Provision has been made for multiple-character output, and it is expected to be implemented shortly after the initial Network monitor is operational. 1f2b2a3a1

3. Implementation 1f3

There are two basic tasks for which the Network monitor must be responsible: the provision of the I/O drivers necessary for using the Network, and the development of a protocol for host-host communication. 1f3a

The I/O drivers have such functions as the following: 1f3a1

(1) Initiation of input/output commands to the hardware interface 1f3a1a

(2) Detection of hardware interface errors and execution of proper corrective or evasive actions 1f3a1b

(3) Buffer allocation and manipulation 1f3a1c

(4) Correct formatting of messages so far as the IMPs and the Network are concerned 1f3a1d

(5) Detection of IMP/Network errors and proper error action 1f3a1e

(6) Notification of 940 status to the IMP and Network 1f3a1f

(7) Initialization and recovery after 940 system crashes 1f3a1g

(8) Allocation and maintenance of links over the Network, including the handling of RFNMs 1f3a1h

(9) Maintenance of necessary internal tables pertaining to the Network 1f3a1i

(10) Communication between the Network and ARC 940 work stations. 1f3a1j

This includes the basic system calls required for input/output, the manipulation of Teletype I/O buffers when a remote user is connected to

the 940 as a telephone-line type user, notification of work stations about Network errors, notification of work stations about illegal requests, etc. 1f3a1j1

A protocol has been established which hosts must adhere to in order to communicate effectively. 1f3a2

The monitor must be able to respond to this protocol in order to use the Network. 1f3a2a

Although the protocol is not yet in final form, some of the probable areas of concern will be: 1f3a2b

(1) Opening and closing of primary links 1f3a2b1

(2) Opening and closing of auxilliary (file-transfer) links 1f3a2b2

(3) Message formatting (host-host) 1f3a2b3

(4) Control message decoding and interpretation 1f3a2b4

(5) Communication of status. 1f3a2b5

Since the fundamental Network drivers will be static once they are implemented, they have been integrated into the existing monitor as efficiently as possible. 1f3b

The protocol, however, will probably be subject to change for some time; therefore, it is being implemented in a less integrated but more flexible manner. 1f3c

Among other things, it is being coded in MOL940, which will make it easier to debug and modify than if it were coded in assembly language. 1f3c1

The general implementation approach is to a large extent dictated by the space restrictions in the 940 monitor. 1f3d

We have tried to put as little code as possible in the resident monitor pages, and as much as possible in a separate page which may be relabeled in and out of the monitor's relabeling. 1f3d1

Thus the resident routines in the monitor are mainly

the ones that are necessary for processing interrupts and certain communications (there are cases when the Network code must communicate with another page which runs in the same position). The remainder of the Network code, and buffer space, resides in the separate page. 1f3d2

G. The NLS UTILTY Subsystem 1g

Manipulation of the large number of files which are directly used in connection with compiling, assembling, loading, and debugging NLS is a significant problem. Accordingly, a subsystem called "NLS UTILTY" has been written to help handle these files. 1g1

NLS UTILTY performs the functions described below for the symbolic, binary, and core-image files of NLS and PASS4 (the output processor).

lg2

1. Archiving lg3

All files relating to NLS are permanently stored on the disc under an archiving system. lg3a

In order for the files to be accessed, they must be explicity read from the archives to temporary storage, and any permanent changes to a file must be recorded by writing the updated version of the file from temporary storage to archive storage. lg3b

NLS UTILTY performs these functions for the user, as well as ensuring the integrity of files written into archival storage. lg3c

2. Compilation lg4

Subprograms for NLS are written in three different programming languages. lg4a

The compilation process is different for different languages, and there is in some instances an interaction between one symbolic file and another. lg4b

The concern that an NLS programmer need have with the details of NLS compilation is minimized by NLS UTILTY. lg4c

With NLS UTILTY, any or all of the NLS subprograms may be compiled; the compilation results are reported to the user in a manner which he designates. lg4d

3. Loading lg5

The loading process for NLS is somewhat complex. lg5a

The unloaded NLS system consists of more than 50 binary files, and they must be loaded in a certain order and in a certain relationship to each other. lg5b

As in compilation, NLS UTILTY makes it unnecessary for the NLS programmer to concern himself with the peculiarities of loading. lg5c

The loaded system consists of 7 core-image files. lg5d

While the files are closely related, there is frequently value in loading only one or another of them. lg5e

For this reason, NLS UTILTY allows a variety of loading options, including one which loads the entire system, and one which loads a specific file. lg5f

4. Listing lg6

Because of the size of NLS, the maintenance of up-to-date listings is a tedious job. lg6a

Functions provided in NLS enable the programmer to produce any number of listings of any or all NLS symbolic files by a simple process. lg6b

More details on the individual functions and the operation of NLS UTILTY may be found in Appendix D. lg7

```
':4870', 09/29/70 1735:10 MGC ;  :RPSFT, 07/09/70 1926:19 DGC ; ["Ref"];
EDITING CHANGES DONE .COD[2|B]=||4B; .PGN=76; .DSN=1; .LSP=0; .DLS=1;
.HLN=3; .RTJ=0; .HED="                                                4870 DGC
09JUL70
7101 ROME FINAL REPORT:  Sec. IV
SOFTWARE SYSTEM";   .DPR=0;
```

Appendix B
THE DIALOGUE SUPPORT SYSTEM (DSS) AND THE JOURNAL

1

I Preface 1a

For his dissertation study at Stanford University, Dr. David A. Evans (then an ARC staff member and associated with the Management Systems Research Activity) developed the case for augmentation of planning teams. 1a1

His thesis (Ref. 1), written with NLS, is over five hundred pages in length. In it he presents for the planning community a broad description of ARC's augmentation approach, developments achieved by ARC, and extrapolations relevant to the planning community. 1a2

As a special case study, Dr. Evans integrated the considerations and possibilities for the Dialogue Support System, as developed within the ARC over a number of years and as studied specially by Evans under this contract. 1a3

Selected extracts from his thesis, slightly condensed, are included below as a good source of relevant concept material about the DSS. These may be considered as trial design notes; the final designs for the various parts of the DSS, and their order of development, are yet to be developed. 1a4

II Basic Components of the Dialogue Support System (DSS) 1b

The DSS can be considered to have two basic parts: (1) the Journal, and (2) a set of NLS features especially designed to operate on the Journal. 1b1

A. The Journal 1b2

One of the most dramatic things NLS enables its user to do is operate on and maintain extremely "plastic" and malleable records of his thought and work. 1b2a

This ever-changing plasticity is the root of basic difficulties in extending NLS for dialogue support. When members of a team are contributing to a plan or design, one of the most important things is that the

"targets" of their contributions remain stationary, as if in a diary, or journal. Ironically, the design of a "Journal" to maintain stationary-target records of the transactions of members of a team proved to be innovative in the NLS environment, whereas it would be "normal" if we were dealing with simple pencil and paper. 1b2b

The Journal is a special repository for NLS files which may be "sent to the Journal" and no longer modified, or changed in any way. 1b2c

The design objective of the Journal is to provide the basis for evolution of a diary for a team, sufficiently rich to play the same role as a personal diary plays when used for record keeping, and as the basis for composition, reflection, and extended memory. 1b2d

B. Operations Based on Journal Entries 1b3

The second component of the DSS is a collection of special NLS features, designed to make the Journal useful as the basis for supporting team dialogue. 1b3a

The Journal provides the team members with a chronicle of their contributions to plans and designs. NLS, as extended for use as part of the DSS, is a vehicle that (for example) enables team members to annotate contributions from others, to call for specific action, to make synopses of records relevant to specific issues, and to make contributions to the evolution of plans and designs that are efficiently and appropriately integrated and connected to the entire record of activity. 1b3b

At another level, NLS is a vehicle enabling team members to "browse" in the Journal, to arrive quickly and efficiently at an understanding of the status of plans and designs that are being documented, monitored, or evolved through the medium of the DSS. 1b3c

Interspersed with this and the previous roles, extended NLS features enable team members to retrieve information from the Journal, to modify and update this information, and to return it to the Journal without destroying the original contributions. 1b3d

III Design of Architecture for the Journal 1c

A. Introduction 1c1

The boundary between the Journal proper and the NLS features that support it is not clearly defined, as those features necessary for servicing the Journal also, indirectly, support the special DSS features. However, the discussion can be simplified by means of this division. 1c1a

B. Stationary Targets 1c2

The ideal record system for dialogue support would be some large, central, evolving record that would keep track of the team's activity as team members contributed modifications, new ideas, new designs, specifications, and so on, over time. We have only to consider the problems raised by the basic file-handling operations of the current NLS to appreciate the difficulty of creating such an evolving record of transactions. 1c2a

In any attempt to use files for dialogue purposes, the first problem encountered arises from multiple access to files. When a file is strictly the "property" of its author, dealing with material for which he alone has prime responsibility, the file owner can quite easily keep track of its development. 1c2b

However, when several individuals make active use of a file, it becomes very difficult for the individuals to avoid canceling each other's work or otherwise interfering with each other. They cannot all access the file simultaneously, and so copies are created; soon there are multiple copies, each copy containing changes and additions made independently by various users. It is then impossible, in the general case, to put these copies back together in such a way that all the work done on the separate copies is preserved. 1c2b1

The problem is much like trying to hit a moving target in the dark, and the desired solution is to find some way to make the target stop moving -- hence the phrase "stationary targets." The existing capabilities of NLS and the file-handling facilities used by NLS are not adequate for achieving this. 1c2c

For example, it would be possible with existing capabilities to give all files a read-only status, so that once a file was created it could never be modified. This would overcome many of the problems of multiple access; however, it would also destroy most of the power and usefulness of NLS as a tool for manipulating information. 1c2c1

Likewise, it would be possible to give all files a public read/write status, permitting any member of the team to modify any file at will. It can be seen that this would lead to immediate chaos: a team member working on a file and wishing to make reference to another file would have no assurance that the referenced file still contained the same information as when he looked at it last. 1c2c2

The concept of the Journal is a way to create stationary targets without the crippling effect of a blanket read-only policy or the anarchy of a blanket public read/write policy. Files "entered in the Journal" have, in effect, read-only status, but numerous capabilities are added to compensate for this; moreover, the Journal contains only selected files which are considered to be "ready" to become stationary targets. 1c2d

C. The Journal 1c3

The Journal is a public repository for information of concern to the team of users. A file sent to the Journal becomes a public record. In principle, at least, it cannot in any way be altered, or retracted. 1c3a

The author has "gone on record" with the statement made by the file's content. He may keep a copy of the file entered in the Journal, and make modifications and corrections in that copy, but cannot replace the original file in the Journal by over-writing it with the revised version. Both the original and revised versions may be entered in the Journal. 1c3a1

A basic Journal function is to provide users with mechanisms and aids to recognize that "later versions" in the Journal have been entered, and to provide users with features to enable them to

retrieve and display the multiple versions of a given file. 1c3a2

In keeping with other (non-computerized) Journals, the only ordering imposed on Journal entries is chronological. 1c3a3

In NLS, "Journal" becomes a distinct user name, with a status similar to all other users. 1c3b

However, the Journal adds a second distinct domain of files to the NLS file universe. Journal files have special features. They are all read-only. They possess two parts -- the text/graphics portions written by their author, and blocks of data containing information added to the file after submission to the Journal. 1c3c

The first component is totally frozen: once a file is "sent to the Journal" the "maximum" user representation for that file may not be subsequently altered. 1c3c1

But the second component, data blocks, may be changed through the addition of new data over time. 1c3c2

1. Journal Entries 1c3d

Although we have been discussing "files" in the Journal, we should refer to a module of information in the Journal as an "entry." From the viewpoint of the NLS file system, an entry is synonymous with a file. However, we wish to emphasize the notion of collecting information from many files together into one module, and sending that module to the Journal as an entry. For this reason, we will persist with the terminology "entry" rather than "file" when discussing the Journal from the point of view of a user (contrasted to the viewpoint of the system). 1c3d1

D. Sending an Entry to the Journal 1c4

Because of the existence of two file universes (regular NLS files, and Journal entries) a user is not compelled to submit all of his files to public scrutiny. 1c4a

He may keep his personal collection of files containing his notes, plans, special reminders, etc.,

separate from the collection of files he submits to the Journal. 1c4a1

Within this personal collection he retains the option of controlling read and write access by other users. He may, for instance, have several files that contain private/confidential information that is of no concern to the team as a whole. 1c4a2

However, the decision to submit one of his own files to the Journal is not totally the prerogative of the user himself, unless all his files have private status. 1c4b

Files stored under a given user name, with other than private status, may be entered to the Journal by any other user. This is similar to the procedure of having testimony, or a speech, or other data, read into the (Congressional) Record. 1c4b1

However, in most cases, Journal entries are submitted by the user who has the file (or component files) stored under his name, as part of the standard NLS file universe. 1c4b2

For one user to submit another's file to the Journal, he must first load that file, make a temporary copy, and submit that copy as a Journal entry as if it was one of his own "normal" NLS files. 1c4c

Entering a file to the Journal involves the following operations: 1c4d

(1) A copy of the file being submitted is made. 1c4d1

(2) That copy is again copied, by the system, and (automatically) written as a new file under the user name "Journal." It is given a new name, which is a unique "Journal Entry Number," and set to read-only status. 1c4d2

(3) The user submitting this file is given a "receipt" by the system, indicating that entry to the Journal has been successful. 1c4d3

The result is that a "shapshot" of the user's file has been recorded as a Journal entry. The user has complete control over the VIEWSPECS controlling the view and

amount of the file submitted to the Journal. For instance, if he so chooses, the user may submit only the first level statements in the file. Or he may submit only selected statements in the file -- for instance, only those that satisfy a specific content pattern. He may, of course, choose to employ no special VIEWSPECS, and submit the entire file to the Journal. The VIEWSPECs used at time of entry to the Journal determine the maximum subsequent view for that Journal entry. 1c4e

Subsequent readers of the Journal entry may employ all available VIEWSPECS to help them study the content of the entry, but are constrained to this "maximum" view. This means, for example, if a file is submitted to the Journal with a 1-1 VIEWSPEC (i.e., only top level statements, and only one line of these), subsequent readers can view no more information in that entry, other than the 1-1 view, even if he uses a VIEWSPEC such as ALL-ALL (i.e., all statements, and all lines of each statement). 1c4f

Thus the result of this entry procedure is the creation of a new read-only file, a stationary target, under the user name Journal, with a unique Journal Entry Number as its name. 1c4g

E. Journal Entry Linkage Systems 1c5

Once we have procedures for submitting entries to the Journal, the next major need concerns linking the individual stationary targets -- the Journal entries -- into a fabric of interconnected information. 1c5a

Interfile links may be used to refer to specific locations in a file from any arbitrary location in another file. The difficulty in this interfile linkage system is that there is no way for a user to discover that a particular entity (e.g., a specific statement) in the file he is reading is being referred to by links embedded in other files, or embedded in other statements within the same file. This basic weakness leads to indiscriminate deletion or alteration of files. 1c5b

To solve this problem in the DSS, Journal entries will have "backlinks." This means that when a link is established in a file (for instance, a file not in the Journal), a special marker will be written automatically

by NLS in the appropriate location of the referent file, indicating that a link is pointing at that entity. 1c5c

This marker will give subsequent readers of the referent file a visual signal that the marked entity is the target of a link in another file. A new NLS command, JUMP BACKLINK, will make it possible for the user to jump from the entity in the referent file "back" to the statement containing the link in the source file. 1c5d

There are five cases of file-pair linkages that produce problems: 1c5e

(1) Linkage between two standard NLS files, A and B, from A to B, and file A subsequently becomes a Journal entry. 1c5e1

Problem: The link in A continues to refer to B, and is unaware of the formation of a Journal entry from B. If B is deleted, the link points to a non-existent file. 1c5e1a

Need: Additional bookkeeping to redirect links to the appropriate Journal entry if B is deleted or otherwise modified to make the link inappropriate. 1c5e1b

(2) Linkage between two standard NLS files, A and B, from A to B, and B subsequently becomes a Journal entry. 1c5e2

Problem: The backlink attached to the referent entity in B points back to A, and is unaware of the Journal entry made from A at a later date. If A is deleted after its copy is sent to the Journal, subsequent efforts to JUMP BACKLINK on the backlink marker from A in B will yield a "no such" message. 1c5e2a

Need: Additional bookkeeping to redirect the backlink to the appropriate Journal entry if A is ever deleted or otherwise modified to make the backlink inappropriate. This leads to the concept of indirect linking.

1c5e2b

(3) Linkages between two standard NLS files, A and B, from A to B, and both A and B subsequently become Journal entries. 1c5e3

Combination of problems and needs of Cases 1 and 2. 1c5e3a

(4) Linkage from a Journal entry to a standard NLS file that subsequently becomes a Journal entry. 1c5e4

Problem: Link in the Journal entry is unaware of the existence of the Journal entry made from B. 1c5e4a

Need: Bookkeeping necessary to redirect the link, if requested, to the appropriate Journal entry if so requested by the user. 1c5e4b

(5) Linkage from a standard NLS file to a Journal entry, and the standard NLS file subsequently becomes a Journal entry. 1c5e5

Same as Case 4 except we are concerned with backlinks rather than links. 1c5e5a

F. Other Basic Journal Needs 1c6

In our first-pass discussion of Journal architecture and needs, we should consider two additional general needs, archiving and cataloguing. 1c6a

Archiving is necessary because the current system has limited storage area for files accessible to NLS. The only mass storage devices presently available in the ARC facility are magnetic tapes, and so, at first, the Journal will have a sequential archive. All Journal entries have archival copies. The archival system provides a back-up to the colon copy of a Journal entry in case of disaster, and a large tertiary storage area for those entries not frequently referenced, that do not have to be kept continually in colon file storage on the disk. 1c6b

Major archiving problems arise because of additional data (including backlinks) associated with an entry after it is submitted to the Journal. 1c6b1

Files are allocated a finite number of blocks on a magnetic tape at the time they are written. Data added after the entry is made may be written in this "slop" area until it is filled. But from then on, these data must be stored elsewhere. Only minor problems arise if the additional data can be stored elsewhere on the same tape, with a link from the original entry to a special file, elsewhere on that tape, associated with that entry, containing additional data. 1c6b1a

However, when the tape is filled, these data have to be stored on a separate tape. This causes considerable difficulty when retrieving the entry and its associated data from the archive. There is no simple solution to this problem while magnetic tape is the archival media. These problems will not arise with random-access mass-storage media. 1c6b2

The final basic Journal feature is a catalogue. Obviously, a Journal reader requires a guide to the contents of the Journal, and this is provided by the catalogue. 1c6c

The Journal Catalogue will have three principal parts: 1c6c1

(1) Subject index 1c6c1a

(2) Citation list for Journal entries 1c6c1b

(3) Keyword lists. 1c6c1c

IV Design for Detailed NLS Features to Support DSS 1d

A. Submission of an Entry to the Journal 1d1

1. Entry/Receipt Procedure 1d1a

When a file is submitted to the Journal, the first operations are concerned with creating a new Journal entry, allocating a unique number to that entry, and giving the sender a receipt. This receipt acknowledges the entry has been made sucessfully, and supplies the sender with sufficient information to enable him to locate and retrieve the entry at a

later date. Details of this procedure are illustrated in the following scenario. 1d1a1

a. Scenario: Entry/Receipt Proecedure 1d1a2

(1) Assume the user, X, has assembled a file (X,X1) to be submitted to the Journal. 1d1a2a

(2) He activates the new NLS command "ENTER FILE TO JOURNAL filename," entering the filename X1, as the operand for this command. 1d1a2b

(3) NLS makes a copy of the file (X,X1) as a temporary file, (JOURNAL,T1), i.e., under the user name "Journal." 1d1a2c

(4) Immediately after making this new file, the system checks a special record, containing a "Journal Entry Number," taking note of the time and date this check is made. Journal Entry Numbers have the form "NNNJMMY." 1d1a2d

"NNN" is a serial number, in the range 1 to z where z is arbitrarily large. 1d1a2d1

"J" is the literal character "J," indicating that the number refers to a Journal entry. 1d1a2d2

"MM" is the month the entry was submitted (e.g., 11 = November). 1d1a2d3

"Y" is the year the entry was submitted (e.g., 9 = 1969). 1d1a2d4

The serial numbers, NNN, are initialized at the start of each month. 1d1a2e

Example: If 4562J119 is the last entry submitted to the Journal in the month of November, 1969 (indicating that 4562 entries were submitted in that month), the next Journal entry would be allocated the number 1J129. 1d1a2e1

Assume that the number in this location at the time of this particular access was 457J119, and the exact time of access was 1451:30, on 11/13/69. Once this number has been secured, the system

updates the latest Journal Entry Number in this location (to 457+1 = 458). 1d1a2f

The system now copies the file (JOURNAL,T1) to a new file -- a Journal entry with file name 457J119. It sets the status of this file to public read-only, and notes the time and date of completion of making this Journal entry: 1451:45, 11/13/69. 1d1a2f1

Once this Journal entry has been made, the system returns a message "FILE (X,X1) ENTERED TO JOURNAL AS NUMBER 457J119 AT 1457:45" to the sender (user X). 1d1a2f2

This message remains on user X's display until a command accept (CA) is entered. Entering the CA releases the file (X,X1) for normal operations, and redisplays the file. User X is now free to continue his normal work. 1d1a2f3

2. Data Assembly Procedures at Input Time 1d1b

The time an entry is submitted to the Journal is an opportune time to capture data associated with the entry. The Journal entry procedure will contain additional operations, in which the system interrogates the user to obtain an abstract and special descriptor tags for the entry. The abstract will be used in the Journal catalogue. Descriptor tags will be used for retrieval of entries. 1d1b1

3. Collection System 1d1c

Part of the Journal entry system gives the user special aids for assembling the entry before actual submission. These are compound operations, combining several simpler ones. These simpler operations include file merging and the "executable statement" capability. 1d1c1

B. Linkages 1d2

Special linking features will be added to NLS to support the DSS needs. One of the most important classes of these new features concerns NLS links. 1d2a

1. "Link" as an NLS Entity 1d2b

In the current NLS a link is a simple text construct; it is not a special entity, in the way that characters, words, and statements (for instance) are entities. 1d2b1

There is no command DELETE LINK in current NLS. A link may be deleted using the normal DELETE TEXT command, requiring two bug selections, one at each of the link parentheses. 1d2b1a

A special NLS entity "link" will be added to NLS. This will be of particular importance in combination with indirect linking and executable statement operations. 1d2b2

To insert a link, the new command INSERT LINK is used. This command requests user input of data necessary to construct the link, and organizes these data in the appropriate syntax (see below).

ld2b3

2. New NLS Link Syntax ld2c

a. Additional Link Data ld2c1

Additional data will be added to the current NLS link construct. These data are (a) link type, (b) time and date the link was first constructed, or last "stamped," and (c) improved resolution to identify link referents. ld2c1a

Link type data are one or more descriptors, being a simple text name, or collection of names, indicating membership of a class, or classes. ld2c1b

Example: Possible link types would be "footnote," "comment," "rebuttal," "owner-evans," etc. A link "owner" could be different from the owner of the file in which the link resided. The definition of these types and their respective mnemonics would be determined by agreement among DSS users. ld2c1b1

A most important addition to NLS links will be the added power to refer to ANY entity. In the current version of NLS, a link may point only to statement entities. ld2c1c

With greater resolution for link references, for instance, a link may be constructed to refer specifically to another link. This is the basis for indirect linking, to be discussed below. ld2c1c1

b. Possible Syntax for New NLS Link Entity ld2c2

<TYPE;DATE,TIME> (USERNAME, FILENAME, LOCENTITY:VIEWSPECS) ld2c2a

TYPE is any number of descriptor mnemonics defining the type of the link. Each descriptor would be delimited by a comma. ld2c2b

MMDDYY HHHH:SS is the date and time the link was created, or the date and time the link was last "stamped," in the format <month, day, year, hour,

second>. 1d2c2c

At any time, the link's owner may initialize the time and date for the link, using a date-time "stamping" command. 1d2c2c1

USERNAME, FILENAME, and VIEWSPEC have the same meaning as in current NLS links. 1d2c2d

LOCENTITY identifies a specific target entity. Detailed syntax for the LOCENTITY may be arbitrarily complex. The following example indicates a simple statement-number syntax. 1d2c2e

c. Example 1d2c3

<comm,urg,Evans;09/17/69 0014:44>
(Engelbart,plans,m-P:xi) 1d2c3a

TYPE is "comm,urg,Evans" 1d2c3a1

DATE,TIME is "09/17/69 0014:44" 1d2c3a2

USERNAME is "Engelbart" 1d2c3a3

FILENAME is "plans" 1d2c3a4

LOCENTITY is "m-P" (the marker "P") 1d2c3a5

VIEWSPECS are xi, meaning display only one line of top-level statements, and switch on the content analyzer. 1d2c3a6

This link refers to the entity with marker "P" affixed ("m-P") in the file ":plans" owned by user name "Engelbart." It points from a comment ("comm") that is urgent ("urg"), and should be brought to the attention of user name "Evans." The link was last stamped 09/17/69 at 0014:44. 1d2c3b

3. New VIEWSPECs for Links 1d2d

Increased link complexity demands more powerful VIEWSPECS to simplify displaying the link construct, so links do not make the remainder of the text illegible. 1d2d1

Additional VIEWSPECS will be available for totally or partially suppressing display of the link construct. For instance, the user could control which fields in the link were displayed at the link's location in a statement (this VIEWSPEC would apply to the entire display). If the link was to be totally suppressed, an additional VIEWSPEC would allow the user to control whether or not special "link markers" were displayed at the link's normal location. 1d2d2

A user would interrogate an individual link marker, to display the particular link represented by that marker, without displaying all links. 1d2d3

4. Links Not Embedded Directly in Text 1d2e

Because of the "stationary target" concept and the frequent need to attach links to existing Journal entries, it will be necessary to have a new NLS command to enable a user to associate an NLS link with any selected text entity, but have that link displayed only as an overlay to the file, rather than an integral part of the normal text. Link markers, similar to those used for backlinking, will be used to indicate the presence of one of these links. New NLS commands will be available to enable the user to control the display of the link and markers. 1d2e1

5. Indirect Linking 1d2f

Once it is possible to "aim" a link at any arbitrary entity, such as another link, or at a simple character in a statement, indirect linking becomes possible. The following example illustrates detailed operation for indirect linking. 1d2f1

Example: The following link is displayed in a statement of the file (Evans,ddd):
<comm;>(Engelbart,plans,m-P:). Note that the date-time field has been suppressed by the new link VIEWSPECS described previously. This link is embedded in a statement (or branch) constituting a comment on its DIRECT target. 1d2f1a

In the file (Engelbart,plans) there is a marker "P" affixed to a character just preceding another link, as follows: <P>xx yyy cc

<comm;>(Evans,rrr,12b:w). This link is a comment on 12b in the file (Evans,:rrr). 1d2f1b

Use of the new command JUMP INDIRECT LINK, with the original link as operand, causes the statement 12b to be displayed under the control of VIEWSPEC "w" (all lines of all statements). 1d2f1c

6. Backlinks 1d2g

The most important additions to existing NLS linking features for use in the DSS are the backlink operations. 1d2g1

Backlinking means that a special executable link marker is deposited in the referent being pointed at by a link. This enables a user, viewing the referent entity, to "JUMP BACKLINK" and display the entity containing the original link. 1d2g2

The existence of an NLS link reference to any displayed NLS entity will be indicated by special backlink markers. Display of these markers will be under user control in a manner similar to link markers, described previously. 1d2g3

A user may interrogate a backlink marker, to have data on the source entity displayed. Execution of the new command JUMP BACKLINK with a backlink marker as operand displays the source entity at the top of the display. 1d2g4

Indirect backlinking will also be available. Indirect backlink jumping means that a user executes JUMP BACKLINK INDIRECT, and the system displays the statement containing the link that points at the source of the backlink marker entered as the operand for this command. 1d2g5

7. Remote Linking 1d2h

The basic concept for remote linking is that of attaching the "head" of a link to its referent entity, followed by insertion of the link itself in the source entity, remote from the referent, at some later time. 1d2h1

This may be accomplished by the following steps: ld2h2

(1) Assigning a temporary marker to yet another entity, "link referent" ld2h2a

(2) Depositing that marker at the appropriate location in the referent statement ld2h2b

(3) Later, while inserting the basic link construct in the source statement, calling for the referent entity data to be inserted in the link by using a special INSERT REFERENT DATA command, entering the referent marker as operand. ld2h2c

This type of operation depends upon each user having at least two NLS files open simultaneously. If links and backlinks are considered to be completely symmetrical, this procedure may be used interchangeably with the conventional INSERT LINK command.

1d2h3

C. Copying a Journal Entry 1d3

A problem arises when a Journal entry, stored as a colon file, is copied to a new filename. All backlink markers are retained, but the links generating these markers continue to refer to the original Journal entry, and do not point at the new file. Thus an additional type of backlink is produced -- one that has no forward-pointing link associated with it. 1d3a

These asymmetrical backlink markers make it possible to jump to files and entries that referred to the original entry. They may be deleted if judged to be inappropriate for the new file. 1d3a1

At the time the new file is created, the system will automatically insert a link in the file's header statement, pointing at the header statement in the Journal entry from which it has been copied, and depositing a backlink marker in the header of the Journal entry. 1d3b

D. Ordered Sets 1d4

A set is a special new NLS entity -- it is a collection of other entities (e.g., of characters, files, statements, links, other sets, etc.). The design and implementation of operations associated with sets is a complex problem. The following indicates what seem to be the most promising possibilities. 1d4a

An "ordered" set has a specified order associated with its member entities. Sets are given unique names for identification. For convenience, a set will be attached to a "parent" file, selected arbitrarily by the user. *[Evans,XXX]* is the set named "XXX" owned by the user name "Evans." Set names are similar to statement names, except they must be unique over the entire universe of a user's files -- it is not possible to have a set named "XXX" associated with the file :ccc and another set "XXX" associated with the file :ddd, if both :ccc and :ddd are owned by the same user. However, different users may own sets with the same name. 1d4b

1. Admission to a Set 1d4c

Other NLS entities, including other sets, way be "admitted" to a set, using the command "ADMIT <entity> TO SET <setname>", and entering the appropriate operands. 1d4c1

"Entity" is the NLS entity selected or specified by the user; "setname" is the name of an existing set -- the set to which the entity is to be admitted. 1d4c1a

Not only entities, but specific views and specific subsets of entities, may be admitted to a set. 1d4c2

Example: The first line of the first two levels of statements in a file satisfying a given content pattern, may be admitted to a set. The remainder of that file, unless specifically admitted on another occasion, does not belong to the set. 1d4c2a

2. Direct and Indirect Use of Sets 1d4d

There are three modes for using sets: "normal," "direct," and "indirect." 1d4d1

"Normal" mode corresponds to normal NLS usage in which the set entity has the same status as normal NLS entities (words, characters, etc.). 1d4d2

Thus in normal mode, the command DELETE SET erases the set whose name is given as an operand. Note that the set is erased, not the members of the set. 1d4d2a

In "direct" mode, operations performed on a set produce changes in the actual entities admitted to the set. 1d4d3

Example: A (nypothetical) command "DELETE WORD m-spec IN SET /evans,X/" is entered; "spec" is an NLS marker name. Upon execution, in direct mode, all words so marked in the entities that are members of the set /evans,X/ will actually be deleted. That is, they will be deleted in the same sense as if the user displayed each entity in the set containing the marker, and manually

deleted the marked word, followed by the command OUTPUT FILE. 1d4d3a

Entities changed through operations performed on sets in "direct" mode remain changed after the system is returned to "normal" mode. 1d4d3b

In "indirect" mode, operations performed on entities that are members of a set (by using the set name itself as the operand) produce changes in those entities ONLY while the user views them "through" the set. 1d4d4

For instance, if in the previous example the same operation was performed in "indirect" mode, the marked words would not be deleted in the files containing the marked entities in question, but would only "appear" to be deleted when the viewer was working with the set *[evans,X]* controlling the entities he could display. This appearance would be negated as soon as the user returned to display any member-file in normal mode. 1d4d4a

3. Open and Closed Sets 1d4e

a. Closed Sets 1d4e1

A closed set is one whose membership is specified explicitly, i.e., there is a finite fully determined membership list associated with the set. For example, statement entities might be specified by a list of NLS links. There are three types of closed sets: frozen, unfrozen, and mixed. 1d4e1a

A frozen closed set retains the exact content and structure of each entity, in the state in which it was originally admitted to the set. Even if (say) a member statment is deleted, a "copy" is retained in the set. 1d4e1a1

An unfrozen closed set retains a finite membership, but permits each member entity to adopt its latest actual state. For example, a whole file, containing three statements admitted to an unfrozen closed set on day 1, subsequently undergoes major modifications. If the set is used as an operand on day 3 (after

the modifications), the file's state at that time is used. 1d4e1a2

A mixed set contains entities whose frozen/unfrozen status is determined individually. In other words, a set may contain some entities whose original status is retained, and some whose status is the latest status of the entity itself. 1d4e1a3

b. Open Sets 1d4e2

An open set is one whose membership is not fixed by explicit identification of its member entities, but rather by the specification of conditions to be met to admit member entities. 1d4e2a

For example, an open set's membership may be determined by those statements in a given file universe that satisfy a given content pattern. 1d4e2b

On day 1, this may yield a different membership than on day 4, if modifications were made to files in that universe during this period. 1d4e2c

4. Set Operations 1d4f

There are two major and distinct classes of operations associated with sets -- operations on sets, and operations within sets. The distinctions between these classes are important. 1d4f1

a. Operations on Sets 1d4f2

Operations on sets use entire sets as operands. 1d4f2a

Simple Operations on Sets 1d4f2b

These operations include the standard NLS operands -- INSERT, DELETE, REPLACE, etc., in addition to a new class of commands -- set-theoretic operations. 1d4f2b1

INSERT SET creates a new set. 1d4f2b1a

REPLACE SET makes it possible for a user to make a new set as the union of one or more

existing sets, and to simultaneously delete the original sets (their names, not members). ld4f2b1b

DELETE SET erases the set (but not its members). ld4f2b1c

Set-Theoretic Operations on Sets ld4f2c

There will be new NLS commands to enable a user to perform set-theoretic operations on sets. The following set-theoretic commands will be available: UNION, INTERSECTION, COMPLEMENT, and DIFFERENCE, where each operation has its usual mathematical meaning. ld4f2c1

b. Operations Within Sets ld4f3

Operations within sets have entirely different meanings from operations on sets, and from operations on member entities outside the influence of the set construct. ld4f3a

When under the control of operations within sets, the conventional NLS commands take on the following meaning: ld4f3b

MOVE: Change the ORDER of member entities in the set. ld4f3b1

DELETE: Remove the operand-entity from membership of the set. ld4f3b2

COPY: Include the operand-entity once more in the set membership (in a different position within the set's order). ld4f3b3

INSERT: Admit the operand-entity to membership in the set. ld4f3b4

REPLACE: Replace the member entity selected as operand with the entity selected. The entity selected as a replacement may or may not be a member of the set. ld4f3b5

E. Executable Statements ld5

An executable statement will be a new text construct, using the current NLS statement as a basis. NLS commands will be pre-specified as a text string in an executable statement. They will be executed by using the command EXECUTE STATEMENT, giving the statement number of the statement as operand. 1d5a

An executable statement will be the means to effect compound or concatenated operations, including set operations. The structure and meaning of the executable statement features can best be illustrated by examples. 1d5b

Example: The following is an executable statement. 1d5b1

```
(XXX)  (evans,sss,12:x) (Engelbart,plans,2:w)  E C
CA /"retrieve "/ OR /"Retrieve"/ ; CA
(evans,rrr,:wi)  END
```
1d5b1a

(1) By activating the command EXECUTE STATEMENT, and entering the operand "XXX" (the name of the executable statement), followed by a single CA, the first link will be executed as if JUMP FILE LINK was used with that link as its operand. 1d5b1a1

(2) The user views the file (evans,sss) with statement 12 at the top of the screen, displaying only the first lines of subsequent top-level statements in the file. 1d5b1a2

(3) A second CA causes the second link to be executed. 1d5b1a3

(4) The user views the file (engelbart,plans), with statement 2 at the top of the screen, displaying all lines of all statements. 1d5b1a4

(5) A third CA causes the content pattern /"retrieve/ OR /"Retrieve"/ to be compiled, automatically followed by the execution of the last link. Note that the VIEWSPEC "i" in the last link activates the pattern. 1d5b1a5

(6) The result is that the file (Evans,rrr) is searched; all statements containing the text construct "retrieve" or "Retrieve" are displayed. 1d5b1a6

Example: The following executable statement illustrates more complex operations on sets. 1d5b2

```
(YYY) [DOD] = [ARMY] UNION [NAVY] ; [USA] =
[DOD] INTERSECTION [MIC] ;E C CA ["weapon"] ; CA
(Nixon,[USA],:wi) CA  DISPLAY:w OUTPUT FILE
':arsenal'  DELETE SET [DOD] AND SET [USA] END
```
1d5b2a

(1) The command EXECUTE STATEMENT is executed with the operand YYY, the name of the statement. 1d5b2a1

(2) A CA causes a new set "DOD" to be formed as the union of the two existing sets "army and "navy." This set will be attached to the file containing the executable statement. 1d5b2a2

(3) Another CA causes a second set, "USA" to be formed as the intersection of the two sets "DOD" and "MIC." 1d5b2a3

(5) Another CA causes the content pattern "weapon" to be compiled, immediately followed by execution of the link transferring control to the first entity containing the text construct "weapon" in the set "USA" (which is owned by the user "Nixon"). 1d5b2a4

(5) The system searches all entities in this set, and displays, under VIEWSPEC contol "w" (all lines of all statements) those statements containing the text string "weapon". 1d5b2a5

(6) A final CA causes this collection of entities to be output as the new file ':arsenal.' Another CA causes both the sets (as distinct from the set membership) [USA] and [DOD] to be deleted. 1d5b2a6

Example: The following executable statement illustrates how the member entities of a set may be displayed. 1d5b3

```
(ZZZ) DISPLAY:w [HEREANDNOW]  END
```
1d5b3a

By giving the command EXECUTE STATEMENT with ZZZ

as the operand, followed by a CA, all entities in the set "HEREANDNOW" will be displayed, under VIEWSPEC control "W" (all lines of all statements). 1d5b3b

Example: The following is an example of simple "chain generation" using an executable statement. 1d5b4

(AAA) MARKER=A1 CHAIN (evans,ss,12:gw) (evans,ss,5:gw (Engelbart,plans,5:wh) END 1d5b4a

By giving the command EXECUTE STATEMENT with the operand "AAA", followed by a CA, the display starts with an all-all view of the branch starting with statement 12 in (Evans,:ss). Normal text operations may be performed on this branch. If a second marker A1 is entered, the all-all view of the branch starting with statement 5 in (evans,:ss) is displayed, and so on. 1d5b4b

Here a marker is used as the means to advance the view along the chain. This permits normal text operations (requiring CA's) to be performed at each view along the chain. 1d5b4c

In all examples, the maximum VIEWSPEC operative on any entity is controlled by the VIEWSPEC assigned to the set member entity itself at the time it was admitted to the set. 1d5b5

F. Entry Descriptors 1d6

Descriptors will be attached directly to Journal entries, either at time of entry to the Journal, or at some later date. These descriptors will cover at least the following classes: 1d6a

(1) Subject matter/type of entry 1d6a1

Examples: comment; message; annouuncement; injunction 1d6a1a

(2) Urgency 1d6a2

Examples: urgent; not urgent 1d6a2a

(3) Names of users whose attention is sought 1d6a3

Example: attention: evans, engelbart. 1d6a3a

(4) Author/source of entry 1d6a4

Example: author: evans; 1d6a4a

(5) Date/time of entry to Journal 1d6a5

Example: entered 9/26/69 1006:30 1d6a5a

G. Interrogation 1d7

Commands will be available to enable a user to interrogate a Journal entry in order to ask the following types of questions: 1d7a

(a) Which Journal entries or other files are pointing at the interrogated entry? 1d7a1

(b) To which sets does the interrogated entry belong? 1d7a2

When interrogating to determine which entries or other files are pointing at the entry, the user will be able to control the universe over which the search for these entries is to be performed. 1d7b

For instance, the user may ask for only those entries that point at the interrogated entry, or are attached by links of a specified type, from entries of another specified type, that were made after a specfied date. 1d7c

Example: Display Journal entries of type "comment" or "injunction" that are attached with link types "urgent" made after 8/12/69 to Journal entry Number XXXXX. 1d7d

Example: Display those members of the set /evans,XXX/ admitted to the set after 10/4/69. 1d7e

H. Miscellaneous New NLS Features 1d8

Numerous new NLS features will have a major effect on the usefulness of the DSS, although they are not designed exclusively for DSS usage. These features include split screens, file merging, new VIEWSPECS, and "file history." 1d8a

1. Split Screen 1d8b

The "split screen" feature generalizes the characteristics of the "freezing" option in the current version of NLS. With a split screen, the user is able to display two different views of the same file, or two different and independent views of any two files, one on each side of the screen. He will be able to work with the displayed information in each "window" as if it was a separate and independent file. The success of this option depends upon having more than one file open for a given user at any given time. The split screen will make interfile editing, and more complex file merging, easy and useful. 1d8b1

2. File Merging 1d8c

The split screen and other new features make the capability for merging any two files to form a third composite file a necessity. In the current version of NLS, only the simplest file merging operation -- appending -- is possible. More useful file merging would include the facility to interleave statements in a specified order, and to transfer pictures from one file to another. 1d8c1

3. File History 1d8d

Keeping track of a file's history becomes more important in the Journal and DSS than in current NLS operations. For this reason a new NLS feature will be added to capture all necessary identification information from the source file every time a file is output or copied. This information may be copied directly from the header statement of the source file, and written into the header statement of the object file at the time it is created. 1d8d1

Example: The following is an example of a standard file header. 1d8d1a

:XVIII, 9/26/69 1209:30 DAE; 1d8d1a1

Here :XVIII is the filename; 9/26/69 1209:30 is the date and time the file was last output to the name :XVIII, and DAE are the initials of

the file owner. 1d8d1a2

Suppose the file :XVIII is output to the new file name ":CHAP18". 1d8d1a3

After the operation is completed, the header of the object file (:CHAP18) reads as follows: 1d8d1a4

```
:CHAP18, 9/26/69 1211:45 DAE;
(evans,XVIII,:) 9/26/69 1211:45;
```
1d8d1a4a

The system has rewritten the source file's header data as an NLS link following the object file's conventional header data. Note that as later versions of :CHAP18 are made, data preceding the first semicolon changes. With subsequent copy operations, or output file operations to new filenames, these data from the file :XVIII will be retained in the new file's header, along with all records of subsequent operations. 1d8d1a5

I. Cataloguing 1d9

A catalogue of all entries in the Journal will be maintained, providing the main conventional aid for retrieval of these files. The catalogue will have three main sections: a subject index, a keyword list, and citations for Journal entries. 1d9a

The subject index contains a hierarchical structure of the subjects describing Journal entries, with their respective keywords attached. A user may scan this index and select keywords attached to the subjects that meet his needs. 1d9a1

The Keyword List will contain keywords (as used in the subject index), followed by links pointing at appropriate citations. 1d9a2

The citation for each Journal entry is stored in the catalogue by order of Journal Entry Number. Each citation will constitute an NLS branch, with the Journal Entry Number, and link to the cited Journal entry, as the first-level statement of each branch. 1d9a3

Each such citation branch will contain the entry

number, the source filename, the name of the user submitting the entry, the date and time when the entry was submitted, and a list of descriptors for entry. 1d9a3a

These data will be stored in a manner that makes them useful for further NLS operations. For example, the data on source filename is stored in the form of a conventional NLS link referring to the source file. Similarly, each catalogue entry contains a link to the Journal entry itself. 1d9a3a1

1. Retrieval System Based on the Journal Catalogue 1d9b

The existing NLS keyword retrieval system will be extended for use as the basic retrieval tool for operations on the catalogue. The major drawback of the current system is that lists of citations can be assembled only from within a single file. 1d9b1

For the DSS, this system will be modified to operate across an arbitrary number of files. Such operations, of course, depend upon other features discussed previously (e.g., file merging, the capability of having more than one file open at any instant, etc.). 1d9b2

The standard keyword statement, which currently uses statement names as keyword arguments, will be changed to use full NLS links as keyword arguments. 1d9b3

Example: 1d9b3a

(key3) This is keyword three *
(JOURNAL,135J99,:) (Journal,146J99,:) 1d9b3a1

The user will then have the following options: 1d9b4

(1) Assemble the citations derived from a selection of keywords from one or more files (which may themselves be stored in several catalogue files), as a list in one file, and use the standard JUMP LINK command to view the actual Journal entries cited, one by one. 1d9b4a

(2) Ask for consecutive display of the actual

Journal entries cited, under the control of the VIEWSPECS in the keyword referent links. Consecutive entries cited would be displayed as if part of the same file. 1d9b4b

This operation could be accomplished by special new NLS machinery, or by combining the capabilities of executable statements and indirect linking. 1d9b4b1

In all cases, all current NLS keyword options, including the allocation of weights to keywords, will be available. 1d9b5

```
':4872', 09/28/70 |7|8:46 MGC ;  :ADSS, 07/|2/70 |8|9:33 DGC ; EDITING
CHANGES DONE .COD[2|B]=||4B; .RTJ=0; .DSN=1; .LSP=0; .DLS=1; .HLN=3;
.PGN=156; .HED="                                                    4872 DGC
12JUL70
7101 ROME FINAL REPORT:  Appendix B
THE DSS AND THE JOURNAL";   .DPR=0;
```

Appendix C
REFERENCE MANUAL FOR PERIPHERAL EQUIPMENT

1

I Introduction 1a

This appendix is an addendum to the previous Hardware Reference Manual, Appendix B of Ref. 3. It consists of a programmer's reference manual for the following equipment: 1a1

A line printer (replacing the line-printer description contained in the previous manual) 1a1a

An inter-core controller for transfers between 940 core and external core ("Xcore") 1a1b

A Network interface connecting the 940 to the ARPA Network via the Interface Message Processor (IMP) 1a1c

A precision clock. 1a1d

II Line Printer 1b

A. General Information 1b1

The printer is a Data Products Model M600-11A with 96 characters and a printing speed of about 340 lines per minute. It will accomodate paper from 2-1/2 to 18-1/2 inches in width. Character spacing is 10 per inch and line spacing is 6 per inch. The maximum number of characters per line is 132. 1b1a

The printer is controlled by EOM instructions and a "unit reference cell" (URC). The URC points to a print buffer resident in core that contains data and control codes. An SKS instruction indicates "printer ready" and an interrupt indicates "end of operation," either normal or error. Error conditions are detected by the controller and an error code written in the URC. 1b1b

The cells immediately following the URC in core are called "URC+1," "URC+2," etc. 1b1b1

Fixed core assignments for the printer are: 1b1c

URC	10	
Interrupt	211.	1b1c1

B. EOM and SKS Codes 1b2

The EOM codes are: 1b2a

20230106	Initiate	
20230406	Reset.	1b2a1

The "initiate" EOM starts the printer with the word and character designated by the contents of the URC at the time the EOM is given. 1b2a2

The printer controller continues to process the printer buffer until an illegal character or end-of-buffer code is read, or until a "reset" EOM is issued. 1b2a2a

An "initiate" EOM given while the printer is busy is ignored. 1b2a2b

The "reset" EOM immediately terminates all printing and returns the system to a reset state. 1b2a3

A "reset" EOM given while the printer is disconnected is ignored. 1b2a3a

One SKS code is provided for the printer. The code is 1b2b

04030106	Skip on ready.	1b2b1

This SKS skips if the printer is ready to begin operation. If the printer is not ready, an interrupt is issued when it is made ready. 1b2b2

C. Unit Reference Cell 1b3

The URC associated with the printer system has the following format: 1b3a

1b3a1

Bits 6-23 contain the absolute address of the first character of the line to be printed (or currently being printed). 1b3a2

Bits 8-23 denote the absolute word address. 1b3a2a

Bits 6-7 indicate the character in the word. 1b3a2b

A 00 code is the leftmost character. The 11 code is not used but is interpreted as the leftmost character. 1b3a2b1

After a line has been successfully printed, the address in the URC is updated to point to the first character of the next line. 1b3a2c

Bits 0-3 are written by the controller with an error code when errors are detected. Error conditions and codes are described below. 1b3a3

Bits 4-5 are ignored by the controller. 1b3a4

D. Print Buffer 1b4

The print buffer is a contiguous sequence of words in core that is interpreted by the printer controller as three 8-bit characters per word. 1b4a

Characters in the print buffer may be either data characters or control characters. 1b4b

The control characters are: 1b4b1

373	(NOP) No operation
375	(EOB) End of print buffer
376	(EOL) End of line
377	(NOP) No operation
015	Shift to lower case and lock
035	Shift to lower case for one character
055	Shift to upper case and lock. 1b4b1a

An EOL or EOB code causes the current line to be printed with any characters already in the line left-justified. 1b4b1b

An EOB code generates an interrupt to the computer after the line is printed and any spacing action

has been completed. 1b4b1c

The three case-shift codes are self-explanatory. They can appear anywhere within a line of data characters and cause the indicated case-shift actions. 1b4b1d

In addition to the explicit control characters, the first character in each line is interpreted as a paper-feed code. These codes are as follows (the word "space" here refers to line spacing, not the "space" character): 1b4b2

020	Space 1 line
021	Space 1 line
022	Space 2 lines
023	Space 3 lines
024	Space 4 lines
025	Space 5 lines
026	Space 6 lines
027	Space 7 lines
000	Space on channel 0 of format tape
001	Space on channel 1 of format tape
002	Space on channel 2 of format tape
003	Space on channel 3 of format tape
004	Space on channel 4 of format tape
005	Space on channel 5 of format tape
006	Space on channel 6 of format tape
007	Space on channel 7 of format tape. 1b4b2a

The action indicated by the space code takes place before the line is printed. 1b4b2b

Two successive spacing operations can be caused by sending one of the above space codes followed by "end of line" (376), then another space code. 1b4b2c

If no spacing is desired, as when printing the top line on a page, a no-op code (377) should be sent in the first position of that line. 1b4b2d

Channel 1 of the format tape is used for "top of form." The number of lines on a page is normally set to 60. 1b4b2e

Except for the first character, the print buffer contains only printing characters (including space

characters) and control characters. Any other character codes in the print buffer are considered illegal and cause an error condition. 1b4b3

Print buffers may be as large as desired, but no relocation mapping is provided. If a buffer is to extend across a page boundary, the software system must ensure that the two pages are consecutive in memory. 1b4c

E. Error Conditions 1b5

On the detection of any error, an interrupt is issued and the error code is written in the URC. 1b5a

The error codes and conditions detected are: 1b5b

000	No error
101	Illegal character code
110	Printer not ready
111	Excessive time.

1b5b1

Zeros in the error-code bits of the URC after an interrupt indicate a normal interrupt (printer made ready or EOB). 1b5b2

The 101 code indicates that an illegal character has been detected in the print buffer. 1b5b3

The 110 code indicates printer off-line, paper out, or ribbon failure. 1b5b4

The 111 code indicates that in a normal printing operation, excessive time has been required for printing a line. 1b5b5

The timer is normally set for 2.5 seconds. This error indicates printer failures not detected by other printer error circuits.

1b5b5a

F. Character Codes 1b6

The printer character codes are given below. The case printed is determined by the shift-control character. 1b6a

CODE	UPPER	LOWER	CODE	UPPER	
LOWER					1b6a1
000	0		040	-	
underbar					1b6a2
001	1		041	J	
j					1b6a3
002	2		042	K	
k					1b6a4
003	3		043	L	
l					1b6a5
004	4		044	M	
m					1b6a6
005	5		045	N	
n					1b6a7
006	6		046	O	
o					1b6a8
007	7		047	P	
p					1b6a9
010	8		050	Q	
q					1b6a10
011	9		051	R	
r					1b6a11
012	null		052		1b6a12
013	=		053	$	1b6a13
014	'		054	*	
+					1b6a14
015	null		055	null	1b6a15
016	>		056	;	
:					1b6a16
017	null		057		1b6a17
020	space		060	null	1b6a18
021	A	a	061	/	
%					1b6a19
022	B	b	062	S	
s					1b6a20
023	C	c	063	T	
t					1b6a21
024	D	d	064	U	
u					1b6a22

025	E	e	065	V	
v					1b6a23
026	F	f	066	W	
w					1b6a24
027	G	g	067	X	
x					1b6a25
030	H	h	070	Y	
y					1b6a26
031	I	i	071	Z	
z					1b6a27
032			072		1b6a28
033	.		073	,	
@					1b6a29
034	)	]	074	(	
[					1b6a30
035	null		075		
&					1b6a31
036	<	←	076	\	
"					1b6a32
037	?	#	077		
overbar					1b6a33

1b6a34

III Inter-Core Controller 1c

A. General 1c1

The inter-core controller controls transfer of data between external core (often referred to as "Xcore") and 940 core. It has two modes of operation: 1c1a

(1) A block transfer mode allows the transfer of blocks of up to 2048 words between any two locations in the two cores. This transfer can be between two locations in the same core. 1c1a1

(2) A short transfer mode allows the transfer of short, fixed-length buffers between fixed locations in 940 core and external core. 1c1a2

Fixed core assignments for the inter-core controller are: 1c1b

URC, 940 core	53
Fixed transfer address, Xcore	100
Interrupt	215.

1c1b1

B. EOM Instructions 1c2

Four EOM instructions are used for the inter-core controller. 1c2a

The EOM codes are: 1c2a1

20230103	Block transfer
20230203	Xcore to 940 fixed transfer
20230303	940 to Xcore fixed transfer
20230403	Disconnect

1c2a1a

The EOM actions are: 1c2a2

Block Transfer -- This EOM starts a variable-length transfer. The number of words to be transferred and the starting addresses in source core and destination core are determined by the contents of three consecutive 940 memory cells starting with the URC. Source and destination may be in the same core. 1c2a2a

Xcore to 940 fixed transfer -- This EOM initiates a transfer of a fixed number of words beginning at a fixed address in Xcore to a location beginning at the URC in 940 core, starting with the URC address in the 940 computer to a fixed starting address in the external core. 1c2a2b

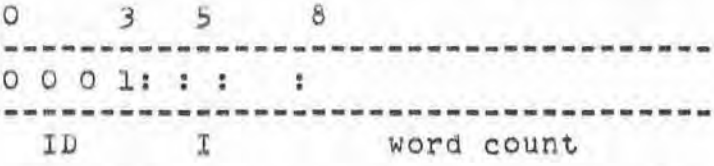
The number of words is determined by a card in the controller and may be set to any number between 1 and 7. The number currently used is 3. 1c2a2b1

940 to Xcore fixed transfer -- This EOM initiates a transfer of a fixed number of words (same number as above) from 940 core to Xcore, with the same fixed locations in each. 1c2a2c

Disconnect -- This EOM terminates any transfer in progress and places the controller in the disconnect state. 1c2a2d

C. Unit Reference Cell 1c3

The URC and the next two cells have the following coding when used to control a block transfer operation: 1c3a



1c3a1

Bits 0-3 contain an identification code. If any other code is detected, the controller disconnects and writes an error code in the URC. 1c3a2

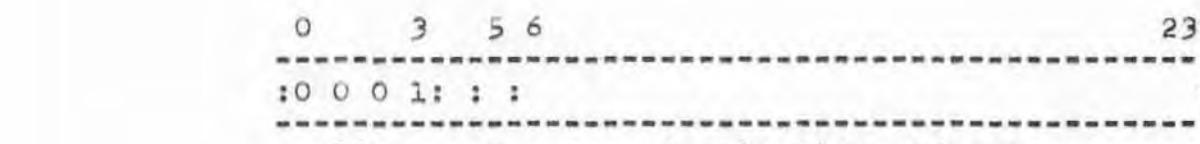
Bit 5 is set to 1 if an interrupt is desired at the completion of the transfer cycle. 1c3a3

Bits 8-23 indicate the number of words to be transferred. 1c3a4

Bits 4 and 6-7 are ignored. 1c3a5

The cell URC+1 contains information relating to the destination of the transfer. It has the following

format: 1c3b



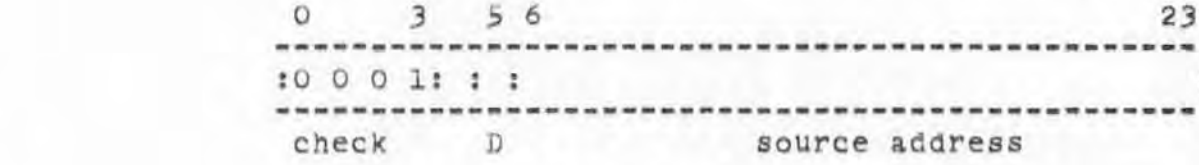
1c3b1

Bits 0-3 contain an identification code as above. 1c3b2

Bit 5 specifies the destination core. A 1 indicates transfer to 940 core and a 0 indicates transfer to Xcore. 1c3b3

Bits 6-23 designate the first address in the destination core. 1c3b4

The cell URC+2 contains information relating to the source for the transfer. It has the following format: 1c3c



1c3c1

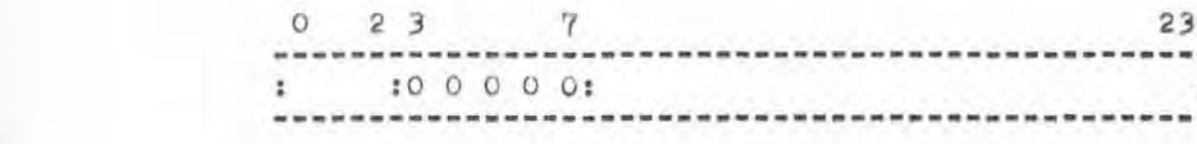
Bits 0-3 contain an identification code as above. 1c3c2

Bit 5 specifies the source core. A 1 indicates transfer from the 940 core and a 0 indicates transfer from Xcore. 1c3c3

Bits 6-23 designate the first address in the source core. 1c3c4

D. Exit Routine 1c4

At the end of any transfer, or when an error is detected, the exit routine is performed. This routine writes the URC and then places the unit in its "disconnect" state. The URC is written with the following format: 1c4a



error word count 1c4a1

Bits 0-2 contain an error code. The errors are reported as follows: 1c4a2

Bit 0 is set to 1 if any error is detected. 1c4a2a

Bit 1 is set to 1 for an error in any of the URC locations (incorrect ID code detected). 1c4a2b

Bit 2 is set to 1 if the controller waited more than 1 millisecond to gain access to the external core. 1c4a2c

Bits 3-7 are set to 0. 1c4a2d

Bits 8-23 contain the contents of the word-count register at the end of the transfer. For a successful transfer this will be 0. 1c4a2e

An interrupt is issued at the end of the exit routine if called for by the URC, or if any error has been detected. No interrupt is issued for the short transfers. 1c4b

IV Network Interface 1d

A. General 1d1

The network interface provides communication between the 940 and an Interface Message Processor (IMP) on the ARPA Computer Network. The interface operates from message buffers in 940 core. A "linked-buffer" scheme permits flexible memory allocation. 1d1a

The interface contains two independent logic systems, the input controller and the output controller. The former receives information from the Network, and the latter sends information to the Network. 1d1b

As seen by the programmer, these two units are almost identical in all aspects except the direction of data flow. Differences between the two are noted in following sections. 1d1b1

The two channels are independent in action, except that they share the same channel into memory. Thus

they cannot make simultaneous core accesses. 1d1b2

Fixed locations assigned to the Network interface are: 1d1c

Receive URC	60
Send URC	70
Receive interrupt	212
Send interrupt	213.

1d1c1

B. Communications with the IMP 1d2

Data moving between the Host and the IMP is in the form of serial bit strings with a maximum length of 8096 bits and a maximum rate of one million bits per second. 1d2a

Details of the communications protocol between the interface and the IMP are covered in Ref. 2. 1d2b

C. EOM Instructions 1d3

EOM Codes are: 1d3a

20230104	Host up
20230204	Initiate receive
20230304	Initiate send
20230404	Reset.

1d3a1

The "host-up" EOM resets the "host-up timer." This is a timer in the interface controlling a signal to the IMP indicating that the host computer is up. If the timer is not reset at least once a second, indication is given to the IMP that the host is down. 1d3a2

The "initiate receive" EOM enables a "receive" operation. Subsequent to this EOM, data received from the IMP will be written in the "receive" buffers. The EOM must be given for each message received. The controller may be left in the "receive enabled" state indefinitely, waiting for a message from the IMP. 1d3a3

The "initiate send" EOM initiates a "send" operation. Data contained in the "send" buffers will be immediately transmitted to the IMP. A "send" EOM must be given for each message to be transmitted. 1d3a4

The "reset" EOM causes both the controllers to

immediately abort any operation in progress and go to the "reset" state. 1d3a5

D. Linked Buffers 1d4

Linked buffers are used for both "send" and "receive" messages. The format of the linked buffer is as follows: 1d4a

The first word of the buffer contains the byte count for the buffer. 1d4a1

If the byte count is zero, the controller goes directly to the next buffer. 1d4a1a

A block of n bytes to be transmitted will occupy the n/3 core addresses immediately following the byte count, since there are three 8-bit bytes in each 24-bit 940 word. When the last byte does not fall on a 940 word boundary, the action depends on the operation: 1d4a1b

In a "send" operation, bytes remaining in the last word are ignored. 1d4a1b1

In a "receive" operation, bytes remaining in the last word are filled with 0's by the controller. 1d4a1b2

The last word of the buffer contains the absolute address of the next buffer. 1d4a2

If the last word contains all 0's in the address field, no more buffers are processed and the operation is terminated. 1d4a2a

The first buffer of a "send" or "receive" message always begins 2 words after the "send" or "receive" URC, respectively (there are two URCs -- see below). 1d4b

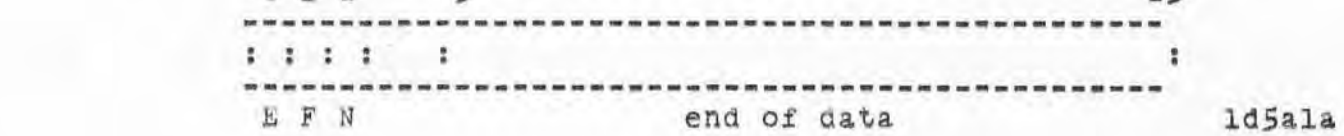
The maximum message length as determined by the IMP is 8096 bits. 1d4c

E. The Unit Reference Cells 1d5

There are two URC locations for the interface, one for "send" and one for "receive." There are two words at

each location, followed by the first message buffer (see above). The URCs have the following format: ld5a

First Word: ld5a1



ld5a1a

Bit 0 -- Error: This bit is set by the controller when an error is detected (see below). ld5a1b

Bit 1 -- List full: This bit indicates that the linked buffers following the URC contain valid data. Its interpretation depends on the operation. ld5a1c

On a "send" operation the controller expects to find this bit a 1, indicating valid data to be transmitted. ld5a1c1

If the controller finds this bit 0 when a "send" is initiated, the "need-new-list" bit will be set to 1 and a "send" interrupt issued. ld5a1c1a

When the "send" operation is completed tha controller resets this bit to 0. ld5a1c1b

On a "receive" operation the controller expects this bit to be a 0, indicating that the buffers are ready to receive a message. ld5a1c2

If this bit is found to be a 1 when a "receive" operation is begun, the "need-new-list bit" will be set and a "receive" interrupt issued. ld5a1c2a

This bit is set to 1 by the controller at the completion of a "receive" operation. ld5a1c2b

Bit 2 -- Need new list: This bit is set by the controller to indicate that the "list-full" bit was not correct at the beginning of an operation . ld5a1d

Bits 5-23 -- End of message: These bits are set by the controller at the end of a "send" or "receive" operation. 1d5a1e

At the end of the "send" operation these bits point to the last word of the last buffer transmitted. This is the zero pointer that terminated the transmission. 1d5a1e1

At the end of a "receive" operation these bits point to the last word filled with data from the received message. 1d5a1e2

Bits 3-4 are not used. 1d5a1f

Second Word: The second word (URC+1) contains error codes and is described below. 1d5a2

F. Interrupts 1d6

Two interrupts are used by the controller, one for "send" and one for "receive." 1d6a

At the normal or error termination of either a "send" or "receive" operation the respective interrupt is issued. 1d6b

G. Errors 1d7

Errors are detected by the controller for both "send" and "receive" operations, and error codes are written into the words following the "send" and "receive" URCs respectively. The "IMP down" error applies to both "send" and "receive," but is reported as a "send" error only.

ld7a

"Receive" errors are reported in the word immediately following the "receive" URC. The errors and bit locations in the error word are: ld7a1

Bit 19 -- Message too long: The message has exceeded the maximum length of 8096 bits. ld7a1a

Bit 20 -- IMP does not respond: During the transmission of a message the IMP pauses for more than 100 milliseconds between bits. ld7a1b

Bit 21 -- List space exceeded: Space in the linked buffers has been exhausted and there are more bits in the message from the IMP. ld7a1c

Bit 23 -- IMP was down: Prior to this message the IMP was down, as indicated by the "IMP-down" line. ld7a1d

"Send" errors are reported in the word immediately following the "send" URC. The errors and bit positions are: ld7a2

Bit 19 -- Message too long: The message has exceeded the maximum length of 8096 bits. ld7a2a

Bit 20 -- IMP does not respond: During the transmission of a message the IMP pauses for more than 100 milliseconds between bits. ld7a2b

Bit 22 -- IMP-ready line is down: This error is reported only when the controller is active -- that is, after a "send" or "receive" EOM has been issued and before the completion of the indicated operation. ld7a2c

Bit 23 -- IMP was down: Prior to this message the IMP was down as indicated by the "IMP-down" line. ld7a2d

V Precision Clock 1e

A. General Information 1e1

The ARC clock system uses a high-stability Hewlett-Packard Model 105B quartz oscillator to drive two accumulators. The accumulators are: 1e1a

(1) An absolute-time accumulator with an output of year, month, day, hour, minute, and second, updated once each second 1e1a1

(2) A relative-time accumulator which consists of a 24-bit binary counter. This counter is advanced once each millisecond. 1e1a2

The short-term jitter of both the absolute and relative accumulators is 10 to 20 milliseconds. This jitter is caused by the variation in the amount of time required to access the 940 core memory. 1e1b

The error caused by the oscillator drift rate is less than 1 second every 250 days. 1e1c

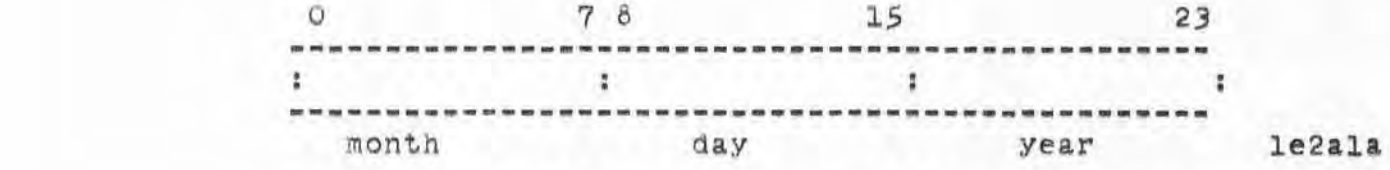
The initial setting of the absolute time is accurate to within 1 second. 1e1d

The programmer has no control over the operation of this unit. Time is written in core whenever the system is operative. 1e1e

B. Word Formats 1e2

The absolute time is written once each second into two words of the 940 computer. 1e2a

The format of the first word is: 1e2a1



1e2a1a

Bits 0-7 contain the month code in straight binary with a range of 1 to 12. 1e2a1b

Bits 8-15 contain the day code in straight binary with a range of 1 to 31. 1e2a1c

Bits 16-23 contain the year code in straight binary with a range of 9 to 99. 1e2a1d

The format of the second word is: 1e2a2

le2a2a

Bits 0-7 contain the hour code written in straight binary with a range of 0 to 23. le2a2b

Bits 8-15 contain the minute code written in straight binary with a range of 0 to 60. le2a2c

Bits 16-23 contain the second code written in straight binary with a range of 0 to 60. le2a2d

The relative time is written once each millisecond into a fixed address. Bits 0-23 contain the relative time in straight binary code with a range of 00000000 to 77777777 (octal). le2b

```
':4873', |0/0|/70 ||37:36 MGC ;  :HDWAP, 07/|2/70 |83|:42 DGC ; EDITING
CHANGES DONE .COD/2|B/=||4B; .DLS=0; .PGN=182; .DSN=1; .LSP=0; .HLN=3;
.RTJ=0; .HED="                                                4873 DGC 12JUL70
7101 ROME FINAL REPORT:  Appendix C
REFERENCE MANUAL FOR PERIPHERAL EQUIPMENT";   .DPR=0;
```

Appendix D .CEN=1;
TECHNICAL DESCRIPTION OF NLS
.CEN=0; 1

Contents .CEN=1;
.CEN=0; 1a

1a4d4

I Introduction 1b

This appendix gives a technical description of NLS and extends the overview given in Sec. IV-E of the main body of this report, covering the utility routines, command specification, and command algorithms used by NLS. 1b1

In addition, the special-purpose languages (SPLs) for command specification, content analysis, and string construction, which are used in large sections of NLS, are discussed in some detail. 1b2

This appendix assumes that the reader is familiar with NLS from the user's viewpoint to the level of the NLS's User's Guide.

1b3

II Utility Routines 1c

The utility routines in NLS fall into two categories, dealing with the overlay system and with file handling. 1c1

The routines in the overlay system provide mechanisms for changing the collection of pages of code in the address space of the program; the file-handling routines provide mechanisms for referencing and changing the actual data base. 1c1a

A. Overlay System in NLS 1c2

1. General 1c2a

The logical structure of the overlays in NLS is a tree structure, with the most widely used code residing in the overlays near the root. 1c2a1

An overlay is confined to a single page, in order to make efficient use of the paging mechanisms of the 940. 1c2a2

2. Implementation 1c2b

The overlay structure is implemented through two tables and several procedures which use them to manipulate the relabeling. 1c2b1

For a given page of program, there is an entry in each table. The index of the entries for the page is the same in both tables and is called the "overlay number" of the page. 1c2b2

One table gives the relabeling byte for the page, while the other gives the overlay number of the parent overlay and the position in the address space that the page should occupy. 1c2b3

The entries in the second table have a POP code in addition to the other information. To relabel in an overlay (and the overlays above it in the tree), the instruction corresponding to that overlay in the second table is executed. 1c2b4

If a call is to be made to a procedure in another overlay that occupies the same logical position in the address space as the calling routine, the call is split into two instructions. 1c2b5

These correspond to the execution of two POPs, the first of which "selects the overlay" and the second of which gives the address to branch to in that overlay. 1c2b5a

Two cells are used in the program to keep a copy of the relabeling. 1c2b5b

When an overlay is selected, the overlay tables are used to update these words without changing the actual relabeling. 1c2b5b1

This change is made when the second POP is executed and after the destination address has been read. 1c2b5b2

On a call such as this, the overlay number of the calling routine, as well as the calling address, is saved on a stack. 1c2b5c

This allows the overlays to be restored to their status before the call when the called routine returns. 1c2b5d

The routines that change the relabeling are in the overlay at the root of the tree, and are thus always available. 1c2b6

In general the root overlay contains utility routines for basic functions, such as changing relabeling and accessing elements of the file. 1c2b7

B. NLS Random-File Structure and Handling 1c3

1. General Considerations 1c3a

The present format and structure of NLS files was determined by certain design considerations. 1c3a1

It is desirable to have virtually no limit on the size of a file. This means that it is not

practical to have an entire file in core when viewing it or working on it. 1c3a1a

A goal in the design was to make the time required for most operations on a file independent of the length of the file. That is, small operations on a large file should take roughly the same time as on a small file. In this way the user and the system are not penalized for large files. 1c3a1b

The system had to include graphic statements, and perhaps other forms of data, as well as text. 1c3a1c

As a result of these considerations, a random-file scheme was chosen. Each file is divided into logical blocks that may be accessed in a random order. There are several different types of blocks, and each type has its own structure. 1c3a2

2. File Structure 1c3b

An NLS file is made up of a header and up to a fixed number (currently 66) of 1024-word file blocks. 1c3b1

a. The Header Block 1c3b2

In each file, there is a header block that contains information about that particular file. 1c3b2a

The header block remains in memory while the file is in use. 1c3b2b

The header includes the following information: 1c3b2c

(1) General information regarding the file, such as the following: 1c3b2c1

(a) The date of creation of the file 1c3b2c1a

(b) The file owner's user number (identifies the user who created the file) 1c3b2c1b

(c) The number of words in the file header block 1c3b2c1c

(d) The initials of the user who last wrote the file out 1c3b2c1d

(e) The date and time at the last writing lc3b2c1e

(f) The name-delimiter characters lc3b2c1f

(g) The average length of statements in characters lc3b2c1g

(h) The total number of statements generated in the life of the file. lc3b2c1h

(2) Status tables for the file blocks. lc3b2c2

The first and largest status table is the random file block status (RFBS) table. lc3b2c3

Each entry in the RFBS table corresponds to a random file block, and indicates the status of that block. The file header is file block zero. The number in the RFBS entry has one of the following meanings: lc3b2c3a

ZERO: The block is not allocated, and does not exist. lc3b2c3a1

POSITIVE: The block is allocated, and is in memory rather than on the secondary storage device. The positive number is the actual starting address for the block. lc3b2c3a2

NEGATIVE: The block is not in core. If the entry equals -1, then the block is allocated, but has not been initialized. In the case of text blocks, -2 indicates that the block contains no garbage statement data blocks, and need not be garbage-collected. Otherwise the number is the negative of the used-word count. lc3b2c3a3

A given file block has only one type of information, such as structure or text. There is a separate status table for each type of file block. These are called secondary status tables. lc3b2c4

An entry in such a table has one of the following meanings: lc3b2c5

ZERO: The block is not allocated. 1c3b2c5a

NON-ZERO: The value is the block number, that is, the entry into the RFBS for that block. 1c3b2c5b

There are secondary status tables for structure, text, graphics, and keyword types of file blocks. The internal structure of these different types of blocks is discussed in the following sections. 1c3b2c6

The use of separate status tables avoids references to absolute locations in the file and reduces the number of bits required to specify the location of a particular piece of information. 1c3b2c7

Pointers to various elements (structural, textual, etc.) consist of two fields: a secondary status-table index and an address giving the start of the element relative to the start of the block. The status table entry contains the number of the block, from which its absolute address can be computed. 1c3b2c7a

Fewer bits are required, since the range of secondary status-table indexes is smaller than the range of possible file-block numbers. The greatest gain from this is in the identifier for a ring element, since a file can have only eight structure blocks in the current configuration of NLS. 1c3b2c7b

In spite of this, the use of the separate status tables is of questionable value. 1c3b2c8

Value of Avoiding Absolute Addresses: By avoiding absolute addresses in the file it is possible to move a block to a new location in the file simply by changing a status-table entry. Such a move can be valuable if the file has become sparse and needs to be compacted. 1c3b2c9

If absolute addresses were used, then all references to the block would have to be changed, but it can be argued that such a

process need only be done on rare occasions and hence its efficiency is not crucial. 1c3b2c9a

Moreover, sufficient backpointers exist so that the process of modifying references would not be difficult (although it might be lengthy). 1c3b2c9a1

Value of Fewer Bits in Pointers: The economy of bits in pointers is a stronger argument for the use of secondary status tables. However, the total savings per ring element (with the current size limits on files) is only six bits. 1c3b2c10

Disadvantages of Secondary Status Tables: Space in the data page is used by the tables (which are always in core) for information that would not be necessary if absolute addresses were used. 1c3b2c11

Their use places arbitrary limits on the number of file blocks of a particular type. 1c3b2c11a

For example, it is possible to exhaust the structure blocks when the file actually contains room for more information. If absolute addresses were used, then blocks of a particular type could be allocated as needed, with a limit only on the total number of blocks rather than a limit on each type of block. 1c3b2c11a1

If further consideration confirms that the secondary status tables should be eliminated, it will not be a difficult task because of the methods used for accessing information in the files. 1c3b2c12

These methods are discussed in a later section; first the remainder of the file structure must be described. 1c3b2c12a

b. File-Block Format 1c3b3

Each random file block has an eight-word header. This header contains the following: 1c3b3a

(1) The checksum of the block 1c3b3a1

This is computed before the block is written, and verified when the block is read. In addition, if room in core is needed for a block, then any block in core that has not been changed may be overwritten without copying it to the file. The checksum provides an easy means of testing whether the block has been changed. 1c3b3a1a

(2) The used-word count (always greater than the header size) 1c3b3a2

(3) The block type, to indicate whether the block is text or structure 1c3b3a3

(4) In structure blocks, the free-list pointer; in text blocks, the garbage-collection flag, indicating whether there are garbage SDBs (statement data blocks) in the block. 1c3b3a4

(5) The secondary status-table index number. 1c3b3a5

c. Structure Blocks 1c3b4

The internal structure of NLS files is a ring structure representing a tree structure. Each node in the ring corresponds to a statement, and contains pointers to the "first son" (called the sub) and the "first brother" (called the successor). The last node in a list contains a flag marking it as the tail and points to the father as its successor. 1c3b4a

The nodes in the ring are kept in four-word ring elements. 1c3b4b

Each structure block contains 254 ring elements. There can be up to eight structure blocks in a file, but not all need be allocated. 1c3b4c

Each ring element in an allocated block either is associated with a statement in the structure of the file or is on the free list for the block. 1c3b4d

A free list consists of a chain of pointers,

starting in the block header and ending with a zero pointer. (As used here a pointer is an address relative to the start of the block.) The pointers are in the first word of the four-word element, and the other three words are zero. 1c3b4d1

A free list is entirely contained within a single block in order to minimize file references. 1c3b4d2

A ring element associated with a statement contains the following information: 1c3b4e

(1) Flags indicating whether the statement 1c3b4e1

(a) has a name or not 1c3b4e1a

(b) has been tested against the current content-analyzer pattern 1c3b4e1b

(c) passed the pattern, if it has been tested 1c3b4e1c

(d) is the head of its plex 1c3b4e1d

(e) is the tail of its plex 1c3b4e1e

(2) A pointer to the text for the statement 1c3b4e2

(3) A pointer to the picture associated with the statement if there is one 1c3b4e3

(4) A pointer to the sub for the statement (or a pointer to the statement itself if there is no substructure) 1c3b4e4

(5) A pointer to the successor for the statement

1c3b4e5

(6) The hash of the name of the statement if it has a name. 1c3b4e6

A ring element is pointed to by a permanent statement identifier (PSID). 1c3b4f

This is an 11-bit integer between 0 and 2047. 1c3b4f1

The three high-order bits give the structure-block number (entry into the RSVST table), and the eight low-order bits determine the location within the block. 1c3b4f2

The PSID of a statement remains unchanged as long as that statement is in the file. That is, the PSID is not changed by textual or structural editing of the file. When the statement is deleted, that same PSID may later be used to identify a different statement. 1c3b4f3

Every file has at least one ring element in its structure, namely the element for the origin statement (root of the ring, first statement in the file), which always has PSID zero. 1c3b4g

d. Text Blocks 1c3b5

In addition to the header, a text-type file block is made up of an arbitrary number of statement data blocks (SDBs) and an area of free storage. 1c3b5a

The free storage area at the end of the file block is simply a number of words available for use in creating new SDBs. 1c3b5b

An SDB is a variable-sized block of words with a six-word header. 1c3b5c

The header contains the following information: 1c3b5c1

(1) The number of words in the SDB. 1c3b5c1a

(2) A flag indicating whether the SDB is unused (i.e. garbage to be collected by the garbage collector) 1c3b5c1b

(3) The PSID of the statement 1c3b5c1c

(4) The date and tne time when the SDB was created and the initials of the user who created it 1c3b5c1d

(5) The number of characters in the statement 1c3b5c1e

(6) The position of the first character in the statement that is not part of the name. (Set to 1 if the statement does not have a name.) 1c3b5c1f

The words following the header contain the text of the statement, three characters per word. The text includes an end character (code 377B) on each end of the statement. The last word is filled to a word boundary with end characters. 1c3b5d

The characters in a statement are explicitly numbered, the first end character being number zero. 1c3b5e

A two-word entity consisting of a PSID and a character count is called a T-pointer, and indicates a particular character within the file. 1c3b5f

A T-string is a string of text within a single statement. 1c3b5g

The text-editing routines make use of T-pointers and T-strings. 1c3b5h

e. Graphics Blocks and Keyword Block 1c3b6

The format of the information stored in these blocks will be described in the sections dealing with the vector package and the keyword system. 1c3b6a

3. File Handling 1c3c

a. Core Tables and File Input/Output 1c3c1

The random files are read into core by blocks. Two pages in NLS are logically divided into four 1024-word sections to contain the file blocks.

Thus, up to four file blocks may be in core at a time. When a file block is requested, if all four are in use, one block will be written out. Core blocks may be "frozen" in, however, so that they will not be removed. lc3c1a

A single procedure called LODRFB controls all file input/output (other than file copying). When any routine wants a block loaded, it calls this procedure with the number of the desired block. The block is then loaded and its location in memory returned. lc3c1b

The procedure makes use of several tables. lc3c1b1

One table indicates which file block is in each core block (it is called RFIFCB for "random file index for core blocks"). A zero in this table means that no file block is there, while a positive number is the random file block number (index to RFBS). lc3c1b1a

A second table indicates which of the core blocks have been frozen. "Frozen" indicates to the file block loading procedure that the core block must not be changed. This is the case if some operation, such as editing, is being performed on data within the block. lc3c1b1b

A value in the table of -1 means that the block is not frozen; this value is incremented by 1 for each reason why the block is frozen. lc3c1b1b1

The algorithm of LODRFB is approximately as follows: lc3c1b2

First, a core block is chosen. A quick scan of the first table mentioned above is made to find an unused block. If all are in use, then a counter is used to find the next core block that is not frozen. (If all are frozen the system aborts.) lc3c1b2a

The counter provides a simple algorithm for determining which block should be removed from core. lc3c1b2a1

If the chosen core block contains a file block, then one of the following things happens: lc3c1b2b

(1) If the file block is empty, it is released to the system and the corresponding status block is set to indicate that that block is unallocated. lc3c1b2b1

(2) Otherwise, the block is written out on the file if the checksum has changed, and the random file status block is set to indicate that the block is on the file and not in core. lc3c1b2b2

At this point the desired file block is loaded into the core block. lc3c1b2c

If the random file block has not been initialized, the initialization is done now. Otherwise the checksum and file type are checked. An error is reported if either of these checks fails. lc3c1b2d

Finally, the random file block status is set to show that the block is now in core, and the index for core blocks (RFIFCB) is set to indicate which random file block is in that core block. lc3c1b2e

b. File Copying lc3c2

The algorithm for copying an NLS file is as follows: lc3c2a

First, the procedure must obtain a core block to do the copying. RFIFCB is scanned to find a block that is not used. If there is no unused block, then the first block that is not frozen is taken, and the file block number in it is saved. That block is checksummed and written out on the output file (in the proper file block). lc3c2a1

Having obtained a block, all of the allocated file blocks (except for the one already written in the event that no core blocks were free) are

copied from one file to the other. This includes the file header. 1c3c2a2

Finally, if no blocks were free, the block which was removed to make room for the copy is restored from the output file. 1c3c2a3

c. Referencing Information in the File 1c3c3

As much as possible, information in the file is referenced indirectly through utility functions. This ensures that the file structure can be modified with minimal changes in the system as a whole. 1c3c3a

For each field in the ring element, there are procedures which, given a PSID as argument, either read the contents of the field or store a new value into it. 1c3c3b

Only these procedures need know the actual format of a ring element. Thus only these procedures need be changed if that format is modified. 1c3c3b1

There are also procedures for reading and writing characters in an SDB. This serves both to ensure flexibility in the format of the SDB and to avoid multiple procedures for performing a very common function. 1c3c3c

Because of the lack of instructions for character manipulation on the 940, a rather elaborate method is used to read characters from a statement. 1c3c3c1

Before any characters are read, the procedure FECHC1 is called to initialize a work area. It is called with the address of the work area and the direction in which characters are to be read from the statement. 1c3c3c2

When calling FECHC1, the first two cells of the work area must contain a T-pointer for the first character to be read. A character count of one indicates the first character of the statement. FECHC1 will initialize

the rest of the work area, which contains the following: 1c3c3c2a

WORD 0: PSID 1c3c3c2a1

WORD 1: character count 1c3c3c2a2

WORD 2: return address for routines reading characters 1c3c3c2a3

WORD 3: instruction to branch indirectly through the fourth, fifth, or sixth cells of the work area 1c3c3c2a4

WORDS 4, 5, and 6: address of code to pass the first, second, or third character respectively of the current word of text 1c3c3c2a5

WORD 7: address of the current word of text 1c3c3c2a6

WORDS 8, 9, and 10: the first, second, and third characters in the current word of text 1c3c3c2a7

WORD 11: unused 1c3c3c2a8

WORD 12: the address of the start of the first word of text in the SDB. 1c3c3c2a9

After the work area has been initialized by calling FECHC1, any number of characters may be read from the statement by simply executing a call to the second cell of the work area. After returning the last character of the statement (or first if the direction of readout is backwards), end characters (code 377B) will be returned from all subsequent calls. 1c3c3c2b

The call to the work area places the return location in the second cell and causes the instruction in the third cell to be executed. This results in a branch to a routine which returns the next character. 1c3c3c2c

When all the characters from a particular word have been read, the next word of text is unpacked into the appropriate cells in the work area. 1c3c3c2c1

Whenever a character is read, the branch instruction in the third cell of the work area is modified so that the next call will result in a branch to the appropriate routine to read the next character. 1c3c3c2c2

To change position within the statement, change direction, or read from a different statment, the work area must be reinitialized by calling FECHC1 again, as described above. 1c3c3c2d

Finally, statements may be read in sequence according to view parameters by means of a group of procedures collectively called the "sequence generator." This is described in detail in Sec. IV-B-2 of this appendix. 1c3c3d

It was mentioned above that it would be possible to eliminate the secondary status tables without an undue amount of effort. 1c3c3e

It should be evident now that this is in fact the case as a result of the use of functions to reference information in the file. 1c3c3e1

It would be possible to modify the field sizes in the ring element by simply rewriting the routines that access the affected fields. 1c3c3e2

In addition, a simple process could be written to take files in the current NLS format and convert them to a format using absolute addresses for pointers rather than status tables.

lc3c3e3

III Command Specification 1d

A. Command Specification in NLS 1d1

1. General 1d1a

The command specification section of NLS is implemented in an SPL designed to facilitate its description and implementation. 1d1a1

The details of this language and its use in NLS are explained in the following sections. 1d1a2

2. Registers in the Command Specification Language 1d1b

Two types of registers are used by the command specification machinery: string registers and character registers. 1d1b1

Some of the registers are used internally in the implementation of the language, some are used as special-purpose registers for operations on certain types of operands, and some are general-purpose operand and storage registers. 1d1b1a

Constructs in the input-feedback SPL allow manipulation of the string and character registers. 1d1b1b

The principal defined operations for string registers are LOAD and DISPLAY. 1d1b1b1

The contents of a string register are normally designated in the SPL as the name of the register immediately followed by an asterisk (*). 1d1b1b1a

A register may be assigned a value by a statement of the form 1d1b1b1b

register-name "*" "=" expression. 1d1b1b1b1

Examples of expressions are: 1d1b1b1c

(1) The name of any of the string or character registers 1d1b1b1c1

(2) The designation of a character, such as SP for space 1d1b1b1c2

(3) The character 0, meaning to set the string to null 1d1b1b1c3

(4) A string of text delimited by T-pointers. 1d1b1b1c4

For example, LIT*=0 clears the literal input register, while LIT*=(B1 B2) loads it with the a text string. 1d1b1b1d

The contents of a register may be displayed in the name area by the command of the form 1d1b1b1e

"DN(" register-name "*" ")". 1d1b1b1e1

Thus DN(STN*) causes the contents of the statement name register to be displayed. 1d1b1b1f

The input character register is normally available to the SPL programmer as a read-only register, which always contains the last character read from the input string. 1d1b1b2

The contents of the register may be put into a string as described above, or displayed in the text area by writing DT(C*). 1d1b1b2a

In addition, the input character is implicitly referenced in the case statement (described in Sec. III-A-5 of this appendix). 1d1b1b2b

3. Entity Character and Entity String; Command Groups 1d1c

The commands in NLS are classified in groups, and with each group is associated a particular entity (such as character, word, statement, or branch). 1d1c1

With this entity is associated a character called the "entity character" and a string called the "entity string." 1d1c2

The entity character is programmatically assigned values in the SPL by the construct 1d1c3

"E*=" character "," string. 1d1c3a

This causes the entity character to be set to the value of the character, and assigns the value of the string to the entity string. 1d1c3b

Thus "E*=B,BRANCH" sets the entity character to "B" and the entity string to "BRANCH." 1d1c3c

The entity string and entity character are used to provide a default option in command specification. 1d1c4

When the command operation (such as DELETE) has been specified, the entity string for the group of the operation is offered as the type of entity for the command. The user may accept this by typing a "command accept" character (CA) or specify some other entity by typing the appropriate character. 1d1c4a

The actual SPL constructs used to express this use of the entity string and entity character are presented in a later example. 1d1c5

4. Command State 1d1d

Except when a command is being specified or executed, the user is in some command state. 1d1d1

If the user begins parameter specification without first specifying a new command, the command executed will be that designated by the current command state. 1d1d2

The command state is defined internally by a special register called the "state register." 1d1d3

The state register always contains the location of the most recently defined command state. 1d1d3a

This location is in the same format as a return location placed on the stack in a subroutine call. 1d1d3a1

The state register additionally contains the command group of the command state. 1d1d3b

The SPL syntax for defining a command state is 1d1d3c

"S*=" label "," command-group, 1d1d3c1

which results in a call to the state defining routine to be produced by the compiler. The label is defined as being equal to the address of this instruction. 1d1d3d

From the command state, control passes directly to a parameter specification point in the program, which acts as an idle or "wait for next input" point. 1d1d4

Control returns to the highest level of the command parsing code if the character read is not a legitimate parameter specification character. 1d1d4a

This is one of the most significant features in making the command language efficient and easy to use. 1d1d4b

The contents of the state register may be used as an operand in designational expressions. 1d1d5

Thus, one may programmatically return to the previous command state by the SPL statement "GOTO *[S]*". 1d1d5a

There are several occasions where this construct is used. 1d1d5b

At any time during the command specification, a user may return to his previous command state by typing a "command delete" character (CD). 1d1d5b1

From the above description of command state, it may be seen that the action of a command delete is to reset any parameters entered during the course of the aborted command and branch to the location contained in the state register. 1d1d5b1a

If a specification error occurs during the execution of a command, the command is aborted and NLS is automatically returned to the previous command state. 1d1d5b2

5. Command Parsing 1d1e

The NLS input commands are parsed through the use of nested case statements. 1d1e1

The depth in the nest of case statements corresponds to the position of the next character to be read in the command input string. 1d1e1a

Thus if a command were specified by three characters, the first character would be read by a first-level case statement, the second by a second-level case statement, and the third by a third-level case statement. 1d1e1a1

Two features of the case statement construct in the input-feedback SPL make it especially suited for parsing the command input strings. 1d1e1b

The selection criterion for the execution of an element of the case statement is equality of two specified characters, one of which appears at the front of the element, the other of which is implicit, 1d1e1b1

The implicit character is normally the last character read from the input string. In addition, it is possible to repeat a case (using a "REPEAT" construct) with some character other than the input character. 1d1e1b1a

In particular, the entity character may be used. This permits the implementation of the command default option mentioned above. 1d1e1b1a1

At the head of the case statement, the entity string is used to offer a default value of the command type. If the user types a command accept, there is an element in the case statement which is executed and results in repeating the case statement using the entity character in place of the input character. 1d1e1b1a2

The net effect is the same as if the user

had typed the entity character rather than a command accept. 1d1e1b1a3

If none of the tests succeed, then an "ENDCASE" statement is executed. 1d1e1b1b

Whenever a case statement is executed, an entry is made on a stack indicating the location of that case statement. 1d1e1b2

A construct in the repeat statement allows the execution of a previous case statement with a particular character. 1d1e1b3

The word REPEAT is followed by an integer indicating which of the stacked cases is to be repeated. 1d1e1b4

Thus REPEAT 2 causes the second previous case statement to be repeated. 1d1e1b4a

The integer is in turn followed by a character specification in parentheses. 1d1e1b5

This may be any of the following: 1d1e1b6

(1) An actual character to be used, such as SP 1d1e1b6a

(2) The entity character (E*) 1d1e1b6b

(3) The next input character, indicated by a period. 1d1e1b6c

A brief example of code for parsing an NLS-like command language is presented here. 1d1e2

It incorporates most of the SPL constructs mentioned in this section, as well as some not mentioned. 1d1e2a

The command language described here allows two groups of commands, used for text editing and structure editing respectively. 1d1e2b

Four commands are specified: 1d1e2b1

```
                Text editing: (initial entity = character)  1d1e2b1a

                   Insert Character                         1d1e2b1a1

                   Insert Word                              1d1e2b1a2

                Structure editing: (initial entity =
                statement)                                  1d1e2b1b

                   Append Statement                         1d1e2b1b1

                   Append Branch                            1d1e2b1b2

             (start) . case                                 1d1e2c

                (i) /textedit/ dsp( < insert ↑ es* ) . case 1d1e2c1

                   (c) s*=ic,textedit dsp( ← < insert
                   character)  e*=c,character +parmspec,prmspc
                   -comex,exectr                            1d1e2c1a

                   (w) s*=iw,textedit dsp( ← < insert word)
                   e*=w,word +parmspec,prmspc -comex,exectr  1d1e2c1b

                   (ca) repeat 0(e*)                        1d1e2c1c

                   (cd) goto /s/                            1d1e2c1d

                   endcase goto start                       1d1e2c1e

                (a) /stredit/ dsp( < append ↑ es* ) . case  1d1e2c2

                   (s) s*=ic,stredit dsp( ← < append statement)
                   e*=s,statement +parmspec,prmspc
                   -comex,exectr                            1d1e2c2a

                   (w) s*=iw,stredit dsp( ← < append word)
                   e*=w,word +parmspec,prmspc -comex,exectr  1d1e2c2b

                   (ca) repeat 0(e*)                        1d1e2c2c

                   (cd) goto /s/                            1d1e2c2d

                   endcase goto start                       1d1e2c2e

                endcase repeat 0(.)                         1d1e2c3
```

6. Parameter Specification 1d1f

Parameter specification is that portion of NLS which is involved with the selection of operands for commands. 1d1f1

Operands may be specified by selecting locations and entities in a file, by entry of strings from the keyboard, or by the naming of pointers with the keyset and mouse. 1d1f2

Specifications of entities in the file are represented by one or more entries on a stack, called the specification stack. (This is independent of the subroutine argument and return stack.) 1d1f3

There is one entry on the specification stack for each selection made in parameter specification. 1d1f3a

A normal entry on the specification stack (spec stack for short) is called T-pointer (which consists of a PSID and a character count). 1d1f3a1

An SPL construct facilitates the placing of arguments onto the spec stack. The syntax is 1d1f3a2

"SPEC(" argument ")", 1d1f3a2a

where an argument can be any of the following: 1d1f3a3

BUG: Process the most recent command accept as a bug selection and place the corresponding T-pointer on the spec stack 1d1f3a3a

POS: Load the last bug selection onto the spec stack. 1d1f3a3b

String register: The action of this command depends on the register specified, and the contents of the register. 1d1f3a3c

If the register is the number register, then the number string in the register is converted to an integer and pushed onto the spec stack as the second word 1d1f3a3c1

If the specified register is the

statement number register, it converts the string in the register (assumed to be a statement number) into a PSID, and pushes it onto the spec stack ldlf3a3c2

In the case of any other register, if the first character in the string is a digit, then the content of the register is assumed to be a statement number, otherwise, a statement name. In either case the corresponding PSID is pushed onto the stack. ldlf3a3c3

Number: The integer indicated is pushed onto the spec stack ldlf3a3d

Identifier: The value of the identifier is pushed onto the spec stack ldlf3a3e

(no argument): This causes the spec stack to be cleared of all entries. ldlf3a3f

A textual entity may be specified (effectively) only through bug selection(s) or with a pointer. ldlf4

A structural entity may be specified by bug selection(s), a pointer, or keyboard entry of statement name(s) or number(s). ldlf5

In the case where the bug selection or pointer serves as a text selection which indicates a string identifying the statement to be specified (e.g., names, links), the selected string is moved into a string register and treated as though it were entered from the keyboard. ldlf5a

The algorithms for converting bug selections into T-pointers are discussed in Sec. IV-B-6-c of this appendix. ldlf6

A pointer is simply a T-pointer which has been given a name and stored in a table. ldlf7

It is specified by depressing the right button on the mouse, and entering the name of the pointer with the keyset. ldlf7a

When a pointer has been specified, the associated T-pointer is simply loaded into the internal register containing the (processed) mouse location, making it appear as though a bug selection had been made. ld1f7b

A statement may be selected from the keyboard by typing either the statement name or the statement number. ld1f8

A statement number is converted into a PSID for a T-pointer by simply running through the ring at each level (beginning with level 1) until the specified statement is reached, or found to be non-existent. ld1f8a

A statement name is converted into a T-pointer by running through the ring, looking for a statemnt which has a name, and whose hash is the same as the hash of the name being searched for. ld1f8b

In the case where an operand is a textual entity which is entered from the keyboard, there need not be an entry on the specification stack for it. ld1f9

Rather, it will go directly into a specified register, and be used in that form for the command. ld1f9a

It should be noted that the selections of textual entities in the file are processed during execution of the command so that (when appropriate) the textual entity is put into a register in the same form it would be in if it had been entered from the keyboard. ld1f9b

7. Subroutine Calls and Parameter Passing ld1g

The subroutine call mechanism in the SPL is very similar to that used by ALGOL. It uses a stack for containing return information, parameters, and local variables. ld1g1

Because of the overlay structure of NLS, it is necessary to indicate in a subroutine call not only the address of the routine being called, but

additionally the name of the overlay in which that routine resides. ld1g1a

The name of the overlay containing the calling routine is stacked with the return location, so that the appropriate overlay may be relabeled in upon return. ld1g1a1

There are two types of subroutine calls, which differ in the return locations placed on the stack. ld1g1b

The return location stacked by a normal subroutine call is the address of the location following the calling instruction. ld1g1b1

The other subroutine call stacks the return location of code which will return NLS to the previous command state. ld1g1b2

The format and operation of the stack (and subroutine call mechanism) are roughly as follows: ld1g1c

The stack is addressed by two pointers, one to the current base and one to the stack top. ld1g1c1

A subroutine call instruction is always preceded by a "mark stack" instruction. ld1g1c2

The "mark stack" instruction pushes the contents of the base-of-stack pointer onto the top of the stack, followed by a zero (which will be used by the actual subroutine call for the return location). ld1g1c2a

The top-of-stack pointer is incremented accordingly, and the base-of-stack pointer is set to point to the new top of the stack (which will eventually contain the return location). ld1g1c2b

Formal parameters are now loaded onto the top of the stack. ld1g1c3

If an overlay has been specified in the subroutine call syntax, a cell is set to

reflect the overlay containing the procedure being called. ld1g1c4

Note that the actual program relabeling is not changed at this time. ld1g1c4a

The subroutine call is now executed. ld1g1c5

The return location is computed. ld1g1c5a

This is a combination of the calling address and the name of the overlay containing the subroutine call instruction. ld1g1c5a1

This is true except in the case of the special subroutine call which returns to the previous command state. ld1g1c5a1a

In the special subroutine call, the contents of the state variable (which in fact is the return location for the previous state, as computed above) are used as a return location. ld1g1c5a1b

The return location is stored in the cell pointed to by the base-of-stack pointer. ld1g1c5a2

Finally, the overlay containing the called procedure is relabeled in if necessary, and a branch is made to the address indicated in the subroutine call. ld1g1c5a3

The syntax of a subroutine call in the SPL is ld1g2

("+" / "-") procedure-name ("," overlay-name / EMPTY), ld1g2a

where " / EMPTY" means the construct before the slash is optional. ld1g3

In addition, parameters may be specified by listing them in square brackets after the call. Individual parameters in the parameter list are separated by commas. ld1g4

The "+" indicates a normal subroutine call, and a "-"

indicates a special subroutine call which returns to the previous command state. 1d1g5

If no overlay name is specified, an overlay which is either the overlay containing the calling procedure or an overlay above it in the overlay tree is assumed, and thus no change is made in the relabeling. 1d1g6

An example of a subroutine call is 1d1g7

```
+subpat  +wdr2,txtedt/bl,pl-4/  -qdv,txtedt.
```
1d1g7a

8. Input Machinery 1d1h

a. Work Station Input from Keyboard, Keyset, and Mouse 1d1h1

Characters are read from the work station by a system routine in the following manner: 1d1h1a

Whenever a button on the keyboard, keyset, or mouse changes state, the TSS I/O software considers it a character entry, and places the following information into its input buffer. 1d1h1a1

(1) The device which caused input 1d1h1a1a

(2) A code which is the input itself: 1d1h1a1b

(a) A character in the case of the keyboard 1d1h1a1b1

(b) A code in the case of the keyset 1d1h1a1b2

(c) A down/up and button indication in the case of the mouse 1d1h1a1b3

(3) The mouse coordinates at the time the character was read 1d1h1a1c

(4) The time (16 millisecond resolution) when the character was read. 1d1h1a1d

A system call is then used by NLS for reading the characters from the system input buffer, which returns a character (and related information as

described above) if there is one, and reports the status of the system input buffer (empty, another character waiting in input buffer, no character read). 1d1h1b

b. Input Fork 1d1h2

Because of the necessity to read characters from the system input buffer so that it does not overflow -- and more important, to provide a facility to interrupt NLS while it is executing a long process -- a fork is activated to run asynchronously in parallel with NLS. 1d1h2a

This fork may be conceptualized as an independent program (called the input fork) which reads characters from the work station and places them in a programmatic input buffer to be read later by NLS. 1d1h2b

NLS always reads characters from the programmatic input buffer before reading them from the system, and when it is reading a character from the system, it checks to ascertain that the input fork is not reading the same character. 1d1h2b1

The input fork additionally has the capability to interrupt NLS from the process it is currently involved in, and it does so when it reads an interrupt character (RUBOUT) from the keyboard. 1d1h2c

Since NLS always reads characters passed to it from the input fork before reading those waiting in the system, and there is no restriction on where the input fork gets the characters it will pass to NLS, the input fork may be used to simulate an NLS user. 1d1h2d

A simple facility is currently provided along this line, whereby the input fork can read characters from a file, and (with a minimum of translation and interpretation) pass them on to NLS. 1d1h2d1

This feature is used mostly for merging and

converting sequential files into NLS files. 1d1h2d1a

c. Character Translation 1d1h3

The keyset and mouse input requires translation from its raw input form to a character which is meaningful to NLS. 1d1h3a

The keyset input is in the form of a number (0-31) which reflects the keys depressed (and released) on the keyset. 1d1h3a1

This is combined with the current state of the left and middle mouse buttons (which provide a case shift) to produce the translated character. 1d1h3a2

The translation algorithm is roughly as follows: 1d1h3a3

If both mouse buttons are down (case 3) then this is a view specification character, so treat specially. 1d1h3a3a

Otherwise, use the keyset character as an index into a table of character values. 1d1h3a3b

This table of character values has three entries for each possible keyset value, one for each of the remaining cases. 1d1h3a3b1

The case is then used to determine the correct table entry as the translated character. 1d1h3a3b2

Additional translation is done when characters are entered from the mouse without concurrent entry from the keyboard or keyset. 1d1h3b

This translation simply looks for combinations of up/down strokes of mouse buttons without intervening characters, and translates them to specific characters. 1d1h3b1

This is used for the command accept, command delete, backspace character, and backspace word characters. 1d1h3b2

9. Output (Display) Machinery 1d1i

a. General 1d1i1

NLS communicates with the user via a display screen divided into six areas. 1d1i1a

Each area is maintained separately of the others, and contains a specific type of information. 1d1i1b

The organization of the registers on the display screen, and the format of the registers themselves, are parameterized. 1d1i1c

There are many parameters which relate specifically to certain registers, and some parameters which relate to all registers. Among the parameters relevant to all of the registers are: 1d1i1c1

location on screen 1d1i1c1a

character size and type used in register 1d1i1c1b

display of register on/off 1d1i1c1c

Insofar as possible, these parameters are the display control words used by the hardware. This minimizes the software required for controlling the screen format. 1d1i1c2

b. View Areas 1d1i2

(1) Echo Register 1d1i2a

The echo register is maintained by the system and reflects the raw character input to NLS. 1d1i2a1

NLS is concerned with this register mainly at initialization, when it must be set up by a series of system calls. 1d1i2a2

(2) VIEWSPEC Area 1d1i2b

The view specification (VIEWSPEC) area reflects those text area view parameters which are not obvious from looking at the text area. 1d1i2b1

The VIEWSPEC area is changed by the same routine which changes the view parameters themselves. 1dli2b2

(3) Command Feedback Line 1dli2c

The command feedback line is the major feedback mechanism of the command specification machine. 1dli2c1

There are two components in the command feedback line: words which reflect in English the command being specified, and an arrow which indicates the user's state in specifying the command (the arrow most commonly indicates whether the user may specify a new command or parameters, or whether he is currently specifying an entity). 1dli2c2

There are three possible positions to which a word may be moved in the command feedback line: 1dli2c3

First position: This causes the command feedback line to be cleared, and the designated word to be displayed as the first word in the line. 1dli2c3a

Next position: This appends the designated word to the end of the command feedback line. 1dli2c3b

Last position: This replaces the last word in the command feedback line with the designated word. 1dli2c3c

The arrow may be pointed to the beginning of the word in a specified position in the command feedback line, or it may be turned off. 1dli2c4

The SPL construct provided for the manipulation of the command feedack line is 1dli2c5

"DSP(" display-parts ")", 1dli2c5a

where the syntax of a display-part is 1dli2c6

word / "ES*" / "<" word / "..." word / "←" / "↑". 1dli2c6a

The DSP command rearranges the command feedback line so that it is formatted in accordance with the display-parts. ldli2c7

The meanings of the display parts are as follows: ldli2c8

Word: A string equal to the text of the the word is placed in the indicated position in the command feedback line ldli2c8a

"ES*": The contents of the entity string are displayed in the indicated position in the command feedback line ldli2c8b

"<" word: The word is placed at the left of the command feedback line ldli2c8c

"..." word: Replace the last string in the current command feedback line with the word ldli2c8d

"←" : Position the up-arrow to the front of the command feedback line. ldli2c8e

"↑" : Position the up-arrow at the start of the following string in the command feedback line. ldli2c8f

There are three additional intrinsic functions which are used in relation to the command feedback line. These are ldli2c9

AF Turn off display of arrow ldli2c9a

AN Turn on the display of the arrow ldli2c9b

QM Display question mark beside the arrow. ldli2c9c

(4) Name Register ldli2d

The name register is used for displaying statement names and arbitrary strings relating to parameter specification. ldli2d1

An SPL function is provided which moves the contents of an arbitrary string register to the

name register. The syntax is "DN(" register ")". 1d1i2d2

(5) Date/Time Register 1d1i2e

The date/time register always reflects the date and time. 1d1i2e1

It is updated every 10 seconds by a fork (similar to the input fork in its relation with NLS) whose sole job is to read the date and time from the system, place it in a core location, and dismiss itself for 10 seconds. 1d1i2e2

(6) Text Area 1d1i2f

The text area serves as the user's window into his file. 1d1i2f1

What is displayed in the text area is a view of the user's file, subject to certain formats and reorganization, which is described by a set of parameters (called view specifications or VIEWSPECs). 1d1i2f1a

The creation of new views is programmatically caused by the display SPL construct "DISPLAY(" optional-parameter ")". 1d1i2f2

If there is a parameter, it is used to determine the PSID of the starting statement for the view creation. 1d1i2f2a

The process of creating a view of the file in the text area is discussed in Sec. IV-B-6 of this appendix. 1d1i2f3

c. Literal Feedback 1d1i3

When a literal string is entered as a part of parameter specification, it is placed in the text area (beginning at the top) according to the format of the text area. 1d1i3a

The part of the file view which was previously in the space used by the literal feedback is temporarily replaced by the feedback. 1d1i3b

B. Command Specification in TODAS 1d2

The TODAS command specification system is much simpler than that of NLS, insofar as it does not use the state machine and no command state is defined other than the null command RESET. 1d2a

1. Command Feedback 1d2b

The command language input string is parsed by case statements in a manner similar to NLS. 1d2b1

The command feedback may best be described as complex character echoing, where each command specification character is reflected by the typing of appropriate words and the state of the command specification is indicated by the position of the carriage. 1d2b2

As in NLS, the user has the ability to control parameters relating to the command feedback, including the number of characters of each word echoed. 1d2b3

2. Input Machinery 1d2c

Much of the NLS input machinery is used by TODAS. 1d2c1

There are, however, some differences: 1d2c2

Because of the allowance which the system makes for an interrupt character (RUBOUT), and the fact that the system teletype buffers are larger than the system work station buffers, an input fork is not required. 1d2c2a

One may still be used, however, in special cases such as sequential file input. 1d2c2a1

All characters read by TODAS undergo a translation on input. 1d2c2b

This facilitates the effective interfacing of TODAS to a number of input devices (six different types of typewriter terminals are currently provided for). 1d2c2b1

The character translation is accomplished by

a table look-up technique (the table is indexed by the raw character value). ld2c2b1a

The result of the look-up may be a normal text character, or it may be a special character (which is indicated by the high-order bit). ld2c2b1b

In the event that it is a special character (command accept, command delete, shift character, centerdot, etc.), an appropriate action is taken if necessary. The character may be echoed (as some previously designated character), and it may be specially flagged as a control character. ld2c2b1c

There is, in addition to straight character translation, a facility to define shift characters which allow devices with restricted character sets (e.g. upper case only) to work with full character sets. ld2c2b2

Four shift modes are currently defined in TODAS: ld2c2b2a

Null: No shifting takes place ld2c2b2a1

Mode 0: Upper-case alphabetic characters are translated to lower case ld2c2b2a2

Mode 1: Lower-case alphabetic characters are translated to upper case ld2c2b2a3

Mode 2: Lower- and upper-case alphabetic characters are translated to control case ld2c2b2a4

TODAS is in one of these modes (as a base mode) at all times. ld2c2b2b

The mode may be changed (either temporarily or permanently) by typing a character which has been defined as a shift character for the new mode. ld2c2b2c

There are currently three types of mode-shifting characters: ld2c2b2c1

Character shift: This causes the following character to be translated according to the mode for which the shift character has been defined, if it is a character which would normally have been translated in either the base mode or the shift mode. If the character would not have been translated, then the shift character is treated as a normal character. ld2c2b2c1a

Word shift: This causes the following word to be translated subject to the same rule as given above for character shift -- i.e., if the next character is translatable, the word is traslated; otherwise th shift character is treated as a normal character. ld2c2b2c1b

Permanent shift: This causes the base mode to be changed, and all subsequent characters are translated according to the new mode. ld2c2b2c1c

The shifting is accomplished in the following manner: ld2c2b2d

If, a permanent shift character is read at any time, the shift mode is changed and another character is read normally. ld2c2b2d1

If a word-shift or character-shift character is read, the next character is read from the input string. ld2c2b2d2

If the next character is a shiftable character, then the shifting is performed, and the shifted character is the result. ld2c2b2d2a

If the shift character is for a word shift, then a global parameter indicating the current shift state is set accordingly, and will not be reset until a space is read. ld2c2b2d2a1

If the next character is not a shift character, it is returned to the front of the input string and the shift character is returned as a normal character. 1d2c2b2d2b

3. Printing 1d2d

Printing of a structure in TODAS is analogous to creating a new view for the text area in NLS, insofar as the same view specifications are used for interpreting and formatting the file. 1d2d1

Three differences are apparent: 1d2d1a

The text area is of unlimited length, so that a whole file may be seen in one view. Pagination is performed when a long view is created. 1d2d1a1

Text undergoes an output translation and shifting which is a counterpart of the translation and shifting done on input. 1d2d1a2

The user has a degree of interactive control over the view being created, specifically: 1d2d1a3

The creation of a view of any particular statement may be aborted at any time. 1d2d1a3a

The creation of the entire view may be aborted at any time. 1d2d1a3b

Implementationally, formatting routines different from those used by NLS are employed. 1d2d1b

The output is formatted one line at a time, and the printing of an entire statement must physically finish before the first line of the next statement will be printed. 1d2d1b1

This restriction is necessary because TODAS must know which statement is currently being typed in order to respond properly to the user's request to abort the view of the statement. 1d2d1b1a

The same sequence generator is used, but the

structure being printed is searched one branch at a time (except in the case of trails and keyword). 1d2d1b2

4. Parameter Specification 1d2e

Parameter specification differs from NLS in three important ways: 1d2e1

All specification must be done via the keyboard. 1d2e1a

A "current statement" is defined as an operand at all times. 1d2e1b

The execution of any command without a specified operand assumes this statement as an operand. 1d2e1b1

The current statement is represented internally as a cell containing the PSID of the last statement addressed in the successful execution of a command. It is updated each time a command is successfully executed. 1d2e1b2

The one exception to this is that during printing, it is set by the print routines to the PSID of the last statement printed. 1d2e1b2a

Operands (statements) may be addressed relative to each other in the tree structure of the file. 1d2e1c

For example, one may specify a statement which is the "successor of the down of the tail" of the current statement -- i.e., the successor of the first substatement of the last statement in the same plex at the same level as the current statement. 1d2e1c1

The relative addresses of operands are interpreted as they are entered by accessing the ring (as necessary). Any error is reported immediately, and nullifies the entire address (except in the case of links). 1d2e1c2

Links are parsed whenever they are referenced in an address field, and executed immediately after selection. That is to

say, when a link is encountered in an address field, the current statement is changed immediately to reflect the value indicated by the link.

1d2e1c2a

IV Command Algorithms 1e

A. Editing 1e1

Editing in NLS includes textual, structural, and graphical modifications to the file. 1e1a

The textual and structural editing actions include insert, move, replace, delete, and copy. These actions may be performed on textual entities such as characters, words, and visible strings, as well as structural entities such as statements, branches, groups, and plexes. 1e1a1

The graphical editing actions include insert and delete for vector labels, and insert, delete, move, transpose, and vertical and horizontal projection for vectors. 1e1a2

1. Text Editing 1e1b

a. General Considerations 1e1b1

The process of textual editing will be discussed first. This process basically consists of delimiting the appropriate substrings, by means of the content-analysis SPL, followed by construction of one or more new statements with the desired modifications. This latter step is specified by a procedure written in another SPL, the string-construction SPL. 1e1b1a

These content-analysis and string-construction procedures are written in such a way that in spite of the large number of combinations of editing actions and textual entities, there is a single content-analysis procedure to delimit each entity and a single string-construction routine to perform each action. 1e1b1b

This is done by standardizing the way in which a substring is delimited by the content-analysis procedures. 1e1b1c

Four pointers are passed to the procedure as

arguments, along with one or two selections made by the user. 1e1b1c1

When the procedure returns, the appropriate substring is delimited by the pointers in the following manner. 1e1b1c2

The first and second pointers mark the first and last characters of the substring, respectively. The third and fourth pointers mark the characters to the left and right of the substring, respectively. 1e1b1c2a

Thus if P1, P2, P3, and P4 are the arguments, the characters from the front of the statement up to P3 precede the desired substring, the characters from P1 to P2 are the substring, and those from P4 to the end of the statement follow the substring. 1e1b1c2b

A detailed description of the word-delimiter routine is useful to clarify this process. 1e1b1d

There are five arguments; the first is the position of the user's selection, the remaining are pointers to be used to delimit the actual text of the word in the manner described above. The body of the procedure is simply 1e1b1d1

```
a1 > CH $LD ↑a3 ↑a5 ←a3 a1 < CH $LD ↑a2 ↑a4
←a2
```
1e1b1d1a

which has the meaning "starting from the selection (a1) scan to the right (>) past a character (CH) and any number of letters or digits ($LD). Set a3 and a5 to the resulting position (↑a3 ↑a5) then move a3 back (←a3) so that it points to the last character of the word. Now reset the search pointer to the selection (a1) and scan to the left (<) to set a2 and a4 (↑a2 ↑a4 ←a2)." 1e1b1d2

Once the substrings have been delimited in the above manner, new statements are constructed under the control of procedures written in the string-construction SPL. 1e1b1e

The syntax of a statement in the string-construction SPL is as follows: 1e1b1f

```
scstat = "IF" posrelation "THEN" scstat "ELSE"
scstat  /                                          1e1b1f1

   "BEGIN" scstat $(";" scstat) "END" /            1e1b1f1a

   "ST" pos  "←" pairlist;                         1e1b1f1b
```

The position and position-relation constructs are the same as in the content-analysis SPL. 1e1b1g

A pairlist is a list of pairs, in this case separated by commas. 1e1b1h

A "pair" specifies a string of text, usually by giving two positions which delimit the string. 1e1b1h1

In addition the "pair" can be a constant string or the contents of some variable string such as the literal input register. 1e1b1h2

The meaning of "ST pos ← pairlist" is "The statement pointed to by pos is constructed from the strings specified by the items in the pairlist." 1e1b1i

Thus, assuming that the pointers have been set as described above, "ST B1 ← SF(B1) P3, P4 SE(B1)" would cause the text from P1 to P2 to be deleted from the statement selected by B1. 1e1b1i1

The "move" procedure offers a more complex example. The procedure has ten arguments; a1 and a2 are the user's selections, a3 through a6 are the pointers associated with a1, and a7 through a10 are the pointers for a2. The body of the move routine is 1e1b1j

```
IF SF(a1) = SF(a2) THEN BEGIN                      1e1b1j1
   IF a1 < a2 THEN                                 1e1b1j1a
      ST a1 ← SF(a1) a4, a7 a8, a6 a9, a10
      SE(a1)                                       1e1b1j1a1
   ELSE                                            1e1b1j1b
      ST a1 ← SF(a1) a9, a10 a4, a7 a8, a6
      SE(a1) END                                   1e1b1j1b1
```

```
ELSE BEGIN                                                    1e1b1j2
   ST a1 ← SF(a1) a4, a7 a8, a6 SE(a1);                       1e1b1j2a
   ST a2 ← SF(a2) a9, a10 SE(a2) END                          1e1b1j2b
```

The pair a7 a8 delimits the text to be moved. The positions a9 and a10 will become adjacent when the text from a7 to a8 is moved. The destination of the text between a7 and a8 is after a4 and before a6. The reader should convince himself that the above procedure does this in all cases. 1e1b1k

b. Implementation 1e1b2

The code compiled for string-construction SPL routines consists mainly of calls to MOL procedures. 1e1b2a

At the start of the code for a pairlist there is a call to a procedure called BSC (begin string construction) and at the end of the pair list there is a call to ESC (end string construction). For the actual items in the pairlist, procedures are called which append the appropriate strings onto the statement being constructed. 1e1b2b

The BSC procedure must create a new statement data block (SDB) to hold the text of the statement being constructed. Since the final size of the statement is not known at the time BSC is called, the average size of SDBs in the file is used as an estimate of the number of words required for the new SDB. 1e1b2c

The search for the required amount of room begins in the file block containing the old SDB, if there was one. 1e1b2c1

If there is not adequate room there, then the procedure looks for room in the file blocks, starting with the lowest index number. 1e1b2c2

This ensures that if there is room in a block already allocated, then that room will be used rather than causing a new block to be allocated. 1e1b2c2a

The procedure ISROOM is called to determine

whether there is adequate room in a given file block. 1e1b2c3

If the block is unallocated, then ISROOM returns TRUE. 1e1b2c3a

If the block is allocated and contains adequate free storage, then such information is held in the status table, RFBS. This avoids the possibility of reading a file block only to find that it does not contain adequate room. 1e1b2c3b

If the block does not contain adequate free storage, but does contain garbage SDBs (also known from RFBS), then ISROOM calls the garbage collector to process the block. 1e1b2c3c

Garbage collection involves moving nongarbage SDBs to fill in the gaps occupied by garbage SDBs and updating pointers in the ring elements corresponding to the moved SDBs. 1e1b2c3c1

If this produces enough room, then ISROOM returns TRUE; otherwise it returns FALSE. 1e1b2c3d

After sufficient room has been found by the above process, the BSC procedure builds a header for the new SDB and then sets up a work area for the subsequent string transfers that will take place during the construction of the statement. This work area contains information such as the address of the SDB. This completes the tasks of BSC, and it returns. 1e1b2c4

The actual construction of the new statement consists of appending characters onto the new SDB. 1e1b2d

For those parts of the statement that remain the same, the text is read out of the old SDB into the new. New parts of the statement are simply characters from other sources, such as literal input or other SDBs. 1e1b2d1

The observant reader will realize that it is

possible to run out of room while appending characters. 1e1b2d2

If this happens, the block is garbage-collected. If this results in room for at least 60 more characters, then the SDB under construction is simply moved in with the same file block to make more room. 1e1b2d2a

If garbage collection of the file block cannot produce that much more room, a location in a different file block is found that does provide the required space. The partially constructed SDB is then moved to this new location. 1e1b2d2b

When all the strings have been appended to the SDB, the procedure ESC is called to finish the job. 1e1b2e

It first gets rid of the old SDB for the statement, then does the bookkeeping to establish the new SDB as the SDB for the statement. This involves updating the SDB header, the running average length of SDB's, the pointer in the statement's ring element, and the name hash for the statement in the ring element. 1e1b2e1

In addition the "content analyzer pattern tested" flag for the statement is turned off (see Sec. II-B-2-c of this appendix). 1e1b2e2

This completes the construction of a new statement and our discussion of text editing in NLS. 1e1b2f

c. Content-Analysis SPL 1e1b3

In NLS it is often necessary to analyze the textual content of a statement in order to delimit certain substrings. 1e1b3a

For example, the user may select a word of text for editing by pointing to any character within the word. The actual substring making up the word is determined by NLS. 1e1b3a1

A special language, the content-analysis SPL, is used for writing such string delimiting procedures. 1e1b3b

Basically, the language provides constructs for controlling the position of a search pointer in a text string and saving various positions in order to delimit the desired substrings. (In the discussion of the content analysis SPL, position refers to a statement identifier and character number -- in other words, a T-pointer as defined elsewhere.) 1e1b3c

The initial position of the search pointer is often determined by a selection made by the user. The positions of such selections are stored in buffers B1, B2, etc. 1e1b3d

Pointers P1, P2, ... may be used to store positions. The current position of the search pointer can be stored in Pn by writing ↑Pn. 1e1b3e

Arguments may be passed to a content analysis procedure. Such arguments are either bug selections (i.e. Bn) or pointers (i.e. Pn). Since the procedure must be able to set the pointers to appropriate values, these parameters are called by (simple) name rather than by value. The formal parameters are A1, A2, etc. 1e1b3f

The three forms, Bn, Pn, and An, are the basic ways of referencing a position. In addition, there are two functions taking a position as argument and yielding a position as result. These are SF and SE, which give the position of the statement front and statement end, respectively, of their argument. 1e1b3g

The position of the search pointer can be set by simply writing any of the above forms to determine a position. For example, "SF(B1)" puts the search pointer at the first character in the statement first selected by the user. 1e1b3h

The search pointer is also moved by tests for basic text elements. The basic text elements are

strings, single characters, and character class variables. 1e1b3i

A string is a sequence of characters delimited by quote marks ("). 1e1b3i1

If the string matches the sequence of characters starting at the current location of the search pointer, then the search pointer is moved to the next position beyond the string and a general flag is set TRUE. 1e1b3i1a

If, on the other hand, there is only a partial match, or no match, then the search pointer is not moved and the general flag is set FALSE. 1e1b3i1b

The test for a single character is logically equivalent to testing for a string of length one, but is implemented in a more efficient manner. The single character is specified by preceding it with an apostrophe. 1e1b3i2

The implementation of these tests makes use of the programmed operator (POP) facility of the 940. 1e1b3i3

For the single character test, the computer produces a single instruction in which the address field contains the code for the character and the rest of the instruction specifies the POP to perform the test. 1e1b3i3a

Similarly, the string test results in an instruction specifying the number of characters in the string and the appropriate POP, followed by words containing the actual string. 1e1b3i3b

The basic text elements of the third type -- the character class variables -- are also implemented using a programmed operator. The character class variables allow tests for any character in a particular class. The classes, with their associated variable names, are as follows: 1e1b3i4

LD	any letter or digit	1e1b3i4a
L	any letter	1e1b3i4b
D	any digit	1e1b3i4c
NP	any nonprinting character	1e1b3i4d
PT	any printing character	1e1b3i4e
SP	space	1e1b3i4f
TAB	tab	1e1b3i4g
CR	carriage return	1e1b3i4h
CH	any character	1e1b3i4i

These tests are implemented in a manner very similar to the single character test, except the address field of the instruction contains a class code rather than a character code. 1e1b3i5

The successful completion of one of the above tests causes the search pointer to be moved. The direction in which it is moved, towards the end of the statement or the front, may also be controlled. 1e1b3j

A ">" means scan (move pointer) to the right, or towards the end, while "<" means scan left. 1e1b3j1

As mentioned above, the current position of the search pointer can be saved by writing "↑" followed by either Pn or An. 1e1b3k

In addition the value stored in a buffer can be modified to point to the preceding character, according to the current scan direction, by writing "←" followed by Pn or An. 1e1b3l

The reason for this operation is that when an entity has been successfully found the pointer is left pointing to the character beyond the entity. Thus to save the position of the last character in the entity it is necessary to write ↑Pn←Pn. 1e1b3l1

The remainder of the language simply provides for building more complex expressions from the basic text elements presented above. 1e1b3m

One of the primary means of doing this is the arbitrary number operation. The general form of this is m$n followed by a text expression and has the meaning "from m to n occurrences of the given expression." 1e1b3m1

Both the upper and lower bounds are optional, with default values of 1000 and 0 respectively. 1e1b3m1a

This is implemented in the following manner. 1e1b3m1b

The upper and lower bounds and a count, initially zero, are pushed on the stack. Then the test for the expression is repeated until it fails, with the count being incremented at the completion of each successful test. 1e1b3m1b1

When the test for the expression does fail, the current value of the count is checked against the bounds and the general flag set accordingly. 1e1b3m1b2

The other operators, in order of decreasing precedence, are as follows: 1e1b3m2

- (minus sign): indicates negation. 1e1b3m2a

After the test for the text expression following the minus sign, the value of the general flag is complemented. 1e1b3m2a1

(space): indicates concatenation. 1e1b3m2b

After the test for each element in a sequence of concatenated tests, the general flag is tested. If it is false, then the preceding element was not found and control branches to the location following the current sequence of concatenations. If the flag is true, then the next test in the sequence is

performed. 1e1b3m2b1

/ (slash): indicates alternatives. 1e1b3m2c

If the expression on the left of the slash is found, then control branches beyond the sequence of alternatives. Otherwise, the search pointer is reset to its position prior to the test for the previous alternative and the next alternative in the sequence is tested. 1e1b3m2c1

NOT: indicates negation. 1e1b3m2d

Equivalent to minus sign except for lower precedence. 1e1b3m2d1

AND: indicates logical conjunction. 1e1b3m2e

If the expression on the left of the AND is not found, then control branches beyond the expression on the right of the AND. Otherwise, the search pointer is reset to its position prior to test for the left expression and then the right expression is tested. 1e1b3m2e1

OR: indicates logical disjunction. 1e1b3m2f

Like AND except branch if flag true instead of false. 1e1b3m2f1

Any expression built using the above operations may be enclosed in parentheses and used as a basic element in a concatenation. 1e1b3m3

Similarly, any such expression may be enclosed in square brackets and used as a basic element. The effect of the square brackets is to "unanchor" the scan. In other words, as long as the test fails, it is repeated starting one character farther along in the statement until either the statement is exhausted or the test succeeds. 1e1b3m4

Thus *["abc"]* is satisfied if the remainder

of the statement contains the string "abc". 1e1b3m4a

Finally, a conditional statement is included in the language to allow a pattern to be selected for testing on the basis of a comparison of positions. 1e1b3m5

If two positions are in different statements, then all relations between them are false except "not equal." Otherwise, the relationship depends on the character number of the position. For example, if B1 and B2 are in the same statement, B1 pointing to character number 3 and B2 to character number 20, then B1 is less than B2. 1e1b3m5a

This completes the description of the content-analysis SPL. 1e1b3n

2. Structure Editing 1e1c

Like text editing, structure editing consists of a phase in which the entity to be edited is delimited, followed by the actual editing action. 1e1c1

Since the structural entities "branch" and "plex" are simply special cases of the group entity, the editing routines all deal with either a single statement or a group. 1e1c2

The delimiting for the move and delete commands is the same. 1e1c3

In all cases a group, specified by two PSID's, is the final entity on which the editing action is performed. 1e1c3a

For a branch the two PSID's for the group are set to the PSID of the selected statement. 1e1c3a1

For a plex the PSID's are set to the head and tail of the plex of the selected statement. 1e1c3a2

For a statement, a test is made to ensure that the statement has no substructure, after which it is treated like a branch. (If the statement

does have substructure the command is aborted.) 1e1c3a3

Finally, if the specified entity is a group, then the two selected statements are checked to verify that they do in fact specify a valid group. 1e1c3a4

Once the group has been delimited, the move commands perform the following sequence of operations. 1e1c4

First, the destination is checked to make sure it is not within the specified group. The command is aborted if it is. 1e1c4a

The group is then removed from the ring structure by the appropriate changes in pointers and flags in the ring element of the predecessor (and possibly the successor) of the group. The group is then reinserted into the ring in its new location through another set of changes in pointers and flags. Notice that no text is moved and no statement identifiers are changed. The only changes are in the successor and substatement fields and the head and tail flags of four or five ring elements. 1e1c4b

The execution of delete commands naturally results in greater changes. The group is first removed as in the move operation. Then the statements making up to the group are deleted according to the following algorithm expressed in MOL. 1e1c5

```
d1←grp1; %start with the first statement in the
group%                                              1e1c5a
LOOP BEGIN                                          1e1c5b
   WHILE (d2 ← getsub(d1)) NOT= d1 DO BEGIN         1e1c5b1
      %d1 has substructure%                         1e1c5b1a
      stosub(d1,d1); %change sub-pointer            1e1c5b1b
         so that d1 no longer appears to have
         substructure%                              1e1c5b1b1
      d1 ← d2 %more to sub% END;                    1e1c5b1c
   %when exit the WHILE statement,                  1e1c5b2
      d2 equals d1 and has no substructure %        1e1c5b2a
   d1 ← getsuc(d1);  %move d1 to the successor,     1e1c5b3
      which will be back to the "father" statement
                                                    1e1c5b3a
```

```
        when all of its descendents have been
        deleted%                                        1e1c5b3b
    relst(d2);  % release SDB for d2%                   1e1c5b4
    frersv(d2);  % free ring element for d2%            1e1c5b5
    IF d2 = grp2 DO-SINGLE RETURN END;                  1e1c5b6
    %finished when have deleted top statement of
    last branch in group%                               1e1c5b7
```

Note that since the successor of the last statement in a plex is the father of the plex, no stack is needed in the above algorithm. Also note the manner in which the sub-pointers are modified to guide the traversal of the group. 1e1c6

As might be expected, copying a group is more complicated than deleting one since the structure cannot be modified during the process. 1e1c7

In very simplified form, the copy group algorithm is as follows: 1e1c8

> Starting at the first statement in the group, if the statement has substructure, copy that first; then copy the statement and move to its successor until the last statement in the group has been copied. 1e1c8a

When the group has been copied, it is inserted in the appropriate position in the ring in the same manner as a group being moved is reinserted into the ring. 1e1c9

3. Graphics Editing 1e1d

Blocks containing picture information are virtually indentical to those containing text information. The main difference is the replacement of statement data blocks by vector data blocks (VDB's). 1e1d1

A vector data block is made up of a header and an arbitrary number of lines and labels making up a picture. 1e1d2

The header contains much the same information as is held in the header of an SDB. Instead of character counts, however, the VDB header contains a count of the number of lines in the picture. 1e1d3

Following the header is a sequence of two-word buffers, each representing a line in the picture. 1eld4

The first word gives the position of one end of the line relative to the lower left-hand corner of the text of the statement. 1eld4a

The second word gives the position of the second end of the line relative to the first endpoint. 1eld4b

Following the buffers for the lines, each label in the picture is stored as a position (in the same format as the first word of a line buffer) and a text string. 1eld5

The current vector package was developed on a trial basis with a relatively small programming investment. As a result of this, the only graphic entities available are lines (vectors) and labels. A more sophisticated graphics system has been designed but not yet implemented. 1eld6

Selection of these entities is handled in the following manner. 1eld7

Line selection is done by finding the line that minimizes the difference between the sum of the squares of the distances from the endpoints of the line to the bug selection and the square of the length of the line. 1eld7a

This is a practical algorithm since the number of lines involved is small (under 100). 1eld7a1

Label selection is done by finding the label that minimizes the square of the distance between the bug selection and the second character of the label. 1eld7b

The "move vector" command will be explained as an example of vector editing. 1eld8

This command allows the user to move one end of a line to a new position. 1eld8a

When the line is selected, the end that is closer to the selection is offered as the end to be

moved. The user may request to move the other end instead by entering a backspace character. 1e1d8b

The next selection by the user specifies the new location for the end which is to be moved. 1e1d8c

Let end-1 be the end specified by the first word of the line buffer, and end-2 be the end specified by the second. 1e1d8d

If end-2 is to be moved, the second word of the buffer is replaced by the vector from end-1 to the selected position. 1e1d8e

If end-1 is to be moved, then the second word of the buffer is replaced by the vector from the selection to end-2, and the first word is replaced by the vector from the lower left corner of the text of the statement to the selection. 1e1d8f

The other vector editing commands are implemented similarly. 1e1d9

B. View Control 1e2

1. Jumps and Links 1e2a

The jump and link machinery is used to select statements to be displayed at the top of the text-viewing area of the screen. Generally speaking, jumps are made within a file and links are used either within or between files. Jumps may be made relative to the structure of the file, to specific statements, or relative to the jump or link ring. Links are to a dynamically determined location in a particular user's file, and can specify that display parameters are to be set when the link is taken. 1e2a1

The jump ring represents the chronological history of the last five jumps made within the current file. Each entry in the ring contains the PSID of the display-start statement and a word representing the display parameters. 1e2a1a

The link stack represents the last few links that have been made, and is only updated if the link is to a statement in another file. The entries in

this stack contain the user's number, the file name, the PSID of the display-start statement, and a word representing the display parameters. 1e2a1b

Code written in the content-analyzer SPL is used to locate and parse links. The four optional fields of the link are: 1e2a2

user name 1e2a2a

file name 1e2a2b

location within the file 1e2a2c

display parameters. 1e2a2d

In parsing a link, those fields which exist are delimited by pointers, which are subsequently used by routines to effect the link. 1e2a3

2. Sequence Generator 1e2b

The collection of routines known as the sequence generator is used to generate a sequence of statements starting from a given PSID and governed by the current view parameters. 1e2b1

The sequence generator work area is used to maintain information controlling the sequence. This work area is updated by the sequence generator whenever it is called. 1e2b2

The work area includes the following: 1e2b3

(1) PSID of current statement 1e2b3a

(2) Maximum and minimum level numbers for statements to be included in the sequence 1e2b3b

(3) Current statement's level 1e2b3c

(4) Address of Statement Vector Work Area (SVWA) 1e2b3d

(5) Address of last cell in SVWA 1e2b3e

(6) Address of current last cell used in SVWA. 1e2b3f

If statement numbers are being generated, the statement vector is generated for the statement in the SVWA. 1e2b4

The statement vector is a list of words, starting with the level of the statement and followed by entries containing the position of the statement in the corresponding plexes. 1e2b4a

For example, if the statement vector contains (4,1,5,3,2) then the statement is at level four and has statement number 1e3b. 1e2b4b

Once the work area has been initialized, the following algorithm is used to determine a candidate for the next statement in the sequence: 1e2b5

If keyword reorganization is being used, then the next PSID can simply be read from a file block. 1e2b5a

If a trail is being followed and the current statement contains the appropriate trail marker followed by the name of a statement in the current file, then: 1e2b5b

If the statement points to itself then the sequence is terminated by returning a -1; 1e2b5b1

Otherwise the PSID of the statement pointed to by the trail is returned. 1e2b5b2

If the current statement has a substatement which is within the current level bounds, then its PSID is returned. 1e2b5c

If the current statement has a successor statement which is within the level bounds, then its PSID is returned. 1e2b5d

Otherwise, a -1 is returned to indicate the end of the sequence. 1e2b5e

After a candidate statement has been selected in the above manner, it must be checked against the current content-analyzer pattern if the content analyzer is in use. If the analyzer is not being used, then the candidate is automatically accepted. 1e2b6

Flags in the ring element for the statement indicate whether the statement has been tested for the current pattern and whether it passed. 1e2b6a

If the statement has not been tested, then the sequence generator calls the code compiled for the pattern to make the test. This code is similar to that described for the content-analysis SPL in a previous section. The general flag is set true if the statement passes the pattern, and false if it does not. 1e2b6b

The process of selecting candidate statements is continued until (1) a statement passes the pattern or (2) the sequence is exhausted. 1e2b7

One of the primary uses of the sequence generator is in determining statements to be displayed. 1e2b8

3. Display Parameters 1e2c

The user has at his disposal two types of display parameters: those which control the selection processes employed by the sequence generator, and those which control the format of the display. 1e2c1

The format parameters control such things as the following: 1e2c1a

(1) The number of lines on the screen 1e2c1a1

(2) The position of various viewing areas on the screen 1e2c1a2

(3) The size of the characters 1e2c1a3

(4) Whether or not the name, number, or signature of a statement is displayed 1e2c1a4

(5) The number of lines per statement which are displayed 1e2c1a5

(6) Whether or not indenting is used to indicate the structure of the file 1e2c1a6

(7) Whether the file is displayed as text or as a tree (schematic). 1e2c1a7

The selection parameters control the following: le2c1b

(1) Whether content analysis is used le2c1b1

(2) Whether keyword reorganization is used le2c1b2

(3) Whether a trail is followed le2c1b3

(4) Whether frozen statements are displayed le2c1b4

(5) Whether the view is limited to only one branch le2c1b5

(6) To what extent the depth into the ring structure is limited. le2c1b6

With the exception of the display parameters which control such things as character size and location of viewing areas on the screen, the display parameters may be modified at any point in the specification of a command. le2c2

At certain points in the specification of some commands, the user is given the opportunity of changing the display parameters as part of the command. At other times the user may change them by using Case-3 keyset characters, which are not interpreted as part of a command specification. Furthermore, the availabilty of a display parameter which causes the display to be regenerated allows the user to treat the changing of display parameters as a pseudo-command. This can be done in the midst of specifying a normal NLS command. le2c2a

4. The User's Content Analyzer le2d

The user's content analyzer is essentially a subset of the programmer's content-analysis SPL, described elsewhere in this appendix. It is composed of two parts: a compiler and the code which is the product of the compiler. le2d1

The compiler is called by a user command to compile content-analysis code from a "pattern" written as text in the user's file (the syntax is that of the content-analysis SPL). le2d1a

A display parameter then determines whether or not the sequence generator is to execute this code for each of the statements which have passed all other selection criteria. 1e2d1b

If executed, the code scans the given statement searching for the specified content. If the search is successful, the statement is displayed; otherwise, it is not. 1e2d1b1

5. Keyword System 1e2e

The keyword system provides a rudimentary form of information retrieval in NLS. The result of a keyword search is a list of PSID's. This list is stored in the keyword file block. The following special terms are used in documenting the keyword system: 1e2e1

hit -- keyword that has been selected and has nonzero weight 1e2e1a

result -- one of the PSID's generated by KEYWORD EXECUTE 1e2e1b

a. Keyword File-Block Format 1e2e2

The keyword data consists of two tables: 1e2e2a

The first contains the PSID's of hits and their weights, with the PSID in the lower 11 bits and the weight in the upper 13. 1e2e2a1

The second contains the results of the most recent search as an ordered list of PSID's. 1e2e2a2

The first few words of the block contain information regarding the current status of these tables, such as the following: 1e2e2b

(1) Address of start of second table 1e2e2b1

(2) Address of item in second table last returned by the sequence generator to create display 1e2e2b2

(3) Address of last entry in second table 1e2e2b3

(4) Number of hits. 1e2e2b4

b. Generation of Results 1e2e3

The following algorithm is used to generate a list of results, given a set of selected keywords. 1e2e3a

A table is built with an entry for each result. Each entry takes two words, the first being the hash for the name of the statement, the second the score for the result (i.e., the sum of the weights for all hits referencing that result). The table is generated in the following manner. 1e2e3a1

For each hit, the statement specified by that PSID is searched for a certain string, which is currently set to be an asterisk followed by two spaces. This search is done by the content-analyzer POP that does unanchored scans. If the string is not found, then the next hit is considered. 1e2e3a1a

If the string is found, the algorithm then finds the names in the remainder of the statement. Each name is copied out of the text into the statement name register (STN). The algorithm then generates the hash for the name. This is compared to the previous entries to see if it already occurs in the table. If it does, then the score is increased by the weight of the current hit; otherwise, a new entry is created with score equal to the weight of this hit. 1e2e3a1b

After the entries have been accumulated in the above manner, the table is sorted according to score. 1e2e3a1c

The sorted entries are used to produce a list of results. The results are PSID's, so for the hash of each entry, the associated PSID must be found by searching the ring. 1e2e3a2

Finally, the information at the front of the file block containing the results is updated to show the new number of results. 1e2e3a3

This list of PSID's is used by the sequence generator when keyword reordering is called for by the user. 1e2e3b

6. Text Display 1e2f

a. General 1e2f1

The collection of routines known as CREATE DISPLAY is used to display in the text area of the user's screen those statements which are selected from the current file by the sequence generator. 1e2f1a

The statement selection process and the format of the display are under the user's control by means of VIEWSPECs and the "viewchange" command. 1e2f1a1

CREATE DISPLAY is called each time the user modifies his file, changes format parameters, selects a new candidate statement for the top of the text area, changes the statement selection parameters, or explicitly requests that the display be recreated. 1e2f1b

A call to CREATE DISPLAY does not imply that the entire display will be recreated. In fact, as little is done as possible in order to minimize file I/O. 1e2f1b1

The entire display is reconstructed from the display-start PSID only in the following cases: 1e2f1b2

(1) A change in the display-start PSID (caused by jumps, "load file" command, etc.) 1e2f1b2a

(2) Editing involving structural elements larger than statements 1e2f1b2b

(3) Changes in format parameters 1e2f1b2c

(4) Explicit user command recreate display. 1e2f1b2d

For statement-editing display changes, the display is updated only for those statements which have changed. 1e2f1b3

The display recreation is guided by the format parameters, such as truncation, and the output of the sequence generator, which is called to find the first statement in the sequence and for subsequent statements until (1) the last in the sequence has been encountered, or (2) the text area of the screen is full. le2f1c

b. Implementation Details le2f2

The main data areas used by CREATE DISPLAY are the following: le2f2a

(1) The display list le2f2a1

(2) The display list reference table (DLRT) le2f2a2

(3) The display buffers. le2f2a3

The entries in the display list are used by the display hardware and have the form of a word count followed by a buffer address. The display hardware processes the specified number of words from the buffer pointed to by the entry. le2f2b

For each line displayed in the text area, there are two entries in the display list. le2f2c

The first points to a one-word buffer (that is part of the DLRT entry for that line) that specifies the position of the start of the line on the screen. le2f2c1

The second points to a buffer that contains the actual character string that makes up the line. le2f2c2

For each line there is a four-word entry in the DLRT, containing information such as the following: le2f2d

(1) A T-pointer for the first character in the line le2f2d1

(2) The first and last column numbers containing text in the line (used in bug selection) le2f2d2

(3) The position on the screen of the left end of the line 1e2f2d3

(4) Flags denoting such things as the following: 1e2f2d4

(a) The line is null 1e2f2d4a

(b) The line contains special (nonprinting) characters 1e2f2d4b

(5) A copy of the second display-list entry for the line (used to restore the display list after displaying an error message). 1e2f2d5

For each PSID which is returned from the sequence generator, a display buffer, DLRT entries, and display-list entries are created. 1e2f2e

On the basis of the above description, the actions of CREATE DISPLAY should be clear for cases where the entire text area is being recreated. 1e2f2f

The series of statements determined by the sequence generator, starting from the statement specified for the display top, is used to fill the lines of the display, with the appropriate information being stored in the display list, DLRT, and display buffers. 1e2f2f1

In the case of text-editing changes, the display is only partially recreated; the process is as follows: 1e2f2g

The DLRT and display-list entries for the statements that were not edited are copied to auxiliary buffers. 1e2f2g1

If the content-analyzer flag is off or the edited statement passes the pattern, then a new display buffer, DLRT, and display-list entries are constructed for it. 1e2f2g2

When this is completed, the DLRT and display list are replaced by the auxiliary buffers and CREATE DISPLAY returns. 1e2f2g3

c. Bug Selection 1e2f3

It is appropriate to consider the problem of converting selections made by the user to valid character and statement specifications at this point, since bug selection makes use of data areas constructed by CREATE DISPLAY. 1e2f3a

Whenever input is read from the user work station, the coordinates of the bug are saved along with it. In the case where the input is meant as a selection by the user, the coordinates must be used to identify a character on the screen. The DLRT contains the information required to do this. 1e2f3b

The text area is "homogeneous," in that each line takes a fixed amount of space vertically and each character takes a fixed space horizontally. 1e2f3b1

Thus the coordinates of the selection can be easily converted to a character and line position in the text area. 1e2f3b2

This is only part of the problem, however, since the selection may be at a character position that does not contain a character. In other words, there are null areas in the text area and selections in these areas must be "rounded" to another position. 1e2f3b3

This rounding process is done using the information in the DLRT. 1e2f3b4

The DLRT has a flag indicating whether a line is null. These flags are checked and the selection moved up the screen until it is on a non-null line. 1e2f3b4a

The DLRT also specifies the first and last columns in the line containing a character. On this basis, the selection is moved to the left or right, if necessary, to put it on a position containing a character. 1e2f3b4b

It is often the case that bug selections must

be converted to T-pointers for operations such as editing. 1e2f3b5

If the line does not contain any special characters, which take up more than one character position in the SDB, the bug selection can be converted into a T-pointer directly from the information in the DLRT. 1e2f3b6

There is a flag in the DLRT which indicates whether the line contains any special characters, and a T-pointer for the first character in the line. 1e2f3b6a

If there are no special characters, the character count for column k is simply k greater than the count for the first character and is thus obtainable from the T-pointer in the DLRT entry. 1e2f3b6b

If the line does contain special characters, then the number of special characters in the line to the left of the selected character must be determined. Rather than store this value, it is computed directly from the SDB for the statement. This amounts to reformatting the line up to the selected character. 1e2f3b7

C. Calculator 1e3

The calculator gives the NLS user the ability to perform arithmetic operations using numbers selected from the text or entered from the keyboard. 1e3a

In addition, arithmetic expressions (functions) with named variables may be evaluated with the aid of a small compiler built into the calculator. 1e3b

The calculator stores numbers internally in a fixed-length decimal notation (currently using sixteen digits to the left of the decimal and seven to the right). 1e3c

The arithmetic routines work with numbers that have been "unpacked" into an "accumulator," one digit to a word. 1e3d

The multiplication algorithm will be briefly outlined as an example. 1e3e

The multiplicand and the product are in unpacked form. 1e3e1

Digits are read one at a time from the low-order end of the multiplier. 1e3e2

The multiplicand is initially "aligned" with the low-order end of the double-length partial product. During the course of the multiplication, they are realigned by "moving" the multiplicand toward the high-order end of the product. 1e3e3

The first step of the algorithm is to zero the partial product. 1e3e4

Then, until all the digits in the multiplier have been processed, the following algorithm is repeatedly executed: 1e3e5

(1) Read, and convert to the equivalent binary number, up to four multiplier digits at a time, thus forming a composite multiplier digit. 1e3e5a

(2) For each digit in the multiplicand, multiply it (using the hardware binary multiplication) by the composite multiplier digit, and add the result to the corresponding digit in the partial product. 1e3e5b

This takes advantage of the unpacked form to allow "digits" in the partial product to take on very large values. Carries out of the partial-product digits are propagated only once, at the end of the algorithm. 1e3e5b1

(3) Realign the multiplicand to the left by the number of digits read from the multiplier. 1e3e5c

Now propagate the carries in the partial product to finish the multiplication. 1e3e6

The calculator contains a small operator-precedence compiler for arithmetic expressions. 1e3f

The compiler produces both code to be interpreted and a

symbol table of the variables used in the expression. The symbol table grows toward higher addresses, while the code grows from the other end of the same block of memory. 1e3g

When the user asks to evaluate the expression, the program asks him to supply values for the variables. The user may fix a variable to a particular value and tell the program not to demand a new value for it. When all variables have been given values, the code compiled for the expression is interpreted and the result transferred to the "accumulator" of the calculator. 1e3h

For each variable in the expression, the symbol table contains the following information: 1e3i

(1) The name of the variable (as an A-string, so that it can be displayed in the command feedback line when the user is asked to give it a value) 1e3i1

(2) The current value of the variable 1e3i2

(3) Flags indicating whether the user should be asked to supply a value for it when the expression is evaluated, and if so whether it has been given a value during the current evaluation. 1e3i3

The code compiled for the expression is made up of the following instruction types: 1e3j

(1) Push values on the stack 1e3j1

(a) Push identifier (specified by the address of the value to be pushed) 1e3j1a

(b) Push constant (the value of the constant follows the instruction in the code) 1e3j1b

(2) Perform arithmetic operations with values on top of stack (unary minus, add, subtract, multiply, and divide) 1e3j2

(3) Halt 1e3j3

The interpreter for the code simply manipulates the stack and calls the appropriate arithmetic routines. 1e3k

D. Processors 1e4

1. File Cleanup 1e4a

The file cleanup program serves to verify (and perhaps even restore, with a bit of luck) the internal soundness of an NLS file. 1e4a1

The program goes through the following stages: 1e4a2

(1) For each structure block: 1e4a2a

Set all the name hashes to zero. 1e4a2a1

Check the free list and mark elements on the free list by setting their hashes to 1. 1e4a2a2

Verify the used cell count for the block. 1e4a2a3

(2) For each text block: 1e4a2b

Check the free space pointer. 1e4a2b1

Check each SDB by doing the following: 1e4a2b2

Compare the length given in the first word of the header to the character count. 1e4a2b2a

Check that the last character is really an end character. 1e4a2b2b

Check that the name character count is reasonable. 1e4a2b2c

Mark SDB's that pass these tests by "OR"ing 360000000B into first word. 1e4a2b3

If the SDB fails any of the tests, then move the free space pointer up to that point and give up on the rest of that block. 1e4a2b4

(3) For each graphics block: 1e4a2c

The process is similar to the process for text blocks. 1e4a2c1

At the end of these stages the entire file has

been inspected once. During this a special routine has handled the loading of file blocks. If at any time there is a "bad" file block (i.e., one that contains an error), it tries to recover by changing the type of the block if that is in error and recalculating the checksum if that is in error. 1e4a2d

File cleanup now continues with a second pass. 1e4a2e

(4) Check the actual structure of the ring. 1e4a2f

Start from the origin and work through, not trusting the head and tail flags. This requires keeping a stack of father PSID's and comparing each successor to the father. 1e4a2f1

Mark ring elements that are used in the structure by setting their hashes to 2 (first making sure that their names are zero, meaning unused, and not one, meaning on the free list). 1e4a2f2

Mark data blocks (both SDB and VDB) of ring elements in the structure, as used, by changing the top six bits in the first word to 34B instead of 36B. 1e4a2f3

Correct errors in head and tail flags if any are found. 1e4a2f4

Errors in structure are handled as follows: 1e4a2f5

If the bad statement is the head of a plex, then that plex is discarded. 1e4a2f5a

Otherwise the remainder of the plex is discarded. 1e4a2f5b

This discarding is done by linking together good parts of the ring. 1e4a2f5c

Thus in the first case the father of the bad statement simply no longer has any substructure. 1e4a2f5c1

In the other case the last good member of

the plex becomes the tail of the plex. 1e4a2f5c2

If a statement that has valid structure has a bad data block associated with it, then a dummy SDB is created for the statement and file cleanup continues. 1e4a2f6

(5) Look for "lost" SDB's and ring elements. 1e4a2g

Ring elements that still have name hashes of 0 are neither on the free list or in the structure. These are now put on the free list. 1e4a2g1

SDB's that still have 360000000B in their first word are not pointed to by any statement. These are now marked as garbage. 1e4a2g2

Marks on SDB's are now erased. 1e4a2g3

(6) The name hashes for all ring elements in the structure are now recomputed. 1e4a2h

This completes the cleanup of the file. 1e4a3

2. File Compaction 1e4b

The basic objective of the file compactor is to reduce the number of SDB blocks in a file by combining the contents of these blocks and eliminating resultant empty blocks. In addition, empty spaces in the random file are eliminated by packing the file into contiguous blocks. Structure blocks are not compacted. 1e4b1

SDB blocks with fewer than a fixed number of unused cells are not processed -- thus compaction for files which need little or no compacting will be a relatively quick operation. 1e4b1a

3. Output Processor 1e4c

The Output Processor is used to produce hard copy from NLS files. The output of this process includes formatted files for a printer, a Dura typewriter, and a Stromberg-Carlson microfilm machine. 1e4c1

The format of the output is controlled by means of directives. 1e4c2

These are parameters for numerous variables such as page dimensions, page numbering, and "on/off switches" for a large set of format options. The user may control these parameters by means of special strings of text (i.e., output-format commands) embedded in the file text. These command strings, which are also called "directives," are normally suppressed from the hard-copy output. 1e4c2a

A full set of directive default values for each type of device has been established; these values may be overridden by directives imbedded in the text of the file. 1e4c2b

The Output Processor runs as a subprocess of NLS and has one page -- a buffer -- in common with it. This process, like the compilers, utilizes the statement-selection mechanisms of NLS to obtain its input data. Thus level clipping, content analysis, keyword reordering, trails, and so forth can be used to control what is output via the Output Processor. 1e4c3

4. Compilers 1e4d

The languages developed by ARC for internal use are discussed in the main body of this report. Source code for any of these languages may be written in an NLS file and output directly from NLS to the appropriate compiler. 1e4d1

:4874, |0/0|/70 |730:32 MGC ; .DPR=|; :P2ROME, 07/|2/70 |94|:07 DGC ;
EDITING CHANGES DONE .DSN=1; .RTJ=0; .DPR=0; .DSN=1;.DPR=0;

ARC TASK SCHEDULES

HARDWARE schedule	WKE																
Month	Oct				Nov					Dec				Jan				Feb
Week	4	1	2	3	4	1	2	3	4	5	1	2	3	4	1	2	3	4	5	1	2	3	4
Net to Xcore																							
Power wiring																							
Move 940 cons.																							
Move ER																							
Disp in Xcore																							
2nd disp in xcore																							
Move xcore																							
Rewire IDC																							
IDC to Xcore																							
TTY patch panel	x																						
Disc modify	x																						
PDP-10 Deliver		x																					
PDP-10 Install		xx																					
Paging Box Mods			xxxxx																				
Paging box arrives						x																	
Paging Box Checkout							x																
Cybernex debug					xxxxxx																		
System operational								x															

SOFTWARE schedule	WHP																
Month	Oct				Nov					Dec				Jan				Feb
Week	4	1	2	3	4	1	2	3	4	5	1	2	3	4	1	2	3	4	5	1	2	3	4
Strmnp																							
SPL compiler																							
SPL source																							
New NLS																							
Libe																							
Tree 2.5																							
L|O																							
Debug L|O																							
L|O on |O	x																						
Xcore monitor	x																						
Tree 3	x																						
miniTodas	x																						
Xcore NLS	x																						
SPL-|O	xx																						
Libe 2	xx																						

ARC TASK SCHEDULES

```
':4875', 09/24/70 0857:58 MEJ ;  :SCHED, 09/22/70 0656:31 JCN ;.HED="
4875 JCN 22SEP70
ARC TASK SCHEDULES";.DPN=|;.SOR=|;
.SNF=72;.MCH=65;.PGN=0;.DSN=|;.DPR=0;
```

Proposal for Research No. ESU 69-119 1

EXPERIMENTAL DEVELOPMENT OF A SMALL COMPUTER-AUGMENTED INFORMATION SYSTEM 2

I INTRODUCTION 3

A. The Augmented Human Intellect Research Center 3a

The Augmented Human Intellect Research Center (AHIRC) of Stanford Research Institute's Information Science and Engineering Division is an externally supported, multiply sponsored group of 24 persons working in close cooperation on the problem of "augmenting the human intellect." "Augmentation" is a term indicating the extension, improvement, and amplification of the intellectual capabilities of humans, both as individuals and as working groups or teams. 3a1

The current approach to this goal concentrates on the use of highly interactive computer systems designed to aid individuals and groups in manipulating the information that they work with. This "manipulation of information" includes the following: 3a2

Externalization and storage of "ideas" in symbolic form -- for example, English text, or drawings, or computer programs, or special structures for relating various stored items. 3a2a

Studying the stored material, by means of high-speed computer display of the text, drawings, etc., coupled with specialized information-retrieval techniques geared for this type of application. 3a2b

Modifying and updating the stored material by means of a highly sophisticated system of interactive editing commands, which permit a range of operations from detail editing to wholesale rearrangement of information structures. 3a2c

II SUMMARY 4

A. Objectives 4a

We propose an experimental investigation of techniques for

the management, within AHIRC, of a collection of externally derived information (an "intelligence" collection), with the eventual purpose of creating, using, and developing an "intelligence system" adapted to our particular needs. We expect to develop design principles applicable to information systems for other groups that will be acquiring advanced interactive computer tools. 4a|

We have available a sizable repertoire of special on-line techniques for information handling, plus special capabilities for the development of more techniques. The objective of the proposed research is to develop a systematic application of these techniques and capabilities to the management of our growing collection of "intelligence" (external material), and to the methods of retrieval, extraction and integration of information by our on-line researchers. 4a2

B. Current Status 4b

An important characteristic of AHIRC is its "bootstrapping" strategy for research on augmentation. All systems designed by the Center are intended for actual, practical use in the Center itself; once designed and implemented they are used heavily on a day-to-day basis. This means that AHIRC staff are both experimenters and experimental subjects, and the result is strong "evolutionary" pressure upon the design process. 4b|

Each special development made by the Center in any of its areas of concern (including software design, management, etc.) evolves and is used within an integrated working environment that provides an otherwise unavailable context for evaluation of the real usefulness of the development. Such evaluation is of great importance in designing and developing tools and methodologies for the future world of on-line working groups. 4b|a

One focus of effort within this approach is the development of systems for managing the working information of the group. Considerable work has been done in the development of small, essentially personal information systems, and the Center is beginning to investigate the problems of somewhat larger systems of coordinated working records for use by the group as a whole. 4b2

We want to devote simultaneous attention to managing our

"external" records, as herein proposed. As a beginning, we have collected some 4000 items (books, periodicals, clippings, etc.) over the past ten years. This "XDOC" (external documentation) collection has a very rudimentary catalog whose citation entries are stored by accession number in computer-held files which we can search on content from on-line CRT consoles, but for which there is no formal indexing. With its present form and its current primitive management and usage methods, the XDOC collection is little used or valued. 4b3

We need to expand the coverage so that we can manage all important forms of useful externally derived information: trip reports, visitor records (including notes on information acquired from visitors), catalog and hearsay information on hardware, press clippings, conference announcements, etc. 4b4

We need to learn to apply our advanced interactive computer aids to the procedures for entering, filtering, cataloging and indexing. We need to explore various forms of file organization and indexing which, together with the associated methods of retrieval and extraction that our interactive aids offer, could provide a practical and useful "intelligence" system for us. 4b5

Our current on-line tools and methods are applicable to these needs. In addition, further tools and methods of a highly relevant nature are currently under development for various special purposes within AHIRC. 4b6

C. Approach 4c

1. General 4c1

This proposal represents a short-term and relatively small project in a long-term activity, all of whose components are continuously developing. 4c1a

We plan to launch a working "intelligence" system, and to pass through several phases of development. At the end of a year we expect to have an initial system which will be usable and reasonably effective, incorporating unusual features and revealing further possibilities. 4c1a1

We expect to spend most of the project resources at the information-systems level (procedures, file

organization, indexing methods, etc.) and a relatively small portion on special software developments. 4c|b

We are already very strong in relevant interactive computer aids, in techniques for programming new aids, and in techniques for tailoring the function and control procedures of these aids to the user's needs. 4c|b|

The development of the system will follow the needs of the AHIRC staff for accessing and integrating externally derived information. Each incremental allocation of this project's system-development resources will be aimed either at achieving an increase in system utility or at experimentation on means for increasing utility. 4c|c

2. Specific Approach 4c2

We plan the following specific tasks: 4c2a

(1) Conduct a bibliographic search for material relevant to our goal of setting up an "intelligence" system for use by our group. 4c2a|

Using the results of this search as an experimental information base, design and use prototype procedures, file structures, indexing, etc. to develop a better feeling for the needs, problems, and possibilities. 4c2a|a

(2) Concurrently, develop a working relationship with a specialist in library science and/or information retrieval for assistance in carrying out the proposed research. 4c2a2

(3) Make a straightforward, first-pass organization of our existing collection, to provide consistent cataloging procedures for the variety of materials we will be dealing with. Conduct initial development of indices for use in retrieval of information from the collection. 4c2a3

The aim of this will be a usable starting system, implemented with minimal software investment consistent with efficient subsequent development toward anticipated improvements. 4c2a3a

(4) Give special attention to several specific

needs, such as a hardware-products reference system, a correspondence record system, and bibliographic studies conducted under other projects in AHIRC. 4c2a4

(5) Evolve a plan for developing the system and its usage. Consider special possibilities for integration into the "intelligence" corpus of notes, reference linkages from the group's working records, subsequent extracts of already cited items, partial extracts, etc. to enrich the information and to provide more access paths. 4c2a5

(6) Follow a continuing cycle of improvement and development.

4c2a6

III DISCUSSION 5

A. General 5a

AHIRC has developed an on-line computer system with which users can study text and associated line drawings on video displays. Certain features of this system provide a unique framework for building an "intelligence" system. 5a|

The system includes powerful editing commands that enable the user to compose and modify text quickly. 5a|a

We can generate microform and paper representations of the text and drawings stored in the computer. Documents can be published directly from this hard copy, using photo-offset techniques. 5a|b

Information can be stored and displayed in a hierarchical structure of "statements". Thus we can construct classification schemes easily. 5a|c

(A "statement" is a structural unit of stored information, and can contain any sort of text string. Statements are frequently used as the equivalent of conventional paragrpahs, or indiividual entries in lists.) 5a|c|

Another feature allows a user to "link" any statement in the system with any other, creating trails of associations. 5a|d

(A link is a machine-executable equivalent of the conventional "cross-reference." Thus a link establishes an association between two statements; a subsequent user, seeing the link embedded in a statement, may cause the associated statement to be displayed instantaneously.) 5a|d|

Another user can follow these trails of associations, and add his own, if he wishes. With these links we can build and study complicated relationships within an information system. 5a|e

An interactive content analyzer lets the on-line user define patterns of words and phrases on-line and retrieve statements that contain these patterns. This

tool can be applied to a fairly simple catalog to provide considerable retrieval power even with no explicit indexing. Applied to appropriately structured indices, the content analyzer adds a new dimension of retrieval power. 5a|f

Another interactive tool lets us group related statements under a single identifier, called a "keyword." One can select one or more of these keywords and display the statements referenced by all of these keywords; a "scoring" technique is employed so that the statements referenced by the greatest number of the selected keywords appear at the top of the list. 5a|g

Since statements may contain links to other files, this capability can be used to retrieve whole files, as well as simply retrieving statements within a file. 5a|g|

Using the keyword feature with a hierarchical catalogue, we can retrieve documents relevant to one area of interest or to several areas. 5a|g2

Our current techniques for composing, modifying, and publishing would alone have a unique impact upon the way in which the group's "intelligence" system could be set up and maintained; our study aids, including the content analyzer and keyword system, add unique possiblities for using this system. 5a2

We plan to go slowly in settling on an over-all design for the "intelligence" system. We expect to go through considerable study, thought, and pilot experimentation before we commit the whole system to an integrated design. 5a3

B. Design Considerations for an "Intelligence" System 5b

Our "intelligence" system must satisfy many different kinds of information needs. 5b|

A small research group receives information from many different sources and in many different forms. Journals, books, newspapers, informal conversations, correspondence, conferences, visitors, and manufacturers are only some of the sources of our working information. This information may be recorded in print, in computer-held files, on audio tape, on film, or in microform. 5b|a

We need to cite all of these items in one central catalog, and we need to organize these citations so that we can easily find all the information we have about a particular topic. 5b|b

We also need to provide techniques that enable us to individually tailor "views" of the items in the collection. For example, if a member of the group is studying commercially available video devices, he may wish to add a trail of associational links that lets him, and other members of his team, compare the prices of these devices at a glance. 5b|c

Our "intelligence" system should provide a well-organized library collection that can be expanded or reorganized easily. It should allow each individual to maintain the integrity of his own personal collection while sharing this information with the group. 5b2

C. Methodology 5c

Our initial efforts at designing an "intelligence" system would focus on the techniques for organizing the collection and for retrieving information from it. The study of these two areas of concern would proceed dialectically: experimental catalogs and indices would be organized to conform to the requirements of a particular retrieval technique, and the development of the retrieval tools would in turn be influenced by the demands of efficient schemes of organization. 5c|

1. Specific Topics for Study 5c2

During the period of this contract we hope to study several specific problems. 5c2a

a. Conventions for Organization of Central Catalog 5c2b

The usefulness of our collection will be strongly influenced by the conventions we adopt for citing items in the central catalog. This catalog should provide a primary source of information for generating indices and subject classifications of items in the collection. 5c2b|

The central catalog should be organized so that when the user retrieves the citation for a given

item in the collection, he may also retrieve the citations for all other items in the collection which are known to refer to the given item. 5c2b1a

In addition, the catalog conventions must be flexible enough to admit many different kinds of items, while having the standardization necessary for convenient machine retrieval. 5c2b2

For example, we may wish to put the tape recording of a conference in the collection. Its citation might include a brief abstract of the material discussed, the names of panel members, and the date and location of the proceedings. 5c2b2a

b. Procedures for Entry of New Material 5c2c

The procedures used to enter items into the collection and catalog must also be developed carefully and experimentally. We hope that the collection can be maintained by someone without the training of a professional librarian, so these procedures must be uncomplicated. 5c2c1

c. Treating Catalog Items According to Their Importance 5c2d

The collection must be organized flexibly enough so that information about an item reflects its current importance in the working atmosphere of the group. For example, a journal article may enter the collection with the notation that a reprint of it has been requested from the author. After it arrives, it may become important enough as a working paper to transcribe into machine-readable form and keep on-line. As the work of the group progresses, the article may be used very infrequently; at this point it should be put into a magnetic tape archive. The procedures for maintaining the collection must allow an item to evolve through these stages. 5c2d1

2. Retrieval Tools and Techniques 5c3

As we experiment with different schemes of organization and administrative procedures, we will also be developing retrieval tools. 5c3a

We plan to begin studying the information-retrieval techniques used in other systems, with the aid of a

professional librarian. We intend to incorporate this information into our "intelligence" system and to study and manipulate it as the working body of information with which to try out various retrieval techniques. 5c3a1

We will construct experimental indices of the items in the collection. These indices, and other possible classification schemes such as thesauri, will be used to locate citations in the catalog file by subject. We intend to use these indices and judge their relative merits as retrieval techniques. 5c3a2

Several features of our current system -- especially the content analyzer and the keyword system -- will prove useful as retrieval tools to extract information and citations from indices. We can use these same tools to create the experimental indices and classification schemes. 5c3a2a

Besides such current techniques, we are considering a number of improvements over the coming year that would increase the power of our tools quite significantly. If implemented, these improvements would be developed cooperatively by several of the various projects within AHIRC, including this proposed project. 5c3a2b

We may expand the power of our "keyword" operations. These extensions would let us save references reordered by the keyword system in their new order. We could use this technique to build comprehensive classification schemes from reasonably simple ones. This expanded keyword system would also let each member of the group construct his own classification schemes and store them for others to use. 5c3a2b1

We may also develop a batch-processor facility that could reorganize files in accord with a user's specifications. Such a processor could convert the entry format of our catalog files. It could also collect all of the items referenced by a trail of links in a new file. Such processors would also be useful for updating catalog and index files from information entered in a format best suited to the clerk. 5c3a2b2

Another aspect of retrieval in our "intelligence" system centers around the problem of locating, from the on-line catalog, items that are not in machine-readable form. The citation catalog will indicate the physical location of each source document. Procedures must be developed for moving this document to another office and updating the information in the central catalog. 5c3a3

We also plan to develop procedures for generating microform and paper versions of the on-line catalog, and of selected indices or portions of indices. We already have the tools to perform the mechanical part of these operations; however, we need to adopt conventions that would make these documents useful as working bibliographies or for publication.

5c3a4

IV PERSONNEL 6

It is planned that the Principal Investigator will be Dr. Douglas C. Engelbart, Head, Augmented Human Intellect Research Center. Dr. Engelbart's Social Security number is [redacted]. 6a

Other significant contributions, including project management, are anticipated from Mary S. Church, Programmer. 6b

V ESTIMATED TIME AND CHARGES 7

It is proposed that the research work outlined herein be performed during a period of twelve months, starting 8 February 1970. 7a

Pursuant to the provisions of ASPR 16-206.2, attached is a cost estimate and support schedule in lieu of the DD Form 633-4. Also enclosed is a signed form complete except as to the "Detail Description of Cost Elements." 7b

VI REPORTS 8

A final report will be submitted upon completion of the work. 8a

During the period of the proposed work, we expect to be developing a "Handbook," which will be a comprehensive description and history of all work in the Center, suitably structured for study and manipulation with the Center's computer aids. It is anticipated that individual projects, such as the proposed work, will be covered in the Handbook as "chapters" and reports will be produced in hard copy directly from the Handbook (with suitable editing to produce useful hard-copy formats). Depending on the state of Handbook development at the completion of the proposed work, the final report may be in this form. 8b

VII GOVERNMENT-FURNISHED EQUIPMENT 9

The performance of the proposed work will involve the use of equipment furnished under Air Force Contract F30602-68-C-0286 and NASA contract NAS1-7897. 9a

VIII CONTRACT FORM 10

It is requested that any contract resulting from this

proposal be awarded on a cost plus fixed fee basis. 10a

IX RELATED SUPPORT FROM OTHER AGENCIES 11

The Augmented Human Intellect research program has been supported largely by the Advanced Research Projects Agency on a continuing basis. Support has also been provided by NASA-Langley Research Center and the U.S. Air Force Rome Air Development Center.

11a

X ACCEPTANCE PERIOD 12

For staff scheduling purposes, this proposal will remain in effect until 31 December 1969. If additional time is required for its consideration, the Institute will be glad to consider a request for an extension of the period. 12a

XI BIOGRAPHIES 13

The following professional biographies are presented as being representative of SRI personnel who may contribute to the proposed work. 13a

Douglas C. Engelbart, Head, AHIRC
Information Science and Engineering Division 13b

Dr. Engelbart received a B.S. degree in Electrical Engineering from Oregon State College in 1948. 13b1

In 1953 he received an E.E. degree from the University of California; his thesis described the logical design and programming of a drum-type general-purpose computer to obtain increased flexibility and speed by optimizing the utilization of the electronic register capacity. 13b1a

In 1955 he received a Ph.D. degree in Electrical Engineering, also from the University of California; his thesis dealt with the development of special gas-discharge tubes for computer use. 13b1b

While studying at the University of California, he was an Associate in Electrical Engineering. 13b1c

He served as an Assistant Professor in 1955-1956. 13b1d

Since 1959, Dr. Engelbart has been principally occupied in developing a program at Stanford Research Institute aimed at improving human intellectual effectiveness through real-time computer aid. 13b2

First with only Institute in-house support, and since March 1961 with joint support from AFOSR, he formulated a comprehensive conceptual framework for man-machine studies with both broad and specific research goals. 13b2a

The specific goals have been translated into the

establishment of a computer-based experimental laboratory and a number of on-going projects within a coordinated and growing program for which Dr. Engelbart serves as Head. 13b2b

From 1948 to 1951, he was an Electrical Engineer in the Electrical Section at the Ames Laboratory, Moffett Field, California. 13b3

In 1955-1956, Dr. Engelbart was a consultant to Marchant Research, Inc., Oakland, where development work has been carried out on patents bought from him. 13b3a

In 1956 he formed and directed a corporation, Digital Techniques, Inc., which in 1956-57, did further development work on his inventions. 13b3b

In October 1957, Dr. Engelbart joined the staff of Stanford Research Institute, where he was initially concerned with basic developmental work on magnetic components for computers and with other fundamental research into the physical techniques of computers. 13b4

In 1959 he began, under Institute sponsorship, to expand and develop the basic concepts for the Augmented Human Intellect program which he had developed independently since 1950. 13b4a

His fields of specialization have included circuits, special components, logical design, and programming of digital computers; vacuum and gas-discharge techniques; large intercommunication systems; wind-tunnel drive and control systems; electromechanical control systems; information systems; and man-machine systems. 13b5

Dr. Engelbart is a member of Pi Mu Epsilon, Sigma Tau, Tau Beta Pi, Phi Kappa Phi, Sigma Xi, Eta Kappa Nu, the Institute of Electrical and Electronics Engineers, and the IEEE Group on Computers (Electronic). 13b6

He was Chairman of the San Francisco Chapter of IRE PGEC in 1959-1960 and has served as member of the IRE Solid State Circuits Subcommittee 4.10 and of the IEEE Cybernetics Committee. 13b6a

Mary S. Church, Programmer, AHIRC
Information Science and Engineering Division 13c

Specialized Professional Competence |3c|

Development of large-scale multifunction computer systems; integration of nonstandard devices (such as specialized computers) into generalized software systems. |3c|a

Current Research Assignment at SRI |3c2

Development of information-retrieval center for a computer network. |3c2a

Other Professional Experience |3c3

Columbia University, Technical writer; supervisor of systems programming, responsible for programming group implementing general operating system to support large computers, small satellite computers, and low-speed terminal devices. |3c3a

Academic Background |3c4

B.S. in biochemistry (1964), Radcliffe College. |3c4a

Graduate work in English literature (1965-67), Columbia University. |3c4b

Professional Associations |3c5

Association for Computing Machinery. |3c5a

Mary G. Caldwell, Research Assistant, AHIRC
Information Science and Engineering Division |3d

Professional Experience |3d|

Research Assistant, Medical Fact Bank Project, Missouri Regional Medical Project 1967-68. |3d|a

Conducted feasibility study of use of IBM 1050 audiovisual system (on-line, remote) for planned Multidisciplinary Learning Laboratory. |3d|a|

Planned construction of a medical thesaurus for Fact Bank retrieval. |3d|a2

Academic Background |3d2

B.A. in English (1967), University of Missouri. 13d2a

David Casseres, Technical Writer, AHIRC
Information Science and Engineering Division 13e

Specialized Professional Competence 13e1

Text-handling procedures in an automated environment. 13e1a

Information structures for computer-held text. 13e1b

Technical writing. 13e1c

Representative Research Assignments at SRI (since 1966) 13e2

Construction and coordination of documentation from existing computer-held information. 13e2a

Techniques for generation of documentation using advanced computer-aid systems.

13e2b

Other Professional Experience 13e3

Technical Report Editor, Engineering, SRI, 1965-1966. 13e3a

Academic Background 13e4

B.A., Reed College, 1965. 13e4a

Professional Associations 13e5

Member, Association for Computing Machinery.

13e5a

XII BIBLIOGRAPHY OF AHIRC PUBLICATIONS 14

Note: This bibliography is arranged in chronological order. Reports with AD numbers are available from Defense Documentation Center, Building 5, Cameron Station, Alexandria, Virginia 22314. 14a

1. D. C. Engelbart, "Special Considerations of the Individual As a User, Generator, and Retriever of Information," Paper presented at Annual Meeting of American Documentation Institute, Berkeley, California (23-27 October 1960). 14b

2. D. C. Engelbart, "Augmenting Human Intellect: A Conceptual Framework," Summary Report, Contract AF 49(638)-1024, SRI Project 3578, Stanford Research Institute, Menlo Park, California (October 1962), AD289565. 14c

3. D. C. Engelbart, "A Conceptual Framework for the Augmentation of Man's Intellect," in Vistas in Information Handling, Volume 1, D. W. Howerton and D. C. Weeks, eds., Spartan Books, Washington, D.C. (1963). 14d

4. D. C. Engelbart, "Augmenting Human Intellect: Experiments, Concepts, and Possibilities," Summary Report, Contract AF 49(638)-1024, SRI Project 3578, Stanford Research Institute, Menlo Park, California (March 1965), AD640989. 14e

5. D. C. Engelbart and B. Huddart, "Research on Computer-Augmented Information Management," Technical Report ESD-TDR-65-168, Contract AF 19(628)-4088, Stanford Research Institute, Menlo Park, California (March 1965), AD622520. 14f

6. W. K. English, D. C. Engelbart, and B. Huddart, "Computer-Aided Display Control," Final Report, Contract NAS1-3988, SRI Project 5061, Stanford Research Institute, Menlo Park, California (July 1965). 14g

7. W. K. English, D. C. Engelbart, and M. L. Berman, "Display-Selection Techniques for Text Manipulation," IEEE Trans. on Human Factors in Electronics, Vol. HFE-8, No. 1, pp. 5-15 (March 1967). 14h

8. D. C. Engelbart, W. K. English, and J. F. Rulifson, "Study For The Development of Human Intellect Augmentation Techniques," Interim Progress Report, Contract NAS1-5904, SRI

Project 5890, Stanford Research Institute, Menlo Park, California (March 1967). 14i

9. J. D. Hopper and L. P. Deutsch, "COPE: An Assembler and On-Line-CRT Debugging System for the CDC 3100," Technical Report 1, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (March 1968). 14j

10. R. E. Hay and J. F. Rulifson, "MOL940: A Machine-Oriented ALGOL-Like Language for the SDS 940," Technical Report 2, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (April 1968). 14k

11. D. C. Engelbart, W. K. English, and J. F. Rulifson, "Development of a Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research," Final Report, Contract AF 30(602)4103, SRI Project 5919, Stanford Research Institute, Menlo Park, California (April 1968). 14l

12. D. C. Engelbart, "Human Intellect Augmentation Techniques," Final Report, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (July 1968). 14m

13. D. C. Engelbart, W. K. English, and D. A. Evans, "Study for the Development of Computer-Augmented Management Techniques," Quarterly Progress Report 1, Contract F30602-68-C-0286, SRI Project 7101, Stanford Research Institute, Menlo Park, California (October 1968). 14n

14. D. C. Engelbart and W. K. English, "A Research Center for Augmenting Human Intellect," in AFIPS Proceedings, Vol. 33, Part One, 1968 Fall Joint Computer Conference, pp. 395-410 (Thompson Book Co., Washington, D.C., 1968). 14o

15. D. C. Engelbart and Staff of the Augmented Human Intellect Research Center, "Study for the Development of Human Intellect Augmentation Techniques," Semiannual Technical Letter Report 1, Contract NAS 1-7697, SRI Project 7079, Stanford Research Institute, Menlo Park, California (February 1969). 14p

16. D. C. Engelbart, W. K. English, and D. A. Evans, "Study for the Development of Computer Augmented Management Techniques," Interim Technical Report RADC-TR-69-98, Contract F30602-68-C-0286, SRI Project 7101, Stanford Research Institute, Menlo Park, California (March 1969). 14q

17. D. C. Engelbart and Staff of the Augmented Human Intellect Research Center, "Study for the Development of Human Intellect Augmentation Techniques," Semiannual Technical Letter Report 2, Contract NAS 1-7897, SRI Project 7079, Stanford Research Institute, Menlo Park, California (August 1969).

14r

```
':4876', 09/24/70 0948:25 MEJ ;  :PRONR, 10/28/69 1630:25 DGC ;
.SINCE(69/10/27 1355:00); [-'" "intelligence" -'"]; '.LLL; ["bora"];
[-'. 2$NP]; ↑P| SE(P|) < -'.; ['. 1$|NP];  .DSN=|; .RTJ=0;  .LSP=0;
.PGN=0; .DPR=0;
```

SUBJECT: Special telephone service for the NIC R&C network

OBJECTIVE: To provide all NET people with 24-hour access to the NIC (or an assured answering service) at no cost to them (with moderate convenience) and at the lowest possible cost to NIC.

REFERENCES

Roy Sexton -- SRI Communication Services -- x2700

Keith T. Burton -- PTT Special Services Consultant -- x4442

Jan Cardwell -- PTT Operator Supervisor -- x4442

Regular Telephone Operators (for long-distance rates)

RECCOMENDATIONS (SEE FOLLOWING SECTIONS FOR DETAILS)

For the NIC we should subscribe for three telephone lines, two to be used for incoming calls (on a hunting basis), the other for outgoing calls.

The incoming lines should be set up for enterprise service from all exchanges where there are NET sites. This will permit all NET people to call us at no charge to themselves and will provide us with itemized billing information to be used in determining possible alternatives for the future (future being anywhere from a week to 2 or 3 months away).

It seems unlikely that we would ever be justified in subscribing for California WATS service. However, it is very likely that interstate measured WATS will prove more economical for incoming out-of-state calls than the enterprise, but it seems reasonable at this time to defer taking that step until we have some hard data regarding actual phone usage.

We should obtain 4 pushbutton phones to service JBN WLB JTM and xxx (whoever serves as our R&C agent). In addition, DCE should be connected into the NIC facilities on his present phone.

The phones would have an outgoing and two incoming NIC lines, a normal SRI extension, an intercom line.

The phones should be equipped with conferencing facilities between the incoming and outgoing lines. The primary incoming line should conference with the outgoing line (to reach an SRI person other than the five connected to the NIC lines, one would dial the outgoing line into the SRI switchboard); the secondary line should conference with the SRI extension (to reach an outside number, one would dial through the SRI switchboard).

Card dialers should be obtained for DCE JBN WLB and xxx.

The three NIC lines should also come into NIL's and BER's call directors.

I suggest that we defer ordering tape-recorder interface devices until we can get better info on what's involved. 7e
We should subscribe to an answering service in lieu of obtaining an automatic answering device. Lacking any other criteria, I suggest we accept JAF's commendation of the Palo Alto Answering Service.

7f
COST ESTIMATES FOR PROPOSED FACILITIES AND SERVICES 8

MONTHLY	INSTALL	DESCRIPTION	
18	30	Three lines	8a
8	30	Four new 6-button phones	8b
24 (?)	120 (?)	Connexion to other phones and call directors	8c
100	0	Enterprise listings	8d
20	100	Conferencing Facilities	8e
14	60	Card dialers	8f
9	10	Connexion of two lines to Ans. Service	8g
60	0	Palo Alto Answering Service (Might cost less)	8h
---	---		8i
244	340	TOTAL FIXED COSTS	8j

8k
GENERAL 9
I enquired about the following special kinds of services: 9a
Possibilities for hooking into an SRI WATS line 9a|
Various categories of leased lines 9a2
Enterprise exchanges 9a3
Wide Area Telephone Service (WATS) 9a4
Contracts for WATS service are very restrictive, and there does not seem to be any possibility of using a line already in SRI's service. 9b
Leased lines are prohibitively expensive: e.g. a leased line to Cambridge would cost in excess of $6000/mo. 9c
Enterprise exchanges exist merely for the purpose of simplifying bookkeeping -- i.e., they would allow remote sites to call us at our expense without having to place the calls collect. However, calls do have to be placed through an operator, which makes the call more expensive than dialing directly. 9d
ENTERPRISE SERVICE 10
The cost of enterprise service is (approximately) $5.00/mo. per exchange (area in which local calls can be made without a toll charge). Since we would need to reach 3 exchanges in California and about 7 out of state, enterprise service would cost us about $50/mo. plus the cost of the individual calls made to us at the premium rate for operator-assist calls. 10a
Within California, there is no premium; outside Calif., the premium is about 23% of the message charge.

|0a|
WATS SERVICE ||

WATS service permits non-operator assist access to a specified area for either incoming or outgoing calls, but not both on the same line -- i.e., for complete in/out service you have to pay for two lines. ||a

The service is provided for various areas, which extend (by states) in more-or-less concentric circles from here. New England (incl. N.Y, N.J, Penn. Md. and D.C.) is in the most distant area, and to get WATS service to there you must pay for service to all the intervening areas as well. Therefore, if we got WATS service, we would be paying the maximum rate which covers all the states except for Hawaii, Alaska, and CALIFORNIA. ||b

NOTE: We can get itemized listings of all calls charged to a WATS line, but only if we specifically request it. ||b|

All WATS services treat interstate and intrastate calls independently, and full continental U.S. coverage involves area six interstate WATS plus north&south California intrastate WATS. ||c

Interstate WATS is available in either full-time or measured service. ||d

Full-time area six interstate WATS costs $|900/mo per line with two lines being required to provide both incoming and outgoing service. ||d|

All prices in this memo are PLUS |0% federal excise tax. ||d|a

In addition, installation of every line and every piece of equipment involves a one-time installation charge. For WATS lines, the installation charge is $|0 per line. ||d|b

Measured service area six interstate WATS costs $3|5/mo/line for the first |0 hours of connect-time per month plus $23.60/hour(or fraction) for additional connect-time. ||d2

California intrastate WATS is available ONLY in measured service and can be obtained either for Northern Calif. or for the entire state. ||e

Northern Calif. intrastate WATS costs $650/mo. for the first |25 hours of connect time. ||e|

Combined Calif. intrastate WATS costs $900/mo. for the first |25 hours of connect time or $330/mo. for |5 hours plus $|9/hr. overtime. ||e2

Thus, the MINIMUM cost for full continental US WATS coverage for both incoming and outgoing calls would be 2*$3|5 + 2*$330 = $|290 with the possibility that the charges for overtime could be extensive. ||f

NOTE: WATS must be paid for month-by-month in advance, and initial payment is for one full month plus pro-rata for remainder of current month (billing to the 2nd of each month).

||f|

REGULAR LONG-DISTANCE RATES |2

Key |2a

= number of sites in this approximate rate area |2a|

rate = charge for first 3 minutes + charge per minute over 3 |2a2

(not including 10% federal excise tax) |2a2a

The following are regular day station non-operator-assist (direct-dialed) rates for calls from Menlo Park (i.e., the lowest daytime rates available). |2b

|2c

#	City	Rate	Sites	
6	Boston	1.35 + 0.45	BBN MAC HAR BTL LIN CARN	|2e
2	Chicago	1.25 + 0.40	UILL CASE	|2f
1	Salt Lake City	0.95 + 0.30	UTAH	|2g
3	Santa Monica	1.10 + 0.35	UCLA RAND SDC	|2h
1	Santa Barbara	1.00 + 0.30	UCSB	|2i
9	Interstate mean	1.20 + 0.42	Five-minute call = 2.12	|2j
4	S. Calif. mean	1.08 + 0.34	Five-minute call = 1.76	|2k

|2d

|2l

The following are regular day station operator-assist rates for calls from Menlo Park (i.e., the rates which would apply to enterprise calls). |2m

|2n

#	City	Rate	Sites	
6	Boston	1.70 + 0.65	BBN MAC HAR BTL LIN CARN	|2p
2	Chicago	1.55 + 0.65	UILL CASE	|2q
1	Salt Lake City	1.20 + 0.30	UTAH	|2r
3	Santa Monica	1.10 + 0.35	UCLA RAND SDC	|2s
1	Santa Barbara	1.00 + 0.30	UCSB	|2t
9	Interstate mean	1.61 + 0.57	Five-minute call = 2.75	|2u
4	S. Calif. mean	1.08 + 0.34	Five-minute call = 1.76	

|2v

COMPARISON OF RATES (FOR INCOMING CALLS) |3

INTERSTATE WATS - LONG DISTANCE (COLLECT) |3a

FULL SERVICE |3a|

Break-even point (assuming 5-minute average per call) |3a|a

= 1900/2.75 = 690.9| calls per month |3a|a|

= 690.9|/9 = 76.77 calls per site per month |3a|a2

MEASURED SERVICE |3a2

Minimum cost per 5-minute call = 5 * 315/10*60 = 2.63 |3a2a

Break-even point (assuming 5-minute average per call) |3a2b

= 315/2.75 = 114.55 calls per month |3a2b|

= 114.55/9 = 12.73 calls per site per month |3a2b2

INTERSTATE WATS - ENTERPRISE |3b

Calculations same as above except for $35/mo. for enterprise listings. |3b|

FULL SERVICE |3b2

Break-even point (assuming 5-minute average per call) |3b2a

= 1900-35/2.75 = 678.18 calls per month |3b2a|

= 678.18/9 = 75.35 calls per site per month |3b2a2

MEASURED SERVICE |3b3

Minimum cost per 5-minute call = 5 * 315/10*60 = 2.63 |3b3a

Break-even point (assuming 5-minute average per call) |3b3b

= 315-35/2.75 = 101.82 calls per month |3b3b|

= 101.82/9 = 11.31 calls per site per month

|3b3b2

INTRASTATE WATS - LONG DISTANCE (COLLECT) |3c

MEASURED SERVICE (15 HOURS) |3c|

Minimum cost per 5-minute call = 5 * 330/15*60 = 1.83 |3c|a

Minimum cost per additional 5-minute call = 5 * 19/60 = 1.58 |3c|b

Cost comparisons 12 19 1.76 21.12 |3c|c

#calls	WATS	LD (at 1.76/call)	
180	330.00	316.80	|3c|c1
192	349.00	337.92	|3c|c2
204	368.00	359.04	|3c|c3
216	387.00	380.16	|3c|c4
228	406.00	401.28	|3c|c5
240	425.00	422.40	|3c|c6
252	444.00	443.52	|3c|c7
264	463.00	464.64	|3c|c8
276	482.00	485.76	|3c|c9
288	501.00	506.88	|3c|c10

Break even point = 255 calls per month = 64 calls per site per month |3c|c11

|3c|d

ANSWERING SERVICE |4

AUTOMATIC ANSWERING SERVICES |4a

There are 2 kind of automatic answering devices available from the phone company. Both operate on the principle that the device will be turned on manually to provide automatic answering during an interval until they are turned off manually. The phone co. does not have any device which will answer at any time that the phone has not been answered manually (say within five rings). However, the more expensive device has a speaker attached which can be turned loud enough to permit personnel in the area to hear the recording being played to a caller and to intervene to answer the phone manually at any time. |4a|

The least expensive automatic answering service can be provided for any incoming line at the cost of $|3.50/mo (plus $35 installation charge). This device will answer the phone (when turned on) and play a pre-recorded message to the caller. It will not record the caller's reply, and to obtain this feature, we would have to go to the more expensive service which costs $25/mo. per line (+ $35). |4a2

COMMERCIAL ANSWERING SERVICES |4b

I have checked with three local commercial answering services. All of them can provide 24-hour answering of our phone(s). For each call they intercept (if not answered on-site) they will either take a message or try to transfer the call to any specified alternate numbers (e.g., staff members' home phones); they handle the transfer at no cost to the calling party -- if there is any cost incurred in transfering the call (e.g., toll charges), the answering service absorbs it and would bill us at the end of the month. |4b|

It costs $4.50/mo. per line (+ $5 per line) to have extensions of our phones connected to an answering service, and it takes about 3 days to have this done. |4b2

Central Exchange Answering Service |4b3

Address: not listed |4b3a

Business phone: 941-2500 |4b3b

Contact: Mrs. Lynch |4b3c

Rates |4b3d

$30/mo. for first line for first 50 calls/mo. |4b3d|

$|0/mo. for additional lines with no free calls |4b3d2

+ .|0/call for additional calls |4b3d3

Comments: Noisy environment in the operators' work room. |4b3e

Tel-Page Answering Service |4b4

Address: |260 Marshall, Palo Alto |4b4a

Business phone: 326-2011 1b4b
Contact: I forgot to ask 1b4c
Rates 1b4d
$24.25/mo. for first line for 70 calls on which they take written messages (or, I assume, make a transfer connection) 1b4d1
$13/mo. for additional lines including 70 messages per line 1b4d2
+ .15/call for additional calls 1b4d3
Comments: 1b4e
Palo Alto Answering Service 1b5
Address: 445 Sherman Ave., Palo Alto 1b5a
Business phone: 321-4412 1b5b
Contact: Mrs. Stuart 1b5c
Rates 1b5d
$30/mo. for each line for any number of calls 1b5d1
If we have two incoming lines, one for California and one for the rest of the world, the charge will be $40/mo. 1b5d2
Comments: JAF recommends this company very highly.

14b5e

MISCESSANEOUS SERVICES 15

"AUTOMATIC" DIALING 15a

The phone co. provides two kind of devices which attach to a user's telephone to assist in dialing. 15a1

A card dialer, which accepts a small plastic card pre-punched with the number to be called, can be obtained for $3.50/mo. (+ $15). This device can conveniently be used when up to about 50-100 numbers have to be kept handy. 15a1a

A mag-tape driven dialer ("Magicall") costs $8/mo. (+$35), and is a better choice when several hundred numbers have to be easily reached. 15a1b

PUSH-BUTTON PHONES 15b

Six-button phones for handling multiple lines cost $2/mo. (+ $15). They are available in a few colors besides black at an extra one-time charge of $5/phone. (Red, yellow, and a few other colors are available on on special order with unpredictable delivery times). I think that "hold" and "conferencing" facilities on a phone reduce the number of lines which can be handled. 15b1

CONFERENCING 15c

A push-button phone can be equipped with a button to permit "conferencing" at a cost of $2/mo. (+ $10). This allows you to dial one more phone into a conversation already taking place. 15c1

Multiple-phone conference calls normally have to be set up by an operator. However, this is cheaper than phoning all the parties separately (for same length of conversation with each). 15c2

CONVERSATION RECORDING 15d

The phone co. does not offer conversation recording devices. However, they will instal a box which permits connection of the phone to a recorder and which emits the legally-required "beep" to indicate that a conversation is being recorded. The cost of this device is $2/mo. (+ $5). I have not yet been able to determine whether this cost is per phone or per line coming into the phone; I also cannot find out just how one connects a tape recorder to the device, as the phone co. is being obstinate about releasing (or discovering for itself) the answer to this question. 15d1

```
':4877', 09/25/70 0840:35 MEJ ;  :NIC PHONE MEMO, 09/24/70 1311:44 WLB ;
.HED="                                                  4877 WLB 24SEP70
NIC INTERNAL MEMO ON TELEPHONE SERVICE                  09/24/70 1310:51
WLB"; .PST=1; .SCR=1; .PGN=0;
.SNF=72;.MCH=65;.PGN=0;.DSN=1;.DPR=0;
```