies that each statement to be displayed must begin with either one or more spaces, or two or three printing characters.

NOT: This indicates negation, and is the same as the minus sign except for lower precedence.

AND: This is logical intersection.

The action of the AND is to return the pointer to the beginning of the search that has just been completed.

Example: The pattern ["abc"]AND["xyz"]; causes each statement to be searched first (from the beginning) for the string abc; then, if it is found, the statement is searched again from the beginning for the string xyz. Each statement displayed will contain both strings, but the order in which they occur will be irrelevant.

Note that this is different from the pattern ["abc"]["xyz"]; because if the AND is not used, the second search is not made from the beginning but from the point just after the end of the first search. Each statement displayed will then contain both strings, but the string xyz must be somewhere after the string abc. When the AND is used, this restriction will not apply.

Note also that the pattern ["abc"AND"xyz"]; is meaningless: it specifies a string that is both "abc" and "xyz".

OR: This is the same as the slash sign except for the lower precedence.

Note on Precedence of Operators: As used here, "high precedence" means that when the pattern is parsed, the higher-precedence operators are used first in grouping the elements of the pattern. Thus a high-precedence operator has low "binding power."

Example: Consider the pattern a AND b OR c/-d AND NOT e f; where a, b, c, d, e, and f are pattern elements such as quoted strings or character-class variables.

This is grouped as follows:

The minus sign has the highest precedence, so that we have a AND b OR c/(-d) AND NOT e f;

Next is concatenation, so we have a AND b OR c/(-d) AND NOT (e f);

Next is the slash, so we have a AND b OR (c/(-d)) AND NOT (e f);

Next the NOT, giving a AND b OR (c/(-d)) AND (NOT (e f));

Finally, the AND gives (a AND b) OR ((c/(-d)) AND (NOT (e f)));.

g. Dates and Initials

The dates and initials associated with each statement may be tested with the constructs .SINCE, .BEFORE, .INITIALS=, and .INITIALS#. (The symbol # is used to mean "not equal.")

The .INITIALS construct requires the following format:

.INITIALS=ABC where the string ABC is a user's initials (three initials must be given).

The .SINCE and .BEFORE constructs require the following format:

.SINCE (68/10/12 13:14) where 68 is the year, 10 is the month, 12 is the day, 13 is the hour, and 14 is the minute. The time may be eliminated by using 0:0.

Examples

.BEFORE (67/3/22 15:15) AND .SINCE (67/1/12 12:00); This pattern will cause display of only those statements bearing dates between noon of 12 January 1967 and 3:15 PM of 22 March 1967.

.SINCE (68/10/10 0:0) AND .INITIALS#DGC; This pattern will cause display of only those statements bearing dates later than 10 October 1968 and not bearing the initials DGC.

i. Special Control of Search

The position of the search pointer can be stored and set, and the direction of search can be controlled, in order to achieve complex effects. These effects also involve the use of the IF construct (described further on), and the possibilities have been explored only superficially at present. It should be possible to create pattern expressions of great complexity which would resemble sophisticated data-processing or information-retrieval programs, but at present the techniques have not been worked out.

The position of the pointer may be stored in any one of nine buffers, P1 ... P9. This is done by writing ↑Pn, where n is some digit from 1 to 9.

The stored value in the buffer can then be decremented by writing ←Pn. The reason for doing this is that when the analyzer has found some entity, the pointer is moved to the next character position; in order to store the value

of the last character actually searched, then, it is necessary to write ↑Pn←Pn.

The search pointer can then be set to the value in a buffer by writing Pn.

The search pointer can also be set to the beginning or end of a statement by writing SF(Pn) for the beginning and SE(Pn) for the end.

Note that SF and SE are functions which require a buffer value as argument; buffer values are not reinitialized after a statement has been scanned but continue to indicate the same character in the statement they were originally set to. Thus it is possible for a search to cover more than one statement.

The normal direction of scanning may be reversed by writing a less-than sign (<) and returned to the forward direction by writing a greater-than sign (>).

The left-arrow (←) used for decrementing a buffer value will increment it instead if the current scan direction is backward. Thus the effect will always be the same -- the buffer value will indicate the character just scanned.

Example

↑P1 SE(P1) < $NP -'.; This pattern causes statements to be searched backwards from the end. Only statements whose last printing character is not a period will be displayed.

The construct "↑P1" at the beginning of the pattern causes the current pointer position (which indicates the beginning of the statement) to be stored. This is simply for the purpose of having an argument for the "SE(P1)" construct, which causes the pointer to be positioned to the end of the statement. The less-than sign then causes the scan to proceed backwards; any number of nonprinting characters will be permitted, and then a character which is not a period is specified.

j. The IF Construct

The IF construct has the following format:

(IF relat THEN exp1 ELSE exp2)

where "relat" is a relationship between two buffer values and exp1 and exp2 are pattern expressions.

The possible relationships are as follows:

.EQ (equals)

.NE (not equal to)

.LT (less than)

.LE (less than or equal to)

.GT (greater than)

.GE (greater than or equal to).

If the specified relationship is true, exp1 is used for a test; if it is not true, exp2 is used.

Example

↑P1 SE(P1) < (['e] ↑P2←P2 AND ['t] ↑P3←P3) (IF P2 .LT P3 THEN SF(P1) > $SP "The" ELSE SF(P1) > [" if "]); This pattern imposes the following condition on statements to be displayed: If the last "e" precedes the last "t", then the first word in the statement must be "The". Otherwise, the statement must contain the word "if", enclosed by spaces. The proof is left to the reader.

k. The .EMPTY Construct

Whenever the analyzer makes a test, a flag is set true or false. After a statement has been tested by the complete pattern, it is displayed if the flag is true and omitted if the flag is false.

The construct .EMPTY simply sets the flag true. Conversely, the construct NOT .EMPTY (or -.EMPTY) sets the flag false.

This is useful in the IF construct, where one may simply wish to test the relationship without imposing further tests.

Example

↑P1 SE(P1) < (['e] ↑P2←P2 AND ['t] ↑P3←P3) (IF P2 .LT P3 THEN NOT .EMPTY ELSE .EMPTY); This pattern is similar to

the previous example, but slightly simpler. The condition is that if the last "e" in the statement does not precede the last "t", the statement will be displayed; otherwise it will not.

The "↑P1" stores the pointer value, which inicates the beginning of the statement. The "SE(P1)" sets the pointer to the end of the statement, and the "<" causes a backward scan. An "e" is found and its position stored in P2; then a "t" is found and its position stored in P3. The IF construct compares the values of P2 and P3: if P2 is less than P3 (i.e. if the "e" precedes the "t" in the statement), the "NOT .EMPTY" takes effect and the flag is set false, so the statement will not be displayed; if P2 is not less than P3, the ".EMPTY" takes effect, the flag is set true, and the statement is displayed.

3. Procedure for Using Content Analyzer

A pattern may be written as text anywhere in a file. A file may thus contain any number of patterns; however, only one pattern may be compiled at a time -- i.e., when a new pattern is compiled the code created by the previous one is lost.

To compile a pattern, the command Execute Content Analyzer is used. The syntax is

```
ec [c1] CA
```

where [c1] means that a character is selected either with the mouse or by means of a pointer call, and CA means that a Command Accept key is struck.

The character selected must be either the first character of the pattern or a nonprinting character preceding the pattern, with no printing characters intervening.

Note that the last part of a pattern may thus be used as a separate pattern, if it is meaningful.

The screen will go momentarily blank with a message. If the pattern has been compiled, the message is "successful compilation"; if the pattern has an error in it which prevents it from compiling, the message is "syntax error."

Syntax errors are frequently caused by inadvertent omission of some character such as a quotation mark. Another common cause for a syntax error or a compiled pattern that does not work as

expected is an error in the way that parts of the pattern are grouped. In the latter case, the problem may often be solved by insertion of parentheses.

When the pattern has been compiled, it will not go into effect until the view-control parameter "i" is placed in effect. When this has been done, the system will display only statements which fit the pattern.

Testing of statements begins with the statement currently designated as the display start; other statements are then tested in the order in which they would appear "normally," i.e. with the analyzer off. Any other view specifications which are in effect continue to work; thus if only first- and second-level statements are being displayed, only first- and second-level statements will be tested by the analyzer.

Statements are tested until the display screen has been filled. If no statements are found that fit the pattern, the screen goes blank with the message "empty" and remains so until the analyzer is turned off or until changed view-control parameters make it possible to find a statement that fits the pattern.

Whenever the display is re-created, the testing process is repeated. Thus if a statement is edited, and the editing changes it so that it no longer fits the pattern, it will disappear from the screen.

THE INFORMATION-RETRIEVAL SYSTEM

1. Introduction

The information-retrieval system permits a user to construct a specially formatted "catalog" file, containing references to other files and capable of being reordered automatically according to some chosen set of weighted keywords. When reordered, the file lists references in order of relevance according to the choice and weighting of keywords.

Any set of statements in a file may be reordered with this system, assuming that each statement has a "name" (parenthesized first word). The specifics given in this appendix refer to the most basic way of using the system.

2. The Catalog File

The catalog file has two functioning sections: a list of file references pointing to other files, and a list of relevant keywords to be used in retrieving file references.

Other material may also be included in the file without any effect on the functioning of the retrieval system. For example, since the keyword section is to be studied directly by the user, it may be desirable to group the keyword entries into categories and separate them with headings and subheadings.

a. File-Reference Section

Each file reference is a separate statement beginning with a serial number in parentheses, followed by a link pointing to the referenced file. This is followed by a list of keywords relating to the file, followed by comments on the file.

Only the first item is actually essential to the working of the system, and it need not actually be a serial number; any string of letters and/or digits enclosed in parentheses (i.e., a "statement name" as recognized by NLS) will suffice, as long as it is unique to the particular reference.

The use of serial numbers as "names" in file-reference statements, and the inclusion of the other items, are matters of convenience to the user.

b. Keyword Section

Each keyword must be a single word -- i.e., it must contain no nonprinting characters. Apart from this, it may be any arbitrary string of characters. It is convenient to use short strings of three or four letters standing for longer words or phrases.

Each entry in the keyword section is a separate statement with the following format: first the keyword itself, in parentheses, serving as the name of the statement; then the word or phrase for which it stands, plus any comments or other information that may be desired; and finally a special code string (such as an asterisk or a dollar sign followed by a space) followed by a list of serial numbers which are the names of statements in the file-reference section. Each of these serial numbers must be enclosed in parentheses.

Examples of a short catalog file and of how it might be reordered are given at the end of this appendix.

3. Keyword Commands

This section explains the effects of the keyword commands. Full details on the syntax and control-dialog procedures may be found in the NLS User's Guide.

The keyword commands operate upon the keywords themselves, i.e., the names of statements in the keyword section of the catalog. The commands permit the user to select keywords as relevant; assign integer weights to them; change weights; display a list of keywords that have been selected, with their weights; and produce an ordered display of the relevant file references.

a. K eyword S elect Command

This command is used to select a given keyword as relevant. It is automatically assigned a weight of 1.

b. Keyword Weight Command

When a keyword is selected under this command, its current weight is displayed (if it has not been previously selected, its weight is zero). The user may then type in an integer which becomes the new weight.

c. Keyword List Command

This command causes display of a list of keywords with nonzero weights.

d. Keyword List Weight Command

This is the same as the "List" command except that the weights are shown.

e. K eyword Forget Command

When a keyword is selected under this command, its weight is reset to zero, just as if it had never been selected.

f. Keyword Forget All Command

This command causes all keyword weights to be reset to zero.

g. Keyword Execute Command

This command executes a program which is the heart of the system: it produces an ordered display of statements from the file-reference section of the catalog.

Each entry for a selected keyword is scanned, and the serial numbers which it contains are noted.

Each of these serial numbers is the name of a statement in the file-reference section: each of these statements is assigned a "score" equal to the weight of the keyword, and this score is accumulated with further references from other keywords.

When all of the selected keywords have been used to score the file references, the file-reference statements with nonzero scores are displayed in order of decreasing score.

4. Example of Catalog File

a. Keyword S ection

(nls) on-line system * (u1) (u2) (u3) (u4)

(ug) user guides * (u1) (u2) (u4)

(kse) keyset * (u1)

(cdp) control-dialog proc. * (u1)

(anz) content analyzer * (u2)

(fij) file jumping * (u3)

(inf) info. retrieval * (u4)

(vs) view control * (u1) (u3)

b. File-Reference Section

(u1) (nlist,1:xnhj) nls,ug,vs,kse,cdp; nls user guide

(u2) (conan,1:x2bhj) anz,ug,nls; content analyzer user guide

(u3) (rlink,1:x2bhj) fij,vs,nls; link jumping and returns

(u4) (infor,1:x2bhj) inf,ug,nls; information retrieval system

5. Example of Reordering

Suppose that the system is used on the catalog shown above. The user has considerable interest in file jumps, so he gives the keyword "fij" a weight of 5. He is also interested in control-dialog procedures, so he gives the keyword "cdp" a weight of 3. Finally, he is also interested in user guides, so he gives "ug" a weight of 1.

When the command Keyword Execute is given, the following scoring is done:

The keyword "fij", with a weight of 5, applies to serial number u3; therefore the statement whose name is "u3" is given a score of 5.

The keyword "cdp", with a weight of 3, applies to serial number u1; therefore the statement whose name is "u1" is given a score of 3.

The keyword "ug", with a weight of 1, applies to serial numbers u1, u2, and u4; therefore the statements whose names are "u1", "u2", and "u4" are given scores of 1 each. In the case of "u1", this is added to the previous score of 3.

The final scores are 4 for "u1", 1 for "u2", 5 for "u3", and 1 for "u4". They are then displayed as follows:

```
(u3) (rlink,1:x2bhj) fij,vs,nls; link jumping and returns
(u1) (nlist,1:xnhj) nls,ug,vs,kse,cdp; nls user guide
(u2) (conan,1:x2bhj) anz,ug,nls; content analyzer user guide
(u4) (infor,1:x2bhj) inf,ug,nls; information retrieval system
```

The user may then access the referenced files by using the Jump to Link command with the links given in the references.

LINKS AND RETURNS, 4/4/69

The material in this document may appear at first glance to be unrelated, but in fact it is a body of information necessary for proper understanding of the "Jump to Link" command and related commands.

LINK SYNTAX

A link is a special string of text, embedded anywhere in a file, which contains information on some sort of cross-reference elsewhere in the file or reference to another file. This special string is designed to be intelligible to a human reader and also machine-readable and machine-executable.

In the case of a reference to another file, the file must exist as a disc scratch file ("colon file").

The basic syntax of a link is

(username, filename, statementspec :VIEWSPECs)

Capitalization in the link text is irrelevant, and blanks are ignored. The components of the link have the following meanings.

The "username" is the name under which the disc scratch copy is held in storage. This element may be omitted; in this case, the system will assume that the file referenced is stored under the same name as that given by the current file's author when he entered NLS (i.e. the "file owner" name stored as part of the file by NLS).

The "filename" is the name of the file referred to. Although this name must refer to a "colon file" on the disc, the colon is not included here as part of the filename. If the filename is omitted, the system assumes that the link refers to a cross-reference somewhere in the current file.

Note that the filename must always be immediately followed by a comma.

The "statementspec" is either a statement name or a statement number, indicating the exact location within the file. If this is omitted, the system will assume that the origin statement (first statement) of the file is intended.

The "VIEWSPECs" are a string of characters controlling the display parameters to be used in displaying the referenced material. The codes are exactly the same as would be used for

entering VIEWSPECs manually. If the VIEWSPECs are omitted, the current display parameters are used.

Note that the VIEWSPECs must always be preceded by a colon.

Examples: Suppose that in a file belonging to user Smith and named PLAN, we find the following links.

(Jones,specifications,performance:ebg)

This refers to a file named "Specifications" and stored under the username Jones. Specifically, it refers to a statement named "performance" in that file. The VIEWSPECs "ebg" are to be applied in viewing this reference.

(specifications,performance:ebg)

This differs from the previous link only in the omission of a username. Therefore it refers to a different file, also named "Specifications" but stored under Smith's name. The reference is again to a statement named "performance" and the same VIEWSPECs apply.

(costs,2b)

This refers to Statement 2b of Smith's file named "Costs". No VIEWSPECs are given, so whatever parameters are in effect when the link is executed will remain in effect.

(staffing:ebg)

Here no filename is given, so the link refers to a statement named "staffing" in the current file (Smith's "Plan"). Again the VIEWSPECs "ebg" are to be applied.

RETURN JUMPS

General

The commands "Jump to Return" and "Jump to File Return" permit the user to return automatically from any jump to a previous view. Thus links may be freely used without the danger of losing one's place.

Proper understanding of the interfile returns requires an

understanding of the WORKING COPY file; this is discussed in the last section of this document.

The Intrafile Return Ring

All jumps made within a file (except jumps made with "Jump to Return" and "Jump to Ahead") are recorded in an ordered list called the Intrafile Return Ring. The ring may have up to five entries, each of which records a display start position and a set of display parameters -- i.e. the information needed for complete reconstruction of a view, assuming that no editing takes place.

The list is a ring in the sense that its ends are joined; i.e. the first entry is also the list successor of the last entry. A pointer indicates the "current" entry, i.e. the entry containing information for the current view. Each new jump (except "Return" and "Ahead") causes a new entry to be made ahead of the current entry, and the pointer is moved to the new entry.

The command "Jump to Return" causes the pointer to be moved back one entry, and the display is recreated from the new "current" entry. No changes are made in the entries themselves.

The command "Jump to Ahead" causes the pointer to be moved forward one entry, and the display is recreated from the new "current" entry. No changes are made in the entries themselves.

It will be seen that because of the ring structure of the list, repeated use of "Jump to Return" or "Jump to Ahead" will eventually bring the user back to the starting point.

It should also be remebered that each new entry in the ring always goes just ahead of the "current" entry, and that an old entry may be overwritten in the process.

T he Interfile Return Stack

Each change to a new file that is made with "Jump to Link," "Jump to File Link," or "Load File" is recorded on an ordered list called the "interfile return stack." This list is similar to the intrafile return ring, but differs in two important respects:

The length of the list is not fixed, nor is it known to the user. The number of entries is a function of the total

amount of information in the entries. Typically, the stack will have room for three or four entries, and this is sufficient for many purposes.

The list is not a ring; the first entry has no list predecessor and the last has no list successor.

The list is handled as a pushdown stack, with some added complexity because of the pointer. As in the ring, each new entry always goes ahead of the current pointer position and the pointer is moved. If the current position is not the last entry, then a new entry will overwrite an old one; if the current position is the last entry and there is room for another entry, then the creation of a new entry does not destroy any old entries; but if the current position is the last entry and the list is full, then the stack is pushed down to make room for the new entry and the first entry on the stack is lost.

It will be seen that this is essentially equivalent to what happens on the ring when it is filled, with the important difference that the user cannot know when the stack is full because its length is not fixed.

The "Jump to File Return" and " Jump to File Ahead" Commands

These commands are exactly analogous to the corresponding intrafile jump commands. "Jump to File Return" moves the pointer back one entry and creates a new display from the information in the new "current" entry, and "Jump to File Ahead" does the reverse. If "Return" is attempted from the first entry in the list or "Ahead" from the last entry, the message "ILLEGAL ENTITY" is displayed and the command is aborted.

These commands make no changes in the stack entries.

The "Jump to File Working Copy" and "Jump to File Current" Commands

NOTE: The WORKING COPY file is discussed in the next section.

Creation of a WORKING COPY file does not create an entry in the stack; therefore the WORKING COPY file cannot be accessed by the "Return" or "Ahead" commands. Instead, the command "Jump to File Working Copy" is provided. This command causes the WORKING COPY file to be displayed but does not change the stack or move the pointer.

Thus there is a situation where the pointer does not indicate the file currently displayed.

When the WORKING COPY file is being displayed, the "Jump to File Return" and "Jump to File Ahead" commands retain their normal meanings in the strict sense: they cause the pointer to be moved one entry backward or forward, and the file indicated by the new pointer position to be displayed.

However, it makes a difference whether the WORKING COPY has just been created by the system or whether it has been previously created, jumped away from, and then jumped back to.

In the first case, the two commands will do what is "intuitively" expected. "Jump to File Return" returns to the file previously displayed, and "Jump to File Ahead" does the reverse.

In the second case, however, "Jump to File Return" will not cause a return to the file previously viewed but to the one before that.

Thus these commands lose their normal "intuitive" meanings.

The confusion arises for two reasons:

(1) The user forgets that he is looking at the WORKING COPY

(2) The pointer in the stack does not point to what is currently being displayed. This is "counterintuitive".

The first of these is overcome by familiarity with the system; the second is overcome by use of the command "Jump to File Current". This command simply causes display of the file indicated by the pointer, i.e. the file from which the WORKING COPY was jumped to.

THE WORKING COPY FILE

An important distinction is made between "ordinary" and "special" files.

Ordinary files have names assigned by the user when he creates

them, and are accessed by these names.

Special files (namely the CHECKPOINT and WORKING COPY files) have names assigned to them by the system when they are created, and are normally accessed by the user via special commands ("Load Checkpoint" and "Jump to File Working Copy). They may also be accessed by their names, which are codes depending on the console number; in this case they behave like normal files.

The WORKING COPY file is used by the system to avoid modifying a file that is being worked on by the user until the user gives the "Output File" command. The procedure followed by the system is as follows.

When the user enters NLS, the WORKING COPY file is automatically created, opened, and displayed. At this point, the WORKING COPY file consists only of an origin statement containing the word "DUMMY".

If the user proceeds to create new material with the "Insert Statement" command or other Insert commands, the new material is added to the WORKING COPY file.

If the user at any time gives a "Load File" command, the WORKING COPY file is closed and the specified ordinary file is opened and displayed.

If the user now begins to modify the displayed material, the system keeps a record of the modifications in its own scratch space and alters the display without making any change in the file itself. Eventually, however, it runs out of scratch space (almost immediately in the case of a large file).

When this happens, the system copies the displayed file, as modified, to the WORKING COPY file. It then closes the original file, opens the WORKING COPY file, and displays it.

This process appears to the user as follows: the display is momentarily blanked, the message "WORKING COPY CREATED" appears, and then the display is restored. (As we have seen, the WORKING COPY file already exists, so this message is slightly misleading. What has happened is a file-transfer process rather than a file-creation process.)

If this process does not occur, the original contents of the

WORKING COPY file are preserved.

In the case just described, the contents would be just the "dummy" origin statement. However, in other cases the working copy might contain a complete modified copy of some file.

In either case, the user might now load or jump to another ordinary file. The file that has been displayed (special or ordinary) is closed, and the newly specified file is opened and displayed. The procedure repeats.

It is important to remember that whenever the message "WORKING COPY CREATED" appears, any previous contents of the WORKING COPY file are lost. Therefore, if the user desires to be sure that the material will be saved he must output it to an ordinary file -- either the original file that the WORKING COPY file was created from, or a new one.

The above describes the situation when an ordinary file is modified and copied, in modified form, to the WORKING COPY file. Different rules govern the use of the CHECKPOINT file.

The CHECKPOINT file is normally accessed only with the command "Load Checkpoint." When this command is executed, the CHECKPOINT file is immediately copied to the WORKING COPY file, which is then opened and displayed. No message is displayed, but any previous contents of the WORKING COPY file are lost.

This does not apply if the CHECKPOINT file is accessed with a "Load File" command using the file's system-generated name.

Finally, there are three brief rules on the "creation" of the WORKING COPY file:

(1) Any "Output" command will always force "creation" of the WORKING COPY file. This takes place before the final CA of the "Output File" command, so the user can always force "creation" by typing o f CD. This has no other effect.

(2) The exclusive use of "Jump" commands will never force creation of the WORKING COPY file.

Note that to use this rule one must use no other commands from the point where the file is first loaded or jumped to. Use of any other command may cause the WORKING COPY to be

"created" on a subsequent "Jump" command.

(3) Use of an "Output" command to write out the displayed WORKING COPY file does not cause the WORKING COPY to be closed and the output file to be displayed; this is only accomplished by a "Load" or "Jump to File" command.

Discussion: This command has an extensive package of subcommands for changing various parameters applying to the display of various entities on the screen. (The syntax description is given further on.)

These entities are the bug (its armed cursor, its active cursor, and the mark it leaves), the command feedback line, the date/time register, the echo register, the name register, the VIEWSPEC feedback area, the working text area, and the tab stops.

The first level of subcommands includes specification of these entities plus the subcommands "save" and "restore."

"Save" is used to store all of the current parameters in a buffer with a number from 1 to 8, and "restore" takes one of these buffers and puts the parameters stored in it into effect.

The second level of subcommands includes the following:

"Character," which applies only to the bug entities and is used to specify the character to be used for plotting each of them.

"Abbreviated," which applies only to the command feedback line and causes it to show only the first letter of each word that it would normally contain.

A special set applying only to the text area:

"Columns" (number of characters in a full line of text)

"Indenting" (number of spaces to indent for each level)

"Lines" (number of lines in text area -- currently limited to 20)

"Rows" (same as "lines")

"Vertical increment" (spacing between lines)

"Parameter set," which applies to all entities except tab stops. This subsubcommand has a number of subsubsubcommands.

Syntax: The syntax is given below in hierarchical form, with the input letters at the beginning of each statement. "Parameter Set" is given in a separate branch following the hierarchy.

Execute Viewchange: e v (b/c/d/e/n/r/s/t/v)

Bug: b (d/a/m)

Disarmed Cursor: d (c/p)

Character: c (any character) CA

Parameter Set: p (see below)

Armed Cursor: a (c/p)

Character: c (any character) CA

Parameter Set: p (see below)

Mark: m (c/p)

Character: c (any character) CA

Parameter Set: p (see below)

Command Feedback Line: c (p/a)

Parameter Set: p (see below)

Abbreviated: a CA

Semantics: Only the first letter of each word in the command feedback is displayed.

Date/Time Register: d (p)

Parameter Set: p (see below)

Echo Register: e (p)

Parameter Set: p (see below)

Name Register: n (p)

Parameter Set: p (see below)

Restore: r (digit, 1-8) CA

Semantics: Viewchange conditions previously saved under the digit typed are placed in effect.

Save: s (digit, 1-8) CA

Semantics: The current Viewchange conditions are saved under the digit typed and may be placed in effect later with "Restore."

Tabs: (tab character) 1$(number CA) CA

Semantics: Each number typed (followed by a CA) sets the position of the next tab stop on the display, relative to the previous one (the first one is relative to the margin). An additional CA terminates the command.

Text Area: t (c/i/l/p/r/v)

Columns: c (number) CA

Semantics: In effect, this sets the number of character positions from the left-hand margin to the right-hand margin. The left-hand margin is fixed.

Indenting: i (number) CA

Semantics: The number typed sets the number of spaces to indent for each level in the statement structure.

Lines: l (number) CA

Semantics: This sets the number of lines available for the text area. It is limited to 20.

Parameter Set: p (see below)

Rows: r (number) CA

Semantics: This sets the number of lines available for the text area. It is limited to 20.

Vertical Increment: v (number) CA

Semantics: This sets the distance between lines in the text area.

VIEWSPEC Area: v (p)

Parameter Set: p (see below)

Syntax Under "Parameter Set"

Parameter Set: p (b/d/f/h/i/p/s)

Boldface On/Off: b $(NOT CA) CA

Semantics: Any character except CA changes "on" to "off" or vice versa.

Display On/Off: d $(NOT CA) CA

Semantics: Any character except CA changes "on" to "off" or vice versa.

Flicker On/Off: f $(NOT CA) CA

Semantics: Any character except CA changes "on" to "off" or vice versa.

Horizontal I ncrement: h (number) CA

Semantics: This sets the horizontal distance between characters.

Italics On/Off: i $(NOT CA) CA

Semantics: Any character except CA changes "on" to "off" or vice versa.

Position: p (bug selection of point on screen) CA

Semantics: This sets the position of the first character of the entity, unless the entity is "text area." In this case, the horizontal component is ignored and the vertical position of the first line is set.

Size: s (digit 0-3) CA

Semantics: This sets the character size for the entity: 0 is smallest, 1 is normal, and 3 is largest.

THE NLS VECTOR PACKAGE

The vector package allows the user to create simple line drawings, with labels, as a part of his file. See the command descriptions for how to enter the vector package in association with a particular statement.

Drawings may be output via the printer or via film; on output to other devices they disappear.

The following commands are valid within the vector package:

Insert Commands

(iv) Insert Vector

Syntax: i v 2$(bug selection of point) b/CD

Semantics: The bug selections determine the endpoints of vectors.

Each selection after the first produces a new vector.

Thus four selections produce three vectors, with vector 1 meeting vector 2 at the position of the second selection, and vector 2 meeting vector 3 at the third selection.

To "lift your pencil" and break the continuity of the lines type a "b" or a CD.

(il) Insert Label

Syntax: i l SPACE LIT CA (bug selection of point)

Semantics: After typing the label, hit a CA to attach the label to the bug. The bug selection fixes the label in its current position on the screen (rounded off to the nearest position that can be output on the printer).

Move Commands

(mv) Move Vector

Syntax: m v (bug selection of vector) $(left mouse button) (bug selection of point)

Semantics: When the vector is selected, its ends are marked O and X. The end marked X will move to the point selected and the end marked O will remain fixed.

Hitting the left-hand button on the mouse will cause the O and X to be interchanged.

The bug is then moved to the desired point and a CA hit to select the point. The "X" end of the vector will move to this point.

(ml) Move Label

Syntax: m l [1] CA (bug selection of point)

Semantics: When the first CA is hit, the label [1] is attached to the bug and moves with it. The bug selection fixes the label in the new position.

(md) Move Drawing

Syntax: m d 2$2(bug selection of point)

Semantics: The two selected points define a translation vector, and the entire drawing is moved by this vector quantity.

Delete Commands

(dv) Delete Vector

Syntax: d v (bug selection of vector) CA

Semantics: The selected vector is marked with an O and an X; when the CA is hit, it is deleted.

(dl) Delete Label

Syntax: d l [1] CA

Semantics: The selected label is deleted.

(dd) Delete Drawing

Syntax: d d CA

Semantics: The entire drawing is deleted. The command is used for starting over from scratch without leaving the Vecor Package.

(t) Translate Vector

Syntax: t (bug selection of vector) $(left mouse button) (bug selection of point)

Semantics: The syntax of this command is identical to Move

Vector in terms of actions by the user to specify which vector and which end.

The end marked X moves to the specified new position and the end marked O moves in such a way as to preserve the length and direction of the vector.

(vv) Vertical

Syntax: v (bug selection of vector) $(left mouse button) CA

Semantics: When the CA is hit, the end marked X is moved horizontally so that the vector is vertical.

(h) Horizontal

Syntax: h (bug selection of vector) $(left mouse button) CA

Semantics: When the CA is hit, the end marked X is moved vertically so that the vector is horizontal.

(g) Grid

Syntax: g CA

Semantics: The grid is either on or off; after typing "g" to get "grid" in the command feedback line, a CA causes the grid to change state.

The grid provides the user a means to draw "pretty pictures."

A rectangular array of dots is displayed in the drawing area for use as a visual guide.

All positions are rounded off to points that can be output on the printer.

Spacing

(sf) S pacing Off

Syntax: s f CA

Semantics: This will set a flag that goes along with the picture telling the display creation routines not to space the statements to leave room for this picture. A picture with spacing off will be displayed, but it will appear superimposed on any following text.

(sn) Spacing On

Syntax: s n CA

Semantics: This is the complementary command to spacing off. Since the flag is set for spacing on as the default option, this command is necessary only to change the flag back.

(a) Abort

Syntax: a CA

Semantics: Everything that has been done in the current instance of the vector package is thrown away, the command "Vector Package" is aborted, and it is as if the command had not been given.

(f) Finished

Syntax: f CA

Semantics: This command returns control from the vector package to NLS proper, storing the picture as part of the file.

ODDS AND ENDS

The Centerdot

The centerdot character may be used in any input of a statement, branch, plex, or group to end a statement and start a new statement. LEVADJ is permitted for the new statement. The sequence is as follows:

LIT CENTERDOT LEVADJ LIT ...

In other words, hitting a centerdot during input of a statement is exactly the same as hitting CA and then making a bug selection of the resulting new statement.

Use of Pointers

Pointers make it possible to select entities that are not on the display. The entity must have a pointer fixed on it with the "Pointer Fix" command for this to be done. To select the entity, hold down the right-hand button on the mouse while entering the name of the pointer from the keyboard or keyset, and then release the button. This is exactly equivalent to making a direct bug selection of the character that has the pointer on it.

The Branch-Only Feature

When the branch-only feature is turned on with the VIEWSPEC g, only one branch at a time is displayed. In some cases, this gives a less confusing view of the text. It also affects the amount of material output under the Output Device command; if branch-only is on, output ceases as soon as the branch defined by the statement at the top of the screen has been output.

The Tree-Display Feature

By using the VIEWSPEC capital G, it is possible to see the file as a tree structure instead of text. The tree structure shows the relationships of statements in the file.

All structural commands may be used with the tree display by pointing to nodes in the tree as one would point to a character in a statement to select it.

To return to the text display, use the VIEWSPEC capital H.

The Trail Feature

The trail feature is used to set up statements in such a way that only a particular set of statements will be displayed, and in a particular order.

To understand this feature, it is necessary to know roughly what the content analyzer is about; see the document on the content analyzer.

The trail feature works as follows: some string of characters is chosen as a "trail marker" for a particular set of statements. A content-analyzer pattern is then set up to recognize occurrences of this string.

The trail marker is used to mark "turning points" from the normal sequence of statements. Each time the marker appears in a statement, it is followed by a statement name in parentheses. This combination of marker and name is used by the display-creation routines as a signpost to the next statement to be displayed. Between signposts, statements are displayed in normal sequence -- subject to any other VIEWSPECs that may be in effect.

Note that normally the content analyzer is not used in its normal mode when using the trail feature. If this were done, only the "turning points" in the trail would be displayed.

LEVADJ (Level Adjust)

LEVADJ means that the user is given an opportunity to control the structural level assigned to a statement, branch, plex, or group when it is is inserted, moved, copied, or created by the Break Statement command.

In the case of the entities branch, plex, and group, the LEVADJ applies explicitly to the first statement in the entity; other statements in the entity retain their structural relationships to the first statement.

The LEVADJ process works as follows:

Any command which involves LEVADJ begins with a bug selection of some existing statement which the new, moved, or copied material will follow. Let us call this the "where" selection or "where" statement.

Some commands involve more than one bug selection. In these cases, as long as more selections are needed, the system displays the statement number of the "where" statement.

As soon as all necessary bug selections have been made, the system displays a proposed new statement number. This number is always the logical successor to that of the "where" statement, at the same level. Thus if the "where" statement is 5a3b2, the proposed new numer will be 5a3b3.

The user may accept this number by hitting CA (in the case of Move and Copy commands) or a space followed by literal type-in (in the case of Insert commands).

If the user wants a higher-level number, he hits a "u" (for "up"); if he wants a lower-level number he hits a "d" (for "down").

The displayed number is changed accordingly. This process may be repeated as often as desired; however, only numbers that make sense will be produced.

Thus if the displayed number is a first-level number, a "u" will have no effect; if the displayed number is already one level lower that that of the "where" statement, a "d" will have no effect.

Whenever the displayed number is the one that the user wants, he accepts it by hitting CA or a space (depending on the command).

It must be understood that the "operand" (the new, moved, or copied material) will not always go immediately after the "where" statement.

The operand material will end up at the next "logical" location implied by the statement number that the user accepts.

If the new number is one level lower than that of the "where" statement, then the first statement of the operand material becomes the first substatement of the "where" statement. Any substatements that already exist are renumbered accordingly.

This is the only case where the operand always goes immediately after the "where" statement.

If the new number is at the same level as that of the "where" statement, then the operand goes after the last statement of the branch defined by the "where" statement.

In this case, the operand immediately follows the "where" statement only if the "where" statement has no substatements.

If the new number is a level higher than that of the "where" statement, the operand immediately follows the last statement in the branch defined by the source statement of the "where" statement, and so forth for higher and higher levels.

These rules can be summed up rigorously as follows, for what it is worth:

The operand always follows the "where" statement, but not always immediately.

It always follows immediately if it is a level lower than the "where" statement.

If the operand is at the same level as the "where" statement or higher, it will immediately follow the last statement in the branch defined by the last previous statement at the same level as the new number.

Bug Selection

There are three kinds of bug selection, all basically the same as far as the user is concerned. All selections are made by pointing with the mouse/bug and hitting CA once.

(1) Selection of a text entity within a statement: here the user points to any character within the text entity (character, word, visible, etc.) that he wishes to select.

When the CA is hit, an "o" is superimposed on the character that was hit.

Selection of the entity "Text" requires two character selections.

(2) Selection of a structural entity (statement, branch, plex, or group): here the user points to any character within a statement that defines the entity.

When the CA is hit, an "o" is superimposed on the character that was hit.

Selection of a group requires two statement selections.

(3) Selection in the Vector Package: This breaks down into three subtypes:

(3a) Selection of a label: this is the same as selection of an entity within a statement -- the user points to any character within the label.

When the CA is hit, an "o" is superimposed on the character that was hit.

(3b) Selection of a point on the screen: here the user simply points to a geometric location in the plane represented by the display.

When the CA is hit, the system immediately takes action; "o" is not plotted.

(3c) Selection of a vector: here the user points to a geometric location near the vector.

When the CA is hit, the system responds by displaying a "o" at one end of the vector and an "x" at the other.

When bug selections are being made in the course of using a command, and the command has not yet been executed, a CD will cause all bug selections to be cancelled.

In certain commands it is undesirable to do this for one of two reasons -- either because the command involves more than one selection and the user only wants to cancel the last one, or because a CD will cancel the whole command.

In some of these commands it is possible to use "backspace" key (or the left-hand mouse button) as a "soft CD". The effect is simply to cancel the last bug selection that has been made. The superimposed "o" remains on the display but is meaningless.

The "soft CD" works in the following commands:

All Jump commands

All Move commands

All Set commands

All commands operating on the entity "text".

Execute Content Analyzer

DEFINITIONS

branch A specified statement, plus all of its substructure -- i.e. all of its substatements, plus all of their substatements, etc.

bug The mark on the screen which is moved by the mouse and which is used for selecting (pointing to) entities on the display.

When the bug is "active," i.e. when a selection can be made, it appears as an up-arrow; when it is inactive it appears as a plus sign.

character Any letter, digit, punctuation mark, space, tab, or carriage return; an indivisible entity

chord A combination of keys on the keyset (see keyset).

For a table showing the meanings of the chords, see document on keyset and VIEWSPECs.

end The last statement in any branch; specified by specifying the branch.

file A complete tree structure of statements with a single root (the origin statement).

filename The name of a file. It appears as the first word in the origin statement of an existing file, and must be supplied by the user in creating a new file.

gap character Any space, tab, or carriage return

GCHAR Abbreviation for gap character.

group A subset of a plex, consisting of all branches from one specified branch to another, inclusive.

head The first statement in a sublist.

The head is specified by pointing to any statement in the sublist.

invisible Any consecutive string of gap characters, bounded by (but not including) printing characters or the end of a statement: see printing character, gap character, statement

Specified by pointing to any character in the string. If a single printing character lying between two invisibles is pointed to, both invisibles (and the printing character) are selected.

keyset The device at the left-hand side of the console. When a combination of keys (a chord) is depressed on the keyset, the effect is the same as striking a key on the keyboard.

For an explanation of the keyset, see document on keyset and VIEWSPECs.

keyword A content indicator for reference retrieval (see document on keyword information-retrieval system).

label A string of text placed in a picture by means of a command in the vector package. See document on vector package.

LEVADJ The specification of level when a statement, branch, plex, or group is newly created or moved. See document ODDS AND ENDS.

level The "rank" of a statement (see statement) in the hierarchy of the file (see file).

The level is equal to the number of fields of letters or digits in the statement number; thus Statement 3 is a first-level statement, Statement 4a10g3 is a fifth-level statement, etc. Level is of great importance in understanding the hierarchical structure of an NLS file.

mouse The device at the right-hand side of the keyboard. When it is rolled around on the tabletop, it causes the bug to move correspondingly. For an explanation of the three buttons on top of the mouse, see "Introductory Notes" document.

name If the first word of a statement is enclosed in parentheses, it is the name of the statement.

The command Jump to Name can then be used to place the statement at the top of the display. This is done by entering the name from the keyboard or keyset, or by finding an occurrence of the name as text on the display and pointing to it with the bug.

origin The first statement in a file; it contains information about the file, plus any other text the user inserts. It has a level of 0, and hence no statement number.

pattern A string of special-language text in a statement which may be compiled via the command Execute Content Analyzer. When compiled, it produces a program that is used by the content-analyzer feature (see document on content analyzer) or by the trail feature (see "trails" entry in document ODDS AND ENDS).

PCHAR Abbreviation for printing character.

plex Another name for a substructure, used in command specifications.

A plex is specified by pointing to any one of its highest-level statements.

pointer A string of up to three characters which is attached to some character in the text with the Pointer Fix command.

predecessor The statement preceding a specified statement in a sublist.

printing character Any letter, digit, or punctuation mark.

source The statement of which a specified statement is a substatement.

statement The basic structural unit of a file of text in NLS. Formally, it is a string of text and/or pictures which is bounded at the beginning by the end of the previous statement or the beginning of the file, and bounded at the end by the beginning of another statement or the end of the file.

Statements are arranged in a tree structure or hierarchy and are assigned "statement numbers" which indicate their positions in the structure. Each statement has a number, made up of alternating fields of digits and letters; the number of fields indicates the "level" of the statement (see level).

A statement is specified by pointing to any character in the string.

sublist The set of all substatements of a specified statement (not including the substatements of the substatements).

substatement A statement "X" is called a substatement of another statement "Y" if it is deeper in the structure than "Y," if it follows "Y," and if there is no intervening higher-order statement. "Y" is called the source of "X." The statement number of "X" will be the same as that of "Y" except that it will have one more field at the end. The value of this field gives its ordinal position in a "sublist" of the substatements of "Y."

A substatement is specified by pointing to the source statement.

substructure The set of all substatements of a specified statement, plus all their substatements, etc. until no more are found. The set of all branches defined by statements in the sublist of a given statement.

successor The statement following a specified statement in a sublist.

tail The last statement in a sublist.

The tail is specified by pointing to any statement in the sublist.

text Any string of characters within a statement, bounded by (and including) two specified characters: see character, statement

trail A set of statements in a file, which can be displayed sequentially by using the trail feature. For a discussion of this, see "trails" entry in document ODDS AND ENDS.

vector A line in a picture. See document on vector package.

visible Any consecutive string of printing characters, bounded by (but not including) gap characters or the end of a statement: see printing character, gap character, statement

Specified by pointing to any character in the string. If a single gap character between two visibles is pointed to, then both visibles (and the gap character) are specified.

word Any consecutive string of letters and/or digits, bounded by (but not including) any other types of characters or the end of a statement: see statement

Specified by pointing to any character in the string. If a single character is pointed to which is not a letter or digit and lies between two words, then both words (and the single character) are specified.

HOW TO USE THE PRINTER FACILITY
Preliminary Document

11/28/69

I INTRODUCTION TO PRINT

PRINT is a subsystem which operates from any work station. It is used for outputting files through the line printer.

PRINT uses a queue to keep track of the files that users want printed. Each time a user designates a file, PRINT places it at the end of the queue; each time it finishes printing a file, it removes that file from the queue and takes the next file, which is now at the beginning of the queue.

Thus the user simply tells PRINT the name of the file, and then leaves PRINT and continues with other work rather than waiting for the file to be printed. He may even log out, as long as he sets the printer file permanent. Various commands allow the user to interrogate PRINT to see where his file is in the queue.

II ESSENTIAL PROCEDURE FOR ROUTINE PRINTING

Use the NLS/TODAS command Output Device Printer File to create a preformatted printer file.

Return to the Exec and call the PRINT subsystem (type PRI then a period). The PRINT subsystem types a plus sign in the margin, showing that it is ready for a command.

The following shows what should appear on your wo station as you operate PRINT. The succeeding statements explain the various commands.

```
@PRINT.
+PREFORMATTED FILE :FILENAME. 3 25
+FINISHED.
@
```

Type a P to get the command PREFORMATTED FILE. This command expects a filename, so give the name of the printer file you have just created, then terminate with a period.

The system responds by giving two numbers.

The first is your file's position in the queue.

The second is a rough measure of the length of the queue -- the number of hundreds of computer words in all the files in the queue up to and including yours.

After typing these numbers, PRINT again types a plus sign.

To return to the executive, type an F to get the command FINISHED. Terminate with a period.

You may now CONTINUE NLS.

If you log out of the system before your file is printed, be sure to set the printer file permanent.

III SUMMARY OF COMMANDS IN THE PRINT SUBSYSTEM

Delete File

Syntax: d <filename> .

Semantics: The named file is removed from the queue. Note that the file itself is not actually deleted.

Finished

Syntax: f .

Semantics: PRINT is terminated and user control returns to the Executive.

Length of Queue

Syntax: l

Semantics: PRINT responds with two numbers.

The first is the number of files on the queue, and the second is the number of hundreds of computer words in all the files in the queue.

Preformatted File

Syntax: p $(<filename> ,) <filename> .

Semantics: Note that this command allows the user to terminate filenames with a comma or a period. If a comma is used, another filename may be added. The effect of the command is to place the names on the queue to be printed.

This command is used to print files created from NLS or TODAS with the Output Device Printer command. PRINT places the named files in the queue, behind the current last item.

PRINT also responds with two numbers for each file. The first is the number of files on the queue, and the second is the number of hundreds of computer words in all the files in the queue, up to and including the named file.

QED File

Syntax: q $(<filename> ,) <filename> .

Semantics: Note that this command allows the user to terminate filenames with a comma or a period. If a comma is used, another filename may be added. The effect of the command is to place the names on the queue to be printed.

This command is used to print files created from NLS or TODAS with the Output Device Printer command. PRINT places the named files in the queue, behind the current last item.

PRINT also responds with two numbers for each file. The first is the number of files on the queue, and the second is the number of hundreds of computer words in all the files in the queue, up to and including the named file.

Status for File

Syntax: s <filename>

Semantics: PRINT responds with two numbers. The first is the number of files on the queue, and the second is the number of hundreds of computer words in all the files in the queue, up to and including the named file.

/PS4UG, 01/ 13/69 1559:46 FKT ;

1 (ps4ug) Introduction

1A PASS 4 User Guide

1B R.E. Hay, F.K. Tomlin

2 (General) General

2A "USER GUIDE, SECTION I: INTRODUCTION"

2A1 (Abstract) PASS4 is the name of the program which takes text in NLS structure and produces formatted output for a variety of external "devices":

2A1A Input is performed "statement" at a time.

2A1B Text is formatted "line" at a time for output according to established conventions, however, much of this process is under control of "directives" which enable the User to change various parameters that govern the output process.

2A1C Text is translated and sent to a User-designated output file in the character codes and with allowances for the idiosyncrasies of an external device (such as Flexowriter, Dura Mach 10, line printer, Teletype, film, etc.).

2A1D The resulting output file may then be transferred to the desired output meduim (such as papertape, hardcopy, magnetic tape, film, etc.).

2B (Labels) The following are file label links (except for directive labels/names, see (DirectiveLabels:pd3bg)) (General:pxb) (Abstract:pwg) (Purpose:px3brg) (FormatAlgorithm:px2brg) (CharacterFormatting:px3b4rg) (LineFormatting:px3bg) (PageFormatting:px3bg) (Directives:px2bg) (Syntax:pebg) (CurrentDirectives:px3bg) (DirectiveLabels:pd3bg) (Notes:pt2rg) (Sample:pwg) (Operation:ptrg)

2C History

2C1 160A

2C2 3100 OFF Line System (3 pass)

2C3 3100 OFF Line System (4 pass)

2C4 PASS4

2C5 940 Output Program

2D (Purpose) Purpose

2D1 The output program is designed to take text in the on line file format and produce formatted hard copy.

2D2 Input

2D2A Input is from text files in the on line system format, ANSI character set. Input is done on a statement basis; that is, a statement is located, and then all of that statement is sent to the output. The order in which statements are chosen is usually in the sequential form in which they occur in the file, but can be chosen by other criteria (see the on line "chaining").

2D3 Formatting

2D3A Text is formatted for output according to the conventions, procedures, etc., that have been established, but much of the output process is under the control of text "directives" which enable the user to change various parameters and flags which govern the output process, and thus obtain different kinds of formatting.

2D4 Output

2D4A As text is gathered from the input, formatted and sent to some output file, it is being translated into the character codes and device idiosyncrasies of some external device (such as a flexowriter, dura mach 10, line printer, teletype, film, etc.)

2E Extensions

2E1 Page formatting

2E2 Pictures and graphics

2E3 Table of contents

2E4 Foot notes

2F Limitations

2F1 Line formatting

2F2 Right justification

2F3 Hyphenation

2F4 Type setting

2F5 Two to one map

2F6 One to two (or none) map.

2G Needs for hard copy

2G1 On hire, all files are accessable

2G2 Network

2G3 Security

2G4 Cost

2H Program logic

2H1 Input statement

2H2 Output statement

2H2A Output statement, character at a time

2H2A1 Determine likely number of lines to statement and posisibley paginate

2H2A2 Delete statement number, fill with spaces

2H2A3 Scan for tabs and carriage returns, setting indentation and right justification

2H2A4 Send out proper number of carriage returns

2H2B Look for directives and execute them

2H2B1 Scan output stream for directives see (directives), executing, and possibly printing, any directives

2H2C Format output

2H2C1 Format characters

2H2C1A Tabs

2H2C1B Underbars

2H2C1C Overbars

2H2C1D Case shifts

2H2C2 Format lines

2H2C2A Indentation

2H2C2B Delete leading spaces

2H2C2C Back scan

2H2C2D Delete trailing spaces

2H2C2E Right justify

2H2C2F Center

2H2C3 Format page

2H2C3A Header

2H2C3B Page number

2H2C3C Number of lines per page, etc.

2H2D Output character to output file.

3 (FormatAlgorithm) Format Algorithm

3A "USER GUIDE, SECTION II: FORMATING ALGORITHMS"

3B To take a file of text as input (with little format structure) and produce a new formatted file as output involves a number of considerations, algorithms, conventions, processes, and procedures. Much of this has evolved over the course of time, and others are a consequence of data structure, conventions, etc.

3C The current implementation uses a sequential input to output process. One statement at a time is brought in, then that statement output, a line at a time. No backup is ever done on a scale larger than a line. Thus all pagination is done in-line, that is, a page is not formatted, then checked, scaned, made to see if all is well, etc., but is done on the basis of overflowing the current page and being forced onto the next page.

3C1 The basic cycle for outputting a document in this manner is to:

3C1A Input a statement from the input file (when no more statements remain, the process ends).

3C1B Output a statement to the output file. This involves:

3C1B1 Making statement level decisions, such as deleting statement numbers, number of lines between statements, looking for tabs and setting right justification parameter, etc.

3C1B2 Sending all characters through the directive analysis program so that any directives that occur in the document can be recognized and performed.

3C1B3 Sending characters to the formatting program, which does the character, line, and page formatting.

3C1B4 And finally sending the converted, formatted characters to the output file.

3D The heart of the output formatting is the program FMT (ForMat Text program). Almost all parameters used by this program are under the control of the directives, and thus the actual output processing becomes available to the user. This part of the output program does not know about statements. It only knows about pages, lines, and characters. The routines which call FMT know that there are statements, and send the extra carriage returns and statement numbers to FMT as text.

3E The formatting considerations, processes, etc. can be grossly broken down into the problems of formatting characters, lines, pages, and statements.

3E1 (CharacterFormatting) Character Formatting

3E1A Tabs: The problem of tabs is two-fold: the manner in which the device tabs, and the tab stops that the user would like to have. The flex will go to the next tabstop, even if you are one character in front of a tab position. The on-line printer does not have such a thing as tabs. Then, regardless of the tab stops on a device, a user sometimes would like to be able to specify where he would like to have tabs, and let the program produce a sequence of tabs and spaces to obtain that tabbed position.

3E1A1 When tabing causes a line to overflow, the line will be broken and continued on the next line.

3E1A2 The concept of "right justified tabs", where the tab causes the word preceeding the tab to go to the tab stop, provides a means of aligning columns of numbers.

3E1B Underbars: The underbar is a little tricky. Depending on the device, an underbar on input can cause a case shift, uniderbar, back space, and another case shift to be sent to the device. In addition, this sequence does not count as a printing character.

3E1C Case: In general, there are four situations (provided that we even have to consider the problem of upper and lower caise); there are characters which can only be printed in upper caise, those which can only be printed in lower case, those which are in both case (as a space), and the alphabetic characters. The overbar can be used to indicate that if the following character is an alphabetic character, then it should be capitalized.

3E2 (LineFormatting) Line Formatting

3E2A Leading and trailing spaces

3E2B Indentation

3E2C Back scan

3E2D Right justification

3E2E Centering

3E3 (PageFormatting) Page Formatting

3E3A Page number

3E3B End of page

3E-3C Header

4 (Directives) Directives

4A "USER GUIDE, SECTION III: DIRECTIVES"

4B "Directive" is a term applied to text embeded in a document which "directs" the output processing and formatting being performed on the document.

4B1 Directives provide a means of changing various parameters, code conversion tables, etc. that are being used in the actual output formatting process. Since these directives are embeded as text in a document, they can be edited and manipulated as text, and yet be recognized on output as having special meaning.

4B2 When directives are recognized in the output process, the parameters are changed, or the action is performed at the point in the output process at which the directive occurs. Thus, directives need to be placed at the point in the text that the particular parameter change (or action) is desired. On output it is possible to supress the printing of directives so that the output will be affected, but the output will not include (as text) the actual directive(s).

4B3 There are six different types of directives. These different types as well as the directives themselves are recognized by a syntax scan of the output done by a META II recognition type of process. The syntax equations used to describe directives are:

4B3A (Syntax) Syntax:

```
; 4B3A1  directive = $("." direct/.empty) ;

  4B3A2  direct = .dir (.t0 "=" '" <anbitarty string> '"
  /.t1 "[" .num "]" "=" exp /.t2 "=" exp /.t3 /.t4 "=" '"
  <arbitrary string> '" /.t5) sp $cond sp ";" ;

  4B3A3  cond = "IF " sp exp ;

; 4B3A4  exp = union ;

  4B3A5  union = inter sp $("AND " sp union) ;

  4B3A6  inter = neg sp $("OR " sp inter) ;

; 4B3A7  neg = "NOT " sp relat / relat ;

  4B3A8  relat = sum sp $(".LT" ssum /".LE" ssum /".EQ" ssum
  /".NE" ssum /".GE" ssum /".GT" ssum) ;

  4B3A9  ssum = sp sum ;

; 4B3A10  sum = prod $("+" prod /"-" prod) ;

  4B3A11  prod = prim $("*" prim /"/" prim /"%" prim) ;
```

4B3A12 prim = var / .num /"(" exp ")" ;

4B3A13 var = "." .dir (.t1 "[" .num "]" /.t2) ;

4B3A14 sp = $" " ;

4B3B In the syntax equations, there are a few constructs that are specific to this META II description of the directives.

4B3B1 ".dir" is used to indicate the attempt to recognize a directive name in the output. A directive name is three letters long, and must occur in the table of given directives.

4B3B2 The construct ".tn" (where n=0,1,2,3,4,5) represents a test that is made to see if the last directive recognized by a ".dir" is of "type n". The six possible types are:

4B3B2A Type 0: This represents a new directive which is being defined by the user. It is possible to give a name that is not already defined, and assign a string to that name which will be inserted in the output each time that this name is later used as a directive. see type 3 below.

4B3B2B Type 1: Some directives represent the names of various arrays that are used in the output process, and provide the user with a means of setting the values in the array, or using an entry in an array in an expression. This type of directive always requires a subscript.

4B3B2C Type 2: This type represents those directives which stand for some parameter or value in the output program. Most directives are of this type.

4B3B2D Type 3: This type of directive is one which has already been defined by the user in type 0. Whenever the name of a directive which the user has defined is given, the string which was assigned to that name is inserted in the output.

4B3B2E Type 4: This is the directive which is used to specify the running header ("hed"). This is a special case and the only one of this type.

4B3B2F Type 5: This describes those directives which require special consideration or action to be taken (such as restoring the page).

4B3C It should be noted that the evaluation of the expressions and boolean operators are done on a left to right analysis, and that the only precedence occurs between boolean expression, relation, and expression.

4B3D Normally, most of this analysis and capabilities will not be needed when giving a directive, and it is instructive to

note that the following things need to be present to constitue a directive:

4B3D1 a period

4B3D2 a directive name

4B3D3 an equal sign (if setting a value)

4B3D4 an optional conditional part

4B3D5 and a terminating ";"

4B3E The "cond" construct provides a means of testing various directives etc. and depending on the outcome of this test, to perform the indicated directive or not. Note that it is possible to string a series of conditional tests together, and the directive is performed only if they all are true.

4B3F Example: .NSW=2 IF .PGN .GE 2 AND NOT .ROM=0; is a valid directive. It has actually been recognized and executed for this document, however as "rom" is zero there has been no effective action. If "rom" were non-zero ther page numbering would be done as if "nsw" were two (2).

4C (CurrentDirectives) Current Directives

4C0 (DirectiveInformation) Directive Information

4C1A There are many directives that are currently implemented, not all of which are of interest to the normal user.

4C1A1 For each directive the initial (or default) values appear immediately following the three character identifier, in order: dura, TTY, NLS-QED, flex, printer, film -- NOTE: 0=no/off, 1=yes/on.

4C1A2 An asterisk has been placed as the last character of the statements describing directives of general User interest. If the User is viewing this document on-line then the following pattern can be executed by the content analyzer in order to display only these statements: *PI SE(PI) < $NP '*;

4C2 (DirectiveCategories) Following are directive category labels: CharacterSet SpecialCharacters LineFormatting PageFormatting StatementFormatting GlobalDirectives DirectiveDirectives ActionDirectives

4C3 (DirectiveLabels) Following are directive labels/names: fcr fsp ftb fub fov fds fbs fsu fsd fsc fig cod tsw tal tsp tst usw dub upr usp osw dov pov sov csw cmd cas mch nch ind nin min ins dls dts fln rtj rsw msp cen icr psw pln plo mln nln nbl ntp nsw pgn pgp fnc rom ndh ssw hsw hln hed dsn lsp scr wln pst tcr dpv igs skp typ dev dpr igd tma tmb tmc tmd dnm dty dvl dmx hlt

res rel crl nul

4C4 (CharacterSet) Directives associated with the character set:

4C4A Input character codes:

4C4A1 (fcr) : all 155b -- input code for a carriage return (i.e., the search code used by the statement input algorithm when looking for a carriage return)

4C4A2 (fsp) : all 0b -- input code for a space (i.e., the search code used by the statement input algorithm when looking for a space)

4C4A3 (ftb) : all 151b -- input code for a tab (i.e., the search code used by the statement input algorithm when looking for a tab)

4C4A4 (fub) : all 134b -- input code for an underbar (i.e., the search code used by the statement input algorithm when looking for an underbar)

4C4A5 (fov) : all 133b -- input code for an overbar (i.e., the search code used by the statement input algorithm when looking for an overbar)

4C4A6 (fds) : all 15b -- input code for a dash (i.e., the search code used by the statement input algorithm when looking for a dash)

4C4A7 (fbs) : 141b,0b,0b,141b,0b,141b -- input code for a back space (i.e., the search code used by the statement input algorithm when looking for a back space). This is sometimes used in the underbar analysis to insert a backspace into the output.

4C4B Output character codes:

4C4B1 (fsu) : 10b,0b,0b,172b,377b,53b -- output code for shift to upper case

4C4B2 (fsd) : 20b,0b,0b,174b,377b,54b -- output code for shift to lower case

4C4B3 (fsc) : 0b,0b,0b,13b,0b,0b -- output code for stop code

4C4B4 (fig) : 201b,0b,0b,201b,0b,0b -- output code for ignore (used to delete next character; see directive (igs:pwg); only has meaning for dura and flex)

4C4B5 (cod) : all "CODE" -- name of code conversion array. By means of this type 1 directive and the directive giving the character case (see directive (cas:pwg)), it is possible to change the output code for any character in the input. For example: to change the code for the letter "a"

to an asterisk, "*", one would produce the following directive: .cod[65]=10;, where the "a" is in the 65th (binary 101st) element of the array and the code for an asterisk is 10 (binary 12). All code conversion arrays are located in the ARRAY file of the PASS4 program (source code).

4C5 (SpecialCharacters) Directives associated with special character considerations:

4C5A Tab character

4C5A1 (tsw) : all 1 -- indication that tabs are to be searched for in order to execute appropriate directives; see directives (tal:pwg) and (tsp:pwg)

4C5A2 (tal) : 2,2,2,1,2,2 -- tab algorithm to be used for the output of tabular information (1=flex type, 2=dura type, 3=one space)

4C5A3 (tsp) : 1,1,0,0,1,1 -- space fill tab, i.e., insert necessary space characters in the output in order to produce proper tab spacing

4C5A4 (tst) : all "TABSTP" -- is an array directive which is used to determine where the tab stop settings are. A non zero entry indicates a tab stop, so setting a "tst" position to 1 will denote a tab stop, and setting a position to 0 will effectively clear a tab stop. This array is initialized for tab stops every eighth column. There are six words to this array (starting with word zero). The bit positions which are on (from left to right within the word) denote the tab stops. Therefore, to change one tab stop one must determine the decimal equivalent of an octal number necessary to change all tab stops within one word of this array.

4C5B Underbars

4C5B1 (usw) : all 1 -- indication that underbars are to be searched for in order to execute appropriate directives; see directives (dub:pwg), (upr:pwg), (usp:pwg), and usp:pwg)

4C5B2 (dub) : 0,1,1,0,1,0 -- delete underbars (if searching for them)

4C5B3 (upr) : 1,0,0,1,0,1 -- treat underbar as printing and spacing character

4C5B4 (usp) : 1,0,0,1,0,1 -- indication that underbar causes output device to space

4C5C Overbars

4C5C1 (osw) : all zero -- indication that overbars are to be searched for in order to execute appropriate directives;

see directives (dov:pwg), (pov:pwg), and (sov:pwg)

4C5C2 (dov) : all zero -- delete overbars from output (if searching for them)

4C5C3 (pov) : all zero -- treat overbar as printing and spacing character

4C5C4 (sov) : all zero -- indication that overbar causes output device to space

4C5D Case shifts

4C5D1 (csw) : 1,0,0,1,1,1 -- indication that case shift analysis is to be performed for output

4C5D2 (cmd) : all 2 -- case mode for alphabetic characters (1=forced upper case, 2=normal, 3=forced lower case) For example: cmd=3 will force all alphabetic characters to lower case until "cmd" is reset*

4C5D3 (cas) : all "KASE" -- directive which gives the case for each character (an array, i.e. type 1, directive). If a code is changed for a character (via "cod" directive), its case should also be set to the proper case in an analogous manner (see (cod:pwg))

4C5D3A 1=upper case, 2=either case, 3=lower case, 4=alpha upper case, 5=alpha lower case, 6=film lower case, 7=film upper case, 8=film greek case, 9=film null case

4C6 (LineFormatting) Directives associated with line formatting:

4C6A Line parameters

4C6A1 (mch) : all 72 -- gives maximum number of symbolic characters to an output line (line length). For output to film the default value is automatically adjusted to raster units*

4C6A2 (nch) : all 1 -- gives current number of characters on the output line*

4C6A3 (ind) : 1,1,0,1,1,1 -- indentation option. If ind=1 then indentation will be performed*

4C6A4 (nin) : all zero -- number to indent for each statement level (set at the start of each statement to be output to: level-of-the-statement, minus one; times "ins")*

4C6A5 (min) : all 48 -- maximum number of spaces to indent. For example, if min=10 then statements would be indented as given by the indentation algorithm up to a maximum of 10*

4C6A6 (ins) : all 3 -- amount to indent for each statement

level (used in calculation of "nin")*

4C 6B Line formatting

4C6B1 (dls) : all 1 -- delete leading spaces on a line. The only time leading spaces would occur would be when there are spaces following a carriage return in the statement which is being input. If one is using leading spaces to produce tabular output, then set dls=0*

4C6B2 (dts) : 0,0,0,0,0,1 -- delete trailing spaces (any extra spaces that are left in a line are usually put at the end - beyond the right justification)

4C6B3 (fln) : all 1 -- format output lines according to the algorithm with directive values as given (either output line as soon as it is full, or take the pains to backscan to last word, right justify, etc.)

4C6B4 (rtj) : 1,1,0,1,1,1 -- right justification option. If rtj=1 then text will be right justifed if possible*

4C6B5 (rsw) : all 1 -- right justification indicator for current line of input. This is the parameter that is actually used during the formatting process. It is set by the program during the outputting of a statement (according to the "rtj" directive) depending upon such things as tabs in a line, the last line of the statement, etc.

4C6B6 (msp) : all 15 -- maximum number of spaces to put into line to obtain the right justification. If more than "msp" spaces are required, no attempt will be made to right justify, and the line will be left as is*

4C6B7 (cen) : all zero -- center "line"(s) on the page (only if right justification off, rtj=0) starting with the line containing cen=1 to, but not including, the line containing cen=0. The User must be careful exactly where this directive is placed. "cen" affects the line which is presently being formatted for output, and normally the User does not know a priori which string of text comprises a line*

4C6B8 (icr) : all zero -- put ignore codes in front of generated carriage returns on output (meaningful only for cura and flex)

4C7 (Pageformatting) Directives associated with page formatting:

4C7A Page parameters

4C7A1 (psw) : 1,1,0,1,1,1 -- pagination option. If psw=1 then page numbering, etc. will be performed on output*

4C7A2 (pln) : 66,66,66,66,63,40 -- number of lines to a page (including header and page number)*

4C7A3 (plo) : all zero -- skip to top of page whenever the statement level is level one (except for statement number 1)*

4C7A4 (mln) : 56,56,56,56,56,38 -- number of printing lines to a page (does not include page number, but does include header -- must be less than value of pln-pgp-ntp)*

4C7A5 (nln) : 65,65,65,65,62,39 -- current line number on the page being output*

4C7A6 (nbl) : all 1 -- n-spacing (i.e. nbl=2 for double spacing)*

4C7A7 (ntp) : 3,3,3,3,0,0 -- number of lines down from top of page to begin printing*

4C7B Page numbering

4C7B1 (nsw) : all 1 -- page numbering option (nsw=1 will provide for page numbering at the bottom center of the page, nsw=2 will provide for odd numbers to appear right-justified and even numbers left-justified at the bottom of the page -- for both roman and arabic numerals)*

4C7B2 (pgn) : all zero -- current number of the page being output*

4C7B3 (pgp) : 5,5,5,5,5,1 -- number of lines up from bottom of the page for the page number*

4C7B4 (fnc) : 3,0,0,3,3,3 -- indicates the case for the roman page number (1=upper case, 2=no change, 3=lower case, 4=upper case, 5=lower case)*

4C7B5 (rom) : all zero -- Roman numeral page number option (roman numerals good only up to 50)*

4C7C End of page

4C7C1 (ndh) : 0,5,0,0,0,0 -- number of dashes (code given by :"fds") at end of page (usually used only with teletype output)

4C7C2 (ssw) : all zero -- indicates a stop code to be inserted (see directive (fsc:pwg)) at the end of each page (for mats - normally only for flex)*

4C7D Header

4C7D1 (hsw) : 1,1,0,1,1,1 -- header option. If hsw=0 then no header will be output at the top of each page*

4C7D2 (hln) : all 1 -- number of blank lines to follow the header*

4C7D3 (hed) : all zero -- directive used to set the header. For example: .hed="HEADING"; will set the program to output "HEADING" at the top of each page (if "hsw" is set on)*

4C8 (StatementFormatting) Directives associated with statement formatting:

4C8A Statement numbering

4C8A1 (dsn) : all zero -- indication that statement numbers are to be deleted if set on*

4C8A2 (lsp) : all 5 -- number of leading spaces to replace the statement number (if dsn=1)*

4C8B Statement formatting

4C8B1 (scr) : all 2 -- number of carriage returns to seperate statements upon output (scr=0 will close statement gaps and merge statements together)*

4C8B2 (wln) : all 2 -- number of lines of a statement guaranteed to be output on the next page if the statement overflows the current page (widow lines)

4C8B3 (pst) : all zero -- paginate whenever statement will not fit on current page*

4C8B4 (tcr) : all zero -- replace all carriage returns in the statement by spaces during output (normally for input from QED using the PASS4 Subsystem)

4C8B5 (dpv) : 1,1,1,1,0,0 -- don't produce vector or vector label output (at present only useful for printer and film. If idevice is dura and dpv=o then space will be left for the vector and vector label information; no vector will be printed!)*

4C9 (GlobalDirectives) Global directives:

4C9A General

4C9A1 (skp) : all zero -- skip over text from skp=1 to skp=0, doing directive recognition, but no outputting and no formatting*

4C9A2 (typ) : all 1 -- do not output text lines from the line which contains typ=0 to the line which contains typ=1, but continue doing directive recognition and formatting. This directive has inadequacies similar to "cen"; it only recognizes the directive after a "line" has been formatted and is ready for output, and it is difficult for the User to determine, a priori, exactly what portion of his output comprises a line*

4C9A3 (igs) : 1,0,0,1,0,0 -- insert ignore codes (see

directive (fig:pwg)) in document before each character added to the output which was not in the input (page number, header, right justification, etc.)

4C9A4 (dev) : 7,1,3,2,6,5 -- gives the device number for which the current document is being formatted. (dura=7, teletype=1, NLS-QED=3, flexowriter=6, printer=6, film=5, controlling teletype (QED format)=4)

4C10 (DirectiveDirectives) Directives about directives:

4C10A General

4C10A1 (dpr) : all 1 -- directive print (dpr=0 will supress output of directives)*

4C10A2 (igd) : all zero -- ignore directives from igd=1 to igd=0, but continue to recognize directives for possible supression of output (see directive (dpr:pwg))*

4C10A3 (dnm) : all "DNAME" -- array directive (type 1) that gives the name of the array in the PASS4 program which contains the directive names

4C10A4 (dty) : all "DTYPE" -- array directive (type 1) that gives the name of the array in the PASS4 program which contains the directive types

4C10A5 (dv1) : all "DVALUE" -- array directive (type 1) that gives the name of the array in the PASS4 program which contains the directive values

4C10A6 (dmx) : all "DMAX" -- array directive (type 1) that gives the name of the array in the PASS4 program which contains the directive maximum values

4C10B Temporary directives

4C10B1 (tma) : all zero -- temporary a (not used by program - for use by user)

4C10B2 (tmb) : all zero -- temporary b (same as for "TMA" above)

4C10B3 (tmc) : all zero -- temporary c (same as for "TMA" above)

4C10B4 (tmd) : all zero -- temporary d (same as for "TMA" above)

4C11 (ActionDirectives) Directives causing some immediate action to be taken (note: these directives do not have a parameter for changing a directive value; e.g. .NUL;):

4C11A General

; 4C1.1A1 (hlt) : all zero -- (currently does nothing)

; 4C1.1A2 (res) : all zero -- causes a page restore at the point the directive occurs*

; 4C1.1A3 (rel) : all 1 -- causes a page restore at the end of the current line of output*

; 4C1.1A4 (crl) : all zero -- inserts a carriage return in the output at the point the directive occurs*

; 4C1.1A5 (tab) : all zero -- inserts a tab character in the output at the point the directive occurs*

; 4C1.1A6 (nul) : all zero -- null directive, i.e., it does nothing

4D (Notes) : Helpful notes

4D1 :Input from QED file format using the PASS4 Subsystem.

4D.1A Upper/lower case problem: the character codes used by QED are the codes for the AHI upper case alpha characters. To get lower case characters for a device like the dura, it is possible to force the alpha characters to come out lower case with "cmd=3;".

4D.1B Now to get those characers that you want upper cased, it is necessary to insert an overbar in front of each letter to be capitalized. To do this a character, such as the back slash, needs to be chosen, and the directives "fov=74b; osw=1;" included in the document before any back slash is used for capitalizing (the code for a back slash is 74b).

4D.1C There is a convention that any carriage return in the document will be honored, i.e., for each carriage return in the document, a carriage return will occur on output. This works well for on-line files, but for QED the carriage returns which are in a document usually are not there for formatting because of QED's line length. With the directive "tcr=1" these carriage returns in the body of a statement will be converted to spaces, thus giving the impression of one long line of free field text. If explicit carriage returns are still wanted, the directive "crl" can be used.

4D.1D Underbars. To obtain underbars, much the same process is necessary as with overbars. First, a character to represent the underbar needs to be chosen, such as the exclamation mark (1b), and the appropriate directives set. These are: "usw=1" (look for underbars), "fub=1" (code for underbar is 1b), "cod[1]=100b" (dura underbar code is 100b), and "cas[1]=1" (underbar is upper case character on dura).

4D.2 :Changes from standard settings:

4D.2A :"hln=3" puts three blank lines beneath header instead of

one.

4D.2B ;"pgn=pgn-1" works well on the first page (header page, page 0) and keeps the count straight if many files are output in sequence.

5 (Sample) Sample

5A Directives:

5A1 ;.HLN=3; .MLN=54;

5A2 ;.HED="Sample Page"; .RES;

: Sample Page

5B This page is an attempt to illustrate some of the parameters that can be used to control the output formatting process in NLS.

5B1 Lines are indented by the amount "nin" which is set to (statement level minus one)*"ins".

5B2 ;.HSW=0; .RES;

6 (Operation) Operations

6A "USER GUIDE, SECTION IV: OPERATING SYSTEM, PASS4-SUBSYSTEM".

6B The User interface

6B1 To provide a means of communicating with the output program, the EPASS4 procedure was written. This procedure responds to characters in the "command recognition" mode used by the executive time sharing system, and makes checks to see that all is set up correctly before initiating the output process.

6B2 The characters recognized are:

6B2A (I) Input: The letter "I" is typed to designate the input file. A legal input file name should be given, then acknowledged with a period.

6B2B (O) Output: The output file is specified by typing the letter "O", then giving the output file name, which is terminated with a period.

6B2C (D) Device: Besides an input and an output file, a device has to be indicated, by typing the letter "D". The program inputs from the input file and outputs to the output file, translating and formatting for some device. The devices built into the program are:

6B2C1 Dura Mach 10 (type "D"), with the characters <>I]←↑ but without the characters exclamation, degree, and cent. These are available on an IBM selectric ball, and a few simple directives will change the tables accordingly. Also there is an Iverson ball available.

6B2C2 Teletype (type "T"), which is used to punch teletype tapes.

6B2C3 Controlling Teletype (type "C"), which is the same as the teletype device, except that leader is not generated. This device can be used for QED output.

6B2C4 Flexowriter (type "X"), which has had keys modified and now has the characters <>[]←↑ but has forfeited the characters pound, cent, on1, and on2. Also note that some of the characters have been moved to make this character change possible.

6B2C5 Printer (type "P"), which prepares an output file to be used as input to the system "PASS4 KLUDGE PRINT" routine for output on the line printer (upper/lower case capability).

6B2C6 Film (type "F"), which prepares an output file to be dumped on a magnetic tape file. The magnetic tape may then

taken to the CDC-3200 computer and used as input to a routine which will have as output 35mm film and, if desired, Xerox hard copy.

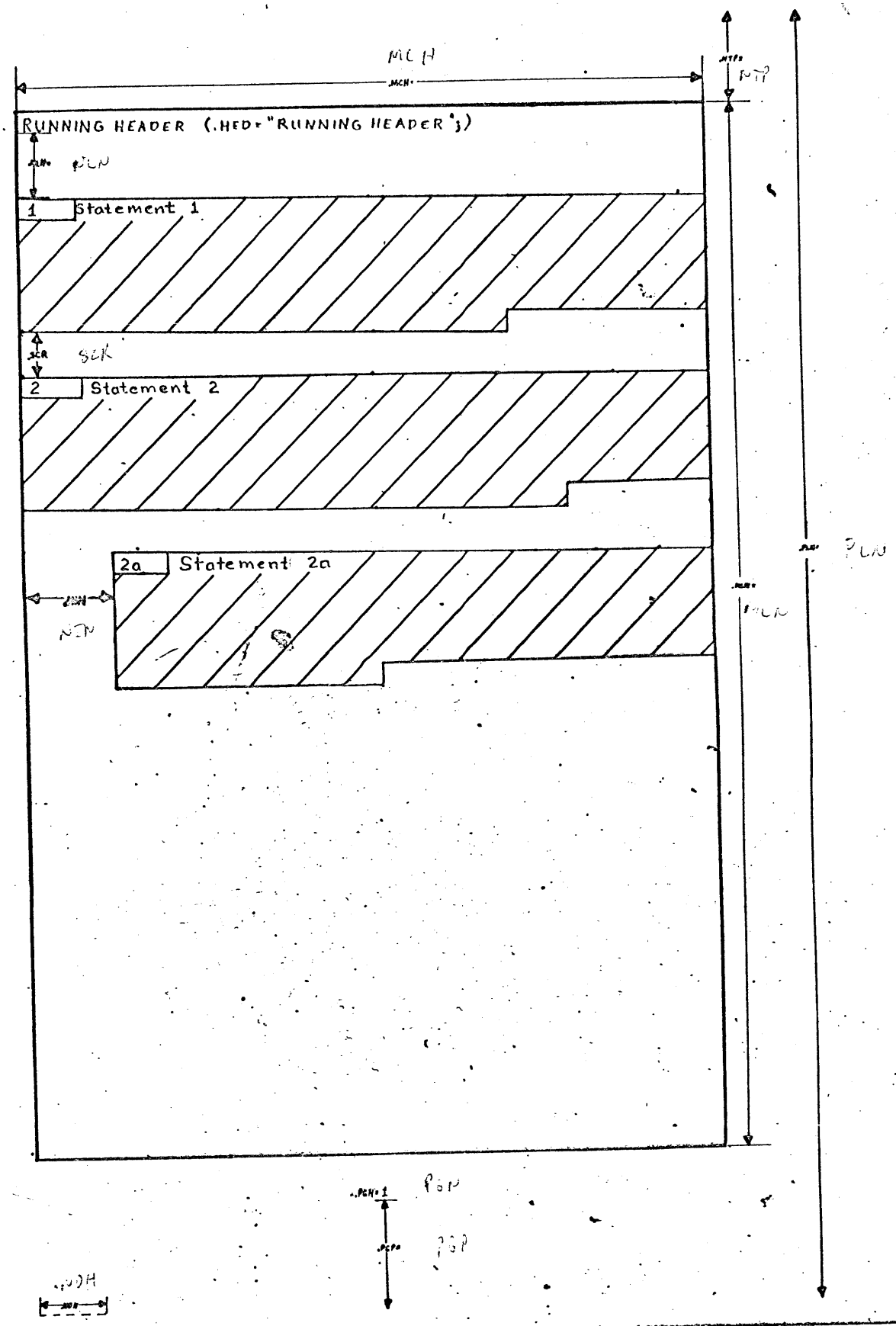
6B2D (B) Begin execution: Typing the letter "B" is used to signal the desire on the part of the User to begin the output process. A check is made to see if all the necessary information has been supplied, the input and output files are opened, and a check is made to see if the output and input file are the same (could be fatal). Then the PASS4 program is called and the output process begins.

6B2E (F) Finished: When the output process is complete (except for closing the output file), control returns to the PASS4 Executive procedure. To terminate the output program, and return control to the TSS executive, the letter "F" is typed. This causes the output file to be closed, and the output program to be terminated.

6B2F (.) It is possible to enter directives from the EPASS4 procedure. These will be parsed and executed by the directive routines, but will not be printed in the output. Thus any last minute directives can be entered before beginning the output process. However, it is important to note that the device should be indicated before directives are typed, as the device command sets the directive tables, which will cancel the effect of any previous directives that have been typed. Also, it is meaningless (and perhaps even fatal) to type a directive of "type 5", ie, a directive which causes some action. This is a problem since any action taken is meaningless as the output process has not been started. Used with common sense, the ability to enter directives before starting the output process can be very useful.

6B2G Any other character than those given above is illegal, and will evoke a response of <space> <question mark>.

6B2H All commands must be ended with an acknowledging period. If any other character is typed, the command will be aborted.


MCH
MCN
MTP
RUNNING HEADER (.HED "RUNNING HEADER")
PLN
1
Statement 1
SCR
SCR
2
Statement 2
2a
Statement 2a
NCN
MCN
PLN
PGP
PGP
PDH

/PS4NS, 01/15/69 0855;1 FKT ;

1 PASS4 has been updated and changes of user significance are noted below:

1A New directives in PASS4:

1A1 (IGD) "igd" is a new directive which is designed to IGnore Directives between igd=1 and igd=0.

1A2 (DPV) "dpv" is a new directive which is designed to not produce vectors when outputing a file through PASS4 (Don't Produce Vectors).

1A2A This directive only has meaning for three output devices: film, printer, and dura.

1A2B Using "dpv" for device "dura" will only result in not producing space for vectors as graphics are not now produced for this device.

1A2C This directive only has meaning when outputing via NLS and not when using the PASS4 Subsystem.

1A3 (TAB) "tab" is a new immediate action directive which places a tab (or the equivalent) in the output stream at the point it is inserted.

1B The PASS4 Subsystem has had certain commands changed:

1B1 "X-flex" has been substituted for "Flex"

1B2 "Film" has been substituted for "3200-film"

1B3 "Begin execution" has been substituted for "Go do it"

1B4 "Finished" has been substituted for "Zap"

2 Be sure to review changes described in the last issue of PASS4.

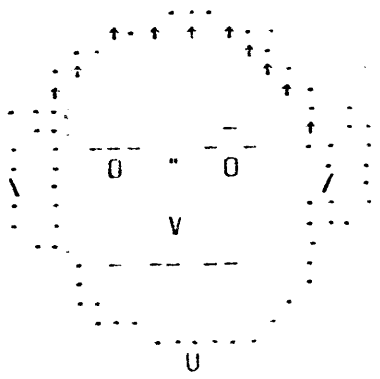
'/PASS4NEWS', 11/22/68 0929:18 FKT ;

I Changes in PASS4 output capability:

IA New output capability for vectors and their labels:

IA1 By indicating "F" (for film) one may eventually produce 35mm film output including both text and vectors (and vector labels). This process is described in (TOMLIN,GODUG,GODUC:pxbng) - GUDAS User's Guide.

IA2 Output to "Printer" will now produce a reasonable facimile of vectors and associated labels. Example:



IA3 Output to "Dura" will leave blank lines where vectors (and labels) would occur.

IA4 For any other output device than those described above vectors and their labels are ignored.

IB The default value for the directive "min" has been changed from 15 to 48. "min" is the maximum number of spaces for which indenting will be done.

IC There should no longer be any spaces added between the statement number and the first character of the statement when right justification is done.

/USER1, 09/18/68 1630:27 DGC ;

1 (XDOC) Documentation for XDOC system

1A Davic Evans, September 3, 1968

1B (g)lossary: synopsis files intro receipt checkout procedure update checkout annotation example directives Casseres

2 (Synopisis): The XDOC collection is the AHI library of "external" documents, such as reports, articles, working papers, books, etc., generated by sources outside AHIRC. Descriptions of material in the XDOC collection are now stored as NLS files on the disk. The standard NLS commands should be used to review and study these descriptions. The opportunity now exists for users to add their comments and notes as substatements under each XDOC description. This will be useful information for users of documents previously read and reviewed by others.

2A A detailed description of procedures (etc.) follows.

3 (intro) A simple system for searching and studying descriptions of documents stored in the AHI XDOC collection, and physically gaining access to the documents in this collection, is now available.

3A Each source document is described in terms of its XDOC number, author, title of article (etc.), publication, and date of publication.

3A1 The XDOC number is simply the order in which the document was entered in the collection.

3B The XDOC Collection is stored in order of XDOC numbers in filing cabinets located in room J2028.

3B1 The keys for these cabinets are available from Roberta Carillon.

3C (checkout) Once the required XDOC source documents have been found or studied using NLS (see PROCEDURE), the documents may be physically checked out of the XDOC Library.

3D (receipt) The library receipt takes the form of an "OUT" card with the user's name entered on it, deposited in place of the document removed from the collection.

4 This first XDOC system is very simple; users should adopt the following procedure:

5 (procedure) The user should read a KDF file from those stored under the name FOLLACK (see username), display it as an NLS file (see FILES), and use the Content Analyzer, VIEWSPECs, freezing, and other standard NLS features to view and review descriptions of the documents in the XDOC collection.

5A In the initial XDOC system there are no new commands specifically

for XDOC.

5B The user simply makes the best use he can of the existing NLS commands. THese should be adequate for elementary retrieval and study operations.

6 (files) Each XDOC file contains descriptions for only 100 XDOC documents.

6A XDOCs have been broken into 100-statement files to make it easier to access them without an excessive drum block allocation.

6A1 Approximately 50 drum blocks are required for each XDOC file.

6B The names of POLLACK's KDF files are made up of an X followed by a number (the two leftmost digits of the first XDOC number in the file), followed by UP.

6B1 Hence the file X32UP contains all XDOC references from X3200 to X3299.

6C AT present, only XDOC numbers greater than X2900 have been entered. See (UPDATE) for procedure for adding additional XDOC references.

7 (UPDATE) From time to time Roberta will prepare the source descriptions of new documents received in the XDOC collection, producing a teletype paper tape. Usually each tape will contain between 15 and 30 new XDOC entries.

7A These tapes contain simple QED and TODAS directives (see directives); they may be input to the SDS 940 using the GO TO Program G-TODAS, which is stored by Evans or Levine as a KDF file named G-TOD.

7A1 (directives) At this stage the directives (apart from the standard QED control characters) are simply < for word delete and $ (as the last character in a statement) for statement delete.

7A2 The use of this program is self-explanatory (but expained below).

7A2A Enter the new XDOC paper tape in the reacer.

7A2B Copy the file G-TOD from Evans' or Levine's disc to the RAD.

7A2C Type the command GO TO /G-TODAS. , where /G-TODAS is the name of the RAD equivalent of the disk G-TOD.

7A2D Response to INPUT is 8 (for 8-LEVEL).

7A2E Response to OUTPUT is the user-designated name for the new QED RAD file.

7A2F The paper tape will then be read, as a QED file, to the

designated file.

7B For this simple system, the new XDOC files should contain no more than 100 statements.

7B1 Hence if the additional XDOC entries span a "100" value, the entries less than ..99 should be appended to the latest existing XDOC file, and the remainder used as the start of a new XDOC file.

7C (casseres) The responsibility for maintaining the NLS XDOC files will reside with Dave Casseres.

8 (annotation) All statements in these original XDOC files are highest-level statements. The opportunity thus exists for a user of the XDOC system to record his notes and comments as substatements of the XDOC descriptions. The following procedure is proposed:

8A Anyone wishing to make notes or comments associated with a given XDOC description should insert a branch immediately following the description.

8A1 The first statement in this branch should carry as its name the name of the commentator followed by the number of the comment (e.g. Evans5), and should contain the date the comments are entered, and a very brief (say one- or two-line) outline of the theme of his comments.

8A1A (EXAMPLE):

8A1A1 (Evans3) 9/4/68 Critical review: article not state of the art.

8A1B This would be the header statement for the third comment by Evans on this XDOC description.

8A2 The comments themselves should then follow as lower-level statements of this branch.

8B For each commentary, the commentator should insert a link containing his name and the comments number (e.g. (Evans3)) at the end of the XDOC description.

8B1 (EXAMPLE):

8B1A (X3345) xxxx ,xxxx xxx xxxx ..... xxxx .....
xxxxx. (Evars1) (Engelbart1) (Evans2) (Rulifscn1) (Evans3)

9 (username) In the near future the user name POLLACK will be changed to XDOC.