USER INTERFACE DESIGN ISSUES FOR A LARGE INTERACTIVE SYSTEM

(J27171) 31-DEC-75 16:54;;;; Title: Author(s): Richard W. Watson/RWW; Distribution: /SRI-ARC([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: RWW; Origin: < WATSON, USER,NLS;7, >, 31-DEC-75 12:50 RWW ;;;; 27171
(This is Chapter II from the '74 Final Report. It was copied from the following: < ARCDOCUMENTATION, FINALCOM,NLS;31, >, 5-DEC-75 15:30 BEV)

.#### ;

Query/help software and data bases

Knowledge Workshop Development

Final Report RADC-TR-75-304

Augmentation Research Center

Stanford Research Institute

Menlo Park, California June 1974

[11] M C GRIGNETTI C HAUSMANN L GOULD

An "intelligent" online assistant and tutor -- NLS scholar

AFIPS Conference Proceedings 1975 NCC

pp 775-781

[12] C H IRBY

The command meta language system

AFIPS Conference Proceedings 1976 NCC Vol 45

[13] D BOBROW et al

TENEX, a paged time sharing system for the PDP-10

Commun. ACM Vol 15, pp 135-143, March 1972

[14] W TEITELMAN et al

INTERLISP reference manual

Bolt Beranek and Newman Inc. and, Xerox Corp. 1974

27171 Distribution

N. Dean Meyer, James E. (Jim) White, Douglas C. Engelbart, Martin E. Hardy, J. D. Hopper, Charles H. Irby, Harvey G. Lehtman, James C. Norton, Jeffrey C. Peters, Dirk H. Van Nouhuys, Kenneth E. (Ken) Victor, Richard W. Watson, Don I. Andrews, Sandra Lee Miranda, Bonny Penny Mosher, Israel A. Torres, Jan H. Kremers, Susan K. Ocken, Raphael Rom, David C. Smith, Buddie J. Pine, Andy Poggio, David L. Retz, Laura J. Metzger, Karolyn J. Martin, Jan A. Cornish, Larry L. Garlick, Priscilla A. Wold, Pamela K. Allen, Delorse M. Brooks, Beverly Boli, Rita Hysmith, Log Augmentation, Raymond R. Panko, Susan Gail Roetter, Robert Louis Belleville, Ann Weinberg, Adrian C. McGinnis, Robert S. Ratner, David S. Maynard, Robert N. Lieberman, James H. Bair, Jeanne M. Leavitt, Rodney A. Bondurant, Jeanne M. Beck, Marcia L. Keeney, Elizabeth K. Michael, Jonathan B. Postel, Elizabeth J. Feinler, Kirk E. Kelley

New Programmer's Documentation Standards

This is a new version of a previous document (HGL/KEV's <15934,>). Of special interest is the documentation for the new procedure template. The template itself resides in <nine, proctemplate.nls, 1>.

INTRODUCTION 1

This document addresses the problem of commenting individual NLS source code files and offers suggested formats for various L10 language constructs; it also provides some suggested standards for error handling and source code file arrangement. There will be other documents to address the problems of extracting information from files, providing global documentation, etc. 1a

While these templates should serve as a starting point for code written by all NLS programmers, it is clear that a certain amount of flexibility is essential. Only the procedure code template is mandatory at this point in time. 1b

There are instances in which the use of a form for the IF statement different than that given here would be desirable. All we ask is that the spirit be preserved. 1b1

This spirit would be hard to define exactly, but among its essentials would be the use of BEGIN and END as "brackets" around blocks of compound statement code, the use of CASE rather than IFs where possible, an attempt to keep "important" logical information at the highest possible level (which results in the suggested form for IFs and the various iterative sttatements), and a general attitude of writing code not for yourself and the computer, but for your brother and sister programmers who must eventually deal with your bugs two years from now. 1b2

The guiding philosophy in selecting the templates has been to find a consistant approach to make listings and/or portrayals of programs easy to read and to have the listing/portrayal of the program follow, to some extent, the logic of the program. 1c

Each logical group of statements is a plex, whose up is a comment. The comment should have some significance when viewed with levels truncated. 1c1

To set off code in L10, one should make use of level indenting where appropriate. Procedures should be thought of as paragraphs; the density of information and the layout in a portrayal can make understanding easy or difficult. Separating to paragraph/procedures should be a blank (created by an EOL). (See below for further discussion of comments.) 1c2

An NLS System Programmers Handbook containing pertinent Journal documents dealing with our system programming tools and conventions and eventually documenting the internal workings of our system should also be prepared. 1d

USE OF COMMENTS 2

It is obvious that comments should be: 2a

meaningful 2a1

clearly visible and distinguisable from code 2a2

used profusely and freely throughout the code 2a3

Code that is not obvious to a programmer relatively naive about that part of the system should be commented. 2b

It is, however, not neccessary to comment code understandable to any system programmer such as an assignment statement:

```
a _ x ;
```

with the comment:

```
% assign x to a %
```

2b1

We prefer that comments refer to a logical group of statements. Logical groups of statements should all be one plex whose up is a comment. 2c

It should be entirely possible to read only the comments and to know what a procedure does and how it does it. 2c1

A bug should not be considered fixed until the comments are brought up to date; similarly a new procedure is not finished, and should not be adopted, until it is fully commented and documented. 2c2

Within a procedure, all level two statements relative to the procedure declaration should be comments, and all code will be substructure of these comments. 2c3

For deeper levels, it is permissible to have comments and the code to which the comments refer, at the same level to conserve levels. 2c3a

However, in this case the group of comments and code may be surrounded on both sides by an optional blank statement or comment (to make the group stand out). 2c3a1

Where it is desirable to comment an individual statement within a logical group, it is permissible to place the comment on the same line as the statement and to the right of the statement. 2d

It is preferable to have all comments of this nature start in the same column (use spaces). 2d1

This makes the reading of a program easier in that there is a clear distinction between comments and code. 2d2

Within a source code file, related procedures should be grouped as a plex under a comment heading. The first branch in such a plex coud be used for explanitory comments if desirable. File wide documentation should appear at the beginning of a file. 2e

DON'T COMMENT BAD CODE, REWRITE IT. 2f

USE OF SPACES 3

Spaces should be used freely to make the reading of a program easier. 3a

They should not be used in places in which their presence obscures a natural binding (e.g., nested procedure calls within the parameter list of anothe proceudre calls; too many spaces between the parentheses could make it difficult to scan which parameters belong to which argument list.) 3a1

The use of punctuation could follow the generally accepted rules for the same puntuation in English. (Thus no spaces before periods or semi-colons or commas, etc.) 3b

The following rules should be obeyed (the rules may be broken if the use of a space clouds an issue or if the use of a space(s) would force a one line statement to take two lines). Spaces should appear: 3c

Between arguments in a parameter list. (This space should follow the comma and precede the succeeding argument.) 3c1

Before and after the "_" or ":=" in assignments. 3c2

Following the colons in the conditional expressions in CASE and ON SIGNAL statements. 3c3

Preceding and following the opening percent-sign of a comment. 3c4

Wherever else the presence of a space serves to clarify or to satisfy your aesthetic sense. 3c5

Parantheses should also be used to form groupings within expressions. 3d

BLOCKS-- BEGIN, END 4

A compound statement (a group delimited by the BEGIN and END; statements) should appear down from the statement of which it is the object. (See examples below.) BEGIN and END should be on the same level. 4a

GOTO STATEMENTS AND LABELS 5

GOTO 5a

GOTO should be avoided and should only be used to preserve structure when unusual transfers of control are used (e.g. an interrupt handler). 5a1

LABELS 5b

Labels should be avoided. They may be used, though, to provide a location to be reached by a goto or for return of control from a catchphrase. 5b1

CASE STATEMENTS 6

```
CASE case-clause OF                                    6a
  cond-exp1: simple-statement;                         6a1
  cond-exp2:                                           6a2
     BEGIN                                             6a2a
     ...                                               6a2b
     END;                                              6a2c
  cond-exp3: simple-statement;                         6a3
  ...                                                  6a4
  cond-expn: simple-statement;                         6a5
  ENDCASE simple-statement;                            6a6
  --OR--                                               6a7
  ENDCASE                                              6a8
     BEGIN                                             6a8a
     ...                                               6a8b
     END;                                              6a8c
```

IF STATEMENTS 7

Please note that very often complicated nested IFs become simpler if rewritten as CASE statements. 7a

FORM 1: entire statement fits on one line 7b

```
IF if-clause THEN then-clause [ELSE else-clause];                7b1
```

FORM 2: entire statement does not fit on one line 7c

FORM 2A: simple then-clause and simple else-clause 7c1

```
IF if-clause THEN then-clause                                    7c1a
ELSE else-clause;                                                7c1b
```

FORM 2B: complex then-clause and complex else-clause 7c2

```
IF if-clause THEN % Comment %                                    7c2a
   BEGIN                                                         7c2a1
   ...                                                           7c2a2
   END                                                           7c2a3
ELSE % Comment %                                                 7c2b
   BEGIN                                                         7c2b1
   ...                                                           7c2b2
   END;                                                          7c2b3
```

FORM 2C: simple then-clause and complex else-clause 7c3

```
IF if-clause THEN then-clause                                    7c3a
ELSE                                                             7c3b
   BEGIN                                                         7c3b1
   ...                                                           7c3b2
   END;                                                          7c3b3
```

FORM 2D: complex then-clause and simple else-clause 7c4

```
IF if-clause THEN                                                7c4a
   BEGIN                                                         7c4a1
   ...                                                           7c4a2
   END                                                           7c4a3
ELSE else-clause;                                                7c4b
```

ITERATIVE STATEMENTS 8

UNTIL STATEMENTS 8a

FORM 1: entire statement fits on one line 8a1

```
UNTIL until-clause DO simple-statment;                    8a1a
```

FORM 2: entire statement doesn't fit on one line 8a2

```
UNTIL until-clause DO                                     8a2a
   simple-statement;                                      8a2a1

--OR--                                                    8a2b

UNTIL until-clause DO                                     8a2c
   BEGIN                                                  8a2c1
   ...                                                    8a2c2
   END;                                                   8a2c3
```

WHILE STATEMENTS 8b

FORM 1: entire statement fits on one line 8b1

```
WHILE while-clause DO simple-statment;                    8b1a
```

FORM 2: entire statement doesn't fit on one line 8b2

```
WHILE while-clause DO                                     8b2a
   simple-statement;                                      8b2a1

--OR--                                                    8b2b

WHILE while-clause DO                                     8b2c
   BEGIN                                                  8b2c1
   ...                                                    8b2c2
   END;                                                   8b2c3
```

DO STATEMENTS 8c

FORM 1: entire statement fits on one line 8c1

```
DO do-clause WHILE/UNTIL while/until-clause;              8c1a
```

FORM 2: entire statement doesn't fit on one line 8c2

```
DO do-clause                                              8c2a
WHILE/UNTIL while/until-clause;                           8c2b
```

```
      --OR--                                                    8c2c

      DO                                                        8c2d
         BEGIN                                                  8c2d1
         ...                                                    8c2d2
         END                                                    8c2d3
      WHILE/UNTIL while/until-clause;                           8c2e

FOR STATEMENTS                                                  8d

   FORM 1: entire statement fits on one line                    8d1

      FOR for-clause DO simple-statment;                        8d1a

   FORM 2: entire statement doesn't fit on one line             8d2

      FOR for-clause DO                                         8d2a
         simple-statement;                                      8d2a1

      --OR--                                                    8d2b

      FOR for-clause DO                                         8d2c
         BEGIN                                                  8d2c1
         ...                                                    8d2c2
         END;                                                   8d2c3

LOOP STATEMENTS                                                 8e

   FORM 1: entire statement fits on one line                    8e1

      LOOP loop-clause;                                         8e1a

   FORM 2: entire statement does not fit on one line            8e2

      LOOP                                                      8e2a
         BEGIN                                                  8e2a1
         ...                                                    8e2a2
         END;                                                   8e2a3
```

FIND SIAIEMENTS 9

Comments are definitely in order here; also, because of the properties of FIND statements, it is more often than not advisable to have them as FIND expressions in IF statements with appropriate error traps: if there is a failure, pointers may not be what you would expect otherwise. 9a

ERROR HANDLING 10

Errors should be handled by signals (ABORT/ HELP/ NOTE), calls on appropriate error routines such as err, or by returning a well defined value. 10a

All procedures that generate or catch signals should have the appropriate documentation (see PROCEDURE CODE, NON-STANDARD CONTROL). 10a1

Note that the use of signals with uniquely defined SIGNAL values permits the passing of more information and control in error situations than a call on err. 10a2

When a signal is caught, care should be taken that the action appropriate for the signaltype is chosen. Each catchphrase should be able to respond to an ABORT, HELP, or NOTE correctly. To aid in handling this problem, a template for catchphrases is suggested. 10a3

CATCHPHRASE STATEMENTS 10b

Catchphrases should be located at the end of the procedure body under a second level comment just prior to the "END." statement. 10b1

```
(catchname) CATCHPHRASE ( : res2, res3, res4);          10b2
   BEGIN                                                10b2a
   CASE SIGNALTYPE OF                                   10b2b
      = aborttype :                                     10b2b1
         simple-statement;                              10b2b1a
      = notetype :                                      10b2b2
         simple-statement;                              10b2b2a
      = helptype :                                      10b2b3
         simple-statement;                              10b2b3a
      ENDCASE                                           10b2b4
         CONTINUE;                                      10b2b4a
   END;                                                 10b2c
```

PROCEDURE CALLS 11

Should not be links. Rather, a "catalog" file of links (produced by a user program making use of sysgd) can be accessed via Jump-to-Name-External. 11a

PROCEDURE CODE 12

All PROCEDURE statements should be at the same level. Procedures should be organized together functionally in a surce code file down a level from an appropriate funtional definition comment. (e.g.,, %.....Parsing Procedures.....%, %..... Error Routines.....%, etc.) 12a

Each procedure should be classed by its potential for use as a black box (e.g. as a procedure called by user-programs). The following class codes should be used to characterize a procedure: 12b

G = Guaranteed to stay around with the same user interface for some time to come. 12b1

U = Useful routine but not guaranteed to stay around from version to version. 12b2

L = Lower level routine that should only be used when higher level routines will not suffice. Absence of "L" implies a higher level routine. 12b3

B = NLS backend routine that is callable by any user program. Absence of "B" implies that the routine is not loaded with NLS. 12b4

^L, contol-L or formfeed, could be used in the code to produce well-paginated listings. There should be a ^L before the funtional definition comments and before long procedures. 12c

This will make smaller blocks of information start on a new page when listed via OUTPUT QUICKPRINT. Thus, a branch-only quickprint of a changed part of a source code file may easily be inserted into our listings. 12c1

An EOL should appear after the "END.". 12d

Example: 12e

```
% Blapping routines %<EOL>                                          12e1

    % General comment about blap routines.  May be a branch.
    %<EOL>                                                         12e1a

    (procname)  % CL:prog-class;  one-line-description % <EOL>
    PROCEDURE (FP1 <type>, ..., FPm % => [meta-res] RES1 <type>,
    ..., RESn %);                                                  12e1b
       % Procedure description                                     12e1b1
          FUNCTION                                                 12e1b1a
             detailed description of what the procedure is
```

```
      supposed to do and the algorithms used (where
      appropriate)                                          12e1b1a1
   ARGUMENTS                                                 12e1b1b
      FP1 -- type - <special entity type> - description
      of value                                              12e1b1b1
      ...                                                   12e1b1b2
      FPm -- type - <special entity type> - description
      of value                                              12e1b1b3
                                                            12e1b1b4
         Example:                                           12e1b1b4a
            src -- LIST - <LSEL> - list of 3 elements
            (Entity type INDEX, user typein LITERAL,
            window-id INTEGER).                             12e1b1b4a1
            userident -- STRING - addr of string
            containing ident; 1-5 chars.                    12e1b1b4a2
   RESULTS                                                   12e1b1c
      [meta-res] -- description of value                    12e1b1c1
      RES1 (proc value) -- type - description of value      12e1b1c2
                                                            12e1b1c3
      ...                                                   12e1b1c4
      RESn -- type - description of value                   12e1b1d
   NON-STANDARD CONTROL                                     12e1b1d1
      SIGNALS GENERATED                                     12e1b1d1a
         Descriptions of errors generated, as well as
         help signals, notes, etc.  Example:                12e1b1d1a1
            SIGNALS GENERATED
               If the argument 'foo' is not a list, the
               error signal                                 12e1b1d1a1a
                  ABORT(cantdothat, "You've got to pass a
                  list, dummy!")                            12e1b1d1a1a1
               is generated.                                12e1b1d1a1b
      SIGNALS CAUGHT                                        12e1b1d2
         Descriptions of signals that are stopped and/or
         modified by invoked catchphrases.  Signals that
         are CONTINUEd need not be mentioned.  If a
         catchphrase generates a new signal, the new
         signal should be described in the SIGNALS
         GENERATED entry.  Example:                         12e1b1d2a
            SIGNALS CAUGHT                                  12e1b1d2a1
               This procedure stops all error signals
               (all signals of type "abort").               12e1b1d2a1a
      OS INTERFACE                                          12e1b1d3
         A description of all unusual transfers of
         control, interrupt routines, context switches,
         stack modifications, etc.                          12e1b1d3a
   GLOBALS                                                  12e1b1e
      Globals read -- type                                  12e1b1e1
      Globals modified -- type - description of value       12e1b1e2
   %                                                        12e1b1f
% Declarations %                                           12e1b2
```

```
            (recommended ordering)                                    12e1b2a
            LOCAL                                                     12e1b2b
            LOCAL TEXT POINTER                                        12e1b2c
            LOCAL STRING                                              12e1b2d
            LOCAL LIST                                                12e1b2e
            REF                                                       12e1b2f
         <body of the procedure>  comments only at this level         12e1b3
         END.<EOL>
                                                                      12e1b4
Programmer's Template  ( ref. <nine, proctemplate,> )                    12f
   (procname)   % CL: ;  one-line-description %
   PROCEDURE (arg1 <type>, ..., argn % => [meta-res] res1 <type>,
   ..., resn %);                                                      12f1
      % Procedure description                                         12f1a
         FUNCTION                                                    12f1a1
            none                                                    12f1a1a
         ARGUMENTS                                                   12f1a2
            none                                                    12f1a2a
         RESULTS                                                     12f1a3
            none                                                    12f1a3a
         NON-STANDARD CONTROL                                        12f1a4
            none                                                    12f1a4a
         GLOBALS                                                     12f1a5
            none                                                    12f1a5a
         %                                                           12f1a6
      % Declarations %                                               12f1b
      <procedure body>                                               12f1c
      END.
                                                                     12f1d
   Example:                                                             12f2
      (xinit)  % CL:GB;  Init. procedure called in FE initrule %
      PROCEDURE (rtnlist REF % => see arg %);                        12f2a
         % Procedure description                                    12f2a1
            FUNCTION                                               12f2a1a
               This procedure is called during the initialization
               rule of the NLS grammar.  The FE will restart a
               failing init rule, so this procedure always returns
               successfully to the FE and returns the real outcome
               as a result.  The result is tested in the grammar
               and if it indicates failure, a parsefunction to
               terminate the tool is called.                    12f2a1a1
            ARGUMENTS                                           12f2a1b
               rtnlist -- REF - addr of result list (for middle
               end).                                            12f2a1b1
            RESULTS                                             12f2a1c
               proc value -- TRUE always.                       12f2a1c1
               rtnlist -- REF - addr of result list; LIST (
               outcome BOOLEAN )                                12f2a1c2
            NON-STANDARD CONTROL                                12f2a1d
```

```
        SIGNALS GENERATED                                          12f2a1d1
           A call is made to err if procedure feinfo fails.
                                                                   12f2a1d1a
        SIGNALS CAUGHT                                             12f2a1d2
           All aborts are terminated.  The catchphrase
           initfail returns a failure result to the FE,
           which calls a parsefunction to terminate the
           tool.                                                   12f2a1d2a
           All other signals are continued.                        12f2a1d2b
     GLOBALS                                                       12f2a1e
        MODIFIED                                                   12f2a1e1
           dacnt -- no. of active da's; initialized to 0.
                                                                   12f2a1e1a
        READ                                                       12f2a1e2
           uboolean                                                12f2a1e2a
     %                                                             12f2a1f
% invoke catchphrase %                                            12f2a2
    INVOKE(initfail, rtnsetfail);                                  12f2a2a
% initialize user options (profile) %                             12f2a3
    uoinit();                                                      12f2a3a
% get frontend information %                                      12f2a4
    IF NOT feinfo() THEN err($"abort back to FE");                 12f2a4a
% display area initialization%                                    12f2a5
    dacnt _ 0;                                                     12f2a5a
    intdafl (&tda _ newda());                                      12f2a5b
% open initials file %                                           12f2a6
    initid();                                                      12f2a6a
% drop catchphrase %                                             12f2a7
    DROP(initfail);                                                12f2a7a
% return %                                                       12f2a8
    #rtnlist#[1] _  USE makedesc(uboolean,TRUE, FALSE);            12f2a8a
    (rtnsetfail):                                                  12f2a8b
        RETURN;                                                    12f2a8b1
% catchphrases %                                                 12f2a9
    (initfail) CATCHPHRASE;                                        12f2a9a
        BEGIN                                                      12f2a9a1
        CASE SIGNALTYPE OF                                         12f2a9a2
           =aborttype:                                             12f2a9a2a
              BEGIN                                                12f2a9a2a1
              #rtnlist#[1] _ USE makedesc(uboolean, FALSE,
              FALSE);                                              12f2a9a2a2
              TERMINATE;                                           12f2a9a2a3
              END;                                                 12f2a9a2a4
           ENDCASE                                                 12f2a9a2b
              CONTINUE;                                            12f2a9a2b1
        END;                                                       12f2a9a3
END.
                                                                   12f2a10
```

```
(dismes)   % CL:GB ;  display a message to the user %
PROCEDURE (type, astrng REF % => %);                                    12f2b
   % Procedure description                                              12f2b1
      FUNCTION                                                          12f2b1a
         This routine is called to display a message on the
         screen or tty.  It performs a PCP call to the
         Frontend "show" function to display the message.
         It will break the string into substrings, at line
         boundries, if the size exceeds the maximum
         allowable by the FE.                                           12f2b1a1
      ARGUMENTS                                                         12f2b1b
         type -- INTEGER - value determines the display
         action                                                         12f2b1b1
            type = 0:                                                   12f2b1b1a
               will remove any message on the screen. An
               A-string need not be given in astrng in this
               case.                                                    12f2b1b1a1
            type > 0:                                                   12f2b1b1b
               the message will be displayed, and the
               routine will return (with the message still
               on the screen).                                          12f2b1b1b1
         astrng -- REF - addr of string to be displayed, if
         type is non-zero.                                              12f2b1b2
      RESULTS                                                           12f2b1c
         none                                                           12f2b1c1
      NON-STANDARD CONTROL                                              12f2b1d
         SIGNALS GENERATED                                              12f2b1d1
            Calls err(s"string too long to display") if
            astrng.L exceeds FE maximum and no end-of-line
            characters exist.                                           12f2b1d1a
      GLOBALS                                                           12f2b1e
         READ                                                           12f2b1e1
            maxdpschar -- max. number of char. FE can
            display.                                                    12f2b1e1a
      %                                                                 12f2b1f
   % Declarations %                                                     12f2b2
      LOCAL                                                             12f2b2a
         wrkstr REF,  % addr of current substring %                     12f2b2a1
         chsent = 0,  % characters sent %                               12f2b2a2
         notdone = TRUE;                                                12f2b2a3
      LOCAL TEXT POINTER                                                12f2b2b
         tps, tpe;                                                      12f2b2b1
      LOCAL LIST                                                        12f2b2c
         showargs[3], resl[1];                                          12f2b2c1
   % package and send the string, within FE constraints %               12f2b3
      &wrkstr _ &astrng;                                                12f2b3a
      WHILE notdone DO                                                  12f2b3b
         BEGIN                                                          12f2b3b1
         IF wrkstr.L >= maxdpschar THEN                                 12f2b3b2
```

```
            BEGIN  % move string header and send next
            substring %                                          12f2b3b2a
            % setup max allowable string %                       12f2b3b2b
               wrkstr.L _ maxdpschar;                            12f2b3b2b1
            % scan back past last end-of-line %                  12f2b3b2c
               FIND SF(*wrkstr*) ^tps SE(*wrkstr*) ^tpe;         12f2b3b2c1
               IF NOT FIND [EOL/(CR LF)/(LF CR)] ^tpe THEN
               err($"String too long to display");               12f2b3b2c2
               *wrkstr* _ tps tpe;                               12f2b3b2c3
               chsent _ chsent + wrkstr.L + 1;                   12f2b3b2c4
            END                                                  12f2b3b2d
         ELSE notdone _ FALSE;                                    12f2b3b3
         % send the string %                                      12f2b3b4
            % build argument list %                              12f2b3b4a
               CASE type OF                                      12f2b3b4a1
                  = 0:                                           12f2b3b4a1a
                     #showargs# _ "",                            12f2b3b4a1a1
                        USE makedesc(uboolean, nocawait,
                        FALSE);                                  12f2b3b4a1a1a
                  ENDCASE                                        12f2b3b4a1b
                     #showargs# _ *wrkstr*,                      12f2b3b4a1b1
                        USE makedesc(uboolean, nocawait,
                        FALSE);                                  12f2b3b4a1b1a
               % make remote call %                              12f2b3b4b
               CASE howpcp OF                                    12f2b3b4b1
                  = 2:  %DPS%                                    12f2b3b4b1a
                     BEGIN                                       12f2b3b4b1a1
                     IF NOT fepkh THEN fepkh _ opnpkg(2,
                     $"tool-package");                           12f2b3b4b1a2
                     pcpcall (feprh, fepkh, $"show",
                     $showargs, 0);                              12f2b3b4b1a3
                     END;                                        12f2b3b4b1a4
                  = 1, = 3:  %S-P, MSG%                          12f2b3b4b1b
                     extcall(femailbox[1], $"show", 2
                     %in-line w/ ack%, $showargs, $res1);
                                                                 12f2b3b4b1b1
                  ENDCASE;                                       12f2b3b4b1c
         % move header past last end-of-line %                   12f2b3b5
            IF notdone THEN                                      12f2b3b5a
               BEGIN                                             12f2b3b5a1
               &wrkstr _ &wrkstr + (wrkstr.L + 4) / 5;           12f2b3b5a2
               wrkstr.L _ astrng.L - chsent;                     12f2b3b5a3
               END;                                              12f2b3b5a4
         % null argument list %                                  12f2b3b6
            #showargs# _ ;                                       12f2b3b6a
         END;                                                    12f2b3b7
% return %                                                       12f2b4
   RETURN;                                                       12f2b4a
```

```
END.
```

12f2b5

FILE STATEMENTS 13

Should satisfy the requirements of the lionls/tasks compilation system. 13a

FILE filename % <compiler-dir>compiler-file <rel-dir>rel-file % 13a1

FILE STRUCTURE 14
Branch 1: FILE STATEMENT 14a
Branch 2: Documentation for this file. What procedures in this file have in common, arrangement of procedures and data structures, other relevant general information. 14b
Branch 3: Declarations for this file and comments on unusual global data structures which may be used. 14c
Branch 4 - Branch n: General procedure cluster grouping comments. 14d
See PROCEDURE CODE recommendation above. 14d1
BRANCH n+1: FINISH 14e

MISCELLANEOUS 15

Do not use short cuts which may prove confusing to others. 15a
A notable example is the time and space saving omission of a REF on pointers which have been passed as arguments to one procedure which uses it only as an argument in another procedure call. Even in this case the variable should be REF'd and then unrefed (with an &) in the argument list of the called procedure. 15a1

If you change procedures, please try to make sure your changes affect no one else. In particular, if you change the number of arguments to a procedure, do a search to see where the procedure is called and change those places if necessary. 15b

Use procedure and parameter names which are functionally descriptive wherever possible. 15c

Don't make your procedures too long with particularly complicated nested loops. If possible and desirable, consider breaking the code into several procedures-- it will make your code easier to read and understand and will also avoid the problem of DDT "forgetting" which procedure you are in if it is too long. 15d

Try to confine global declarations to (documented) branches in data files or constant files. 15e

Try to use locals before declaring globals. Avoid the use, where possible, of global strings. 15f

Try to put procedures in appropriate files. 15g
For example, all procedures dealing with file opening and closing should be grouped. 15g1

Try to avoid duplicating code except where essential for efficiency. If you can use a procedure except for a few instructions, generalize the procedure or separate out common code into an additional "core" procedure. 15h

Note all changes in a status file or a branch in (nls,tasks). 15i

A High-Level Framework for Network-Based Resource Sharing

23-DEC-75

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

(415) 326-6200 x2960

This paper proposes a high-level, application-independent protocol and software framework that would extend the local programming environment to embrace modules in other computers within a resource sharing computer network, and thereby facilitate the construction of distributed systems and encourage the sharing of resources.

The work reported here was supported by the Advanced Research Projects Agency of the Department of Defense, and by the Rome Air Development Center of the Air Force.

This paper has been submitted for publication in the Proceedings of the 1976 National Computer Conference.

Future-Net

< JOURNAL, 27197.NLS;1, >, 23-DEC-75 12:37 XXX ;;;; Title:
Author(s): James E. (Jim) White/JEW; Distribution: /DLS([ACTION] If
you would run this through the RADC approval mill as you offered, I
would much appreciate it) SRI-ARC([INFO-ONLY]) ; Sub-Collections:
SRI-ARC; Clerk: JEW; Origin: < JWHITE, DPSPROPER.NLS;2, >,
23-DEC-75 12:52 JEW ;;;; ####;

To be submitted, in essentially this form, for publication in the 76 NCC Proceedings. Throughout the body of the paper, double quotes delimit text to be rendered in italics.

THE GOAL, RESOURCE SHARING 1

The principal goal of all resource-sharing computer networks, including the now international ARPA Network (the ARPANET), is to usefully interconnect geographically distributed hardware, software, and human resources [1]. Achieving this goal requires the design and implementation of various levels of support software within each constituent computer, and the specification of network-wide "protocols" (that is, conventions regarding the format and the relative timing of network messages) governing their interaction. This paper outlines an alternative to the approach that ARPANET system builders have been taking since work in this area began in 1970, and suggests a strategy for modeling distributed systems within any large computer network. 1a

The first section of this paper describes the prevailing ARPANET protocol strategy, which involves specifying a family of application-dependent protocols with a network-wide inter-process communication facility as their common foundation. In the second section, the application-independent command/response discipline that characterizes this protocol family is identified and its isolation as a separate protocol proposed. Such isolation would reduce the work of the applications programmer by allowing the software that implements the protocol to be factored out of each applications program and supplied as a single, installation-maintained module. The final section of this paper proposes an extensible model for this class of network interaction that in itself would even further encourage the use of network resources. 1b

THE CURRENT SOFTWARE APPROACH TO RESOURCE SHARING 2

Function-Oriented Protocols 2a

The current ARPANET software approach to facilitating resource sharing has been detailed elsewhere in the literature [2, 3, 4]. Briefly, it involves defining a Host-Host Protocol by which the operating systems of the various "host" computers cooperate to support a network-wide inter-process communication (IPC) facility, and then various function-oriented protocols by which processes deliver and receive specific services via IPC. Each function-oriented protocol regulates the dialog between a resident "server process" providing the service, and a "user process" seeking the service on behalf of a user (the terms "user" and "user process" will be used consistently throughout this paper to distinguish the human user from the computer process acting on his behalf). 2a1

The current Host-Host Protocol has been in service since 1970. Since its initial design and implementation, a variety of deficiencies have been recognized and several alternative protocols suggested [5, 6]. Although improvements at this level would surely have a positive effect upon Network resource sharing, the present paper simply assumes the existence of some form of IPC and focuses attention upon higher level protocol design issues. 2a2

Each of the function-oriented protocols mentioned in this paper constitutes the official ARPANET protocol for its respective application domain and is therefore implemented at nearly all of the 75 host installations that now comprise the Network. It is primarily upon this widely implemented protocol family (and the philosophy it represents) that the present paper focuses. Needless to say, other important resource sharing tools have also been constructed within the ARPANET. The Resource Sharing Executive (RSEXEC), designed and implemented by Bolt, Beranek and Newman, Inc [7], provides an excellent example of such work. 2a3

Experience with and Limitations of Hands-On Resource Sharing 2b

The oldest and still by far the most heavily used function-oriented protocol is the Telecommunications Network protocol (TELNET) [8], which effectively attaches a terminal on one computer to an interactive time-sharing system on another, and allows a user to interact with the remote system via the terminal as if he were one of its local users. 2b1

As depicted in Figure 1, TELNET specifies the means by which a user process monitoring the user's terminal is interconnected, via an IPC communication channel, with a server process with access to the target time-sharing system. TELNET also legislates a standard character set in which the user's commands and the system's responses are to be represented in transmission between machines. The syntax and semantics of these interchanges, however, vary from one system to another and are unregulated by the protocol; the user and server processes simply shuttle characters between the human user and the target system. 2b2

Although the hands-on use of remote resources that TELNET makes possible is a natural and highly visible form of resource sharing, several limitations severely reduce its long-term utility: 2b3

(1) It forces upon the user all of the trappings of the resource's own system.

To exploit a remote resource, the user must leave the familiar working environment provided by his local system and enter an alien one with its own peculiar system structure (login, logout, and subsystem entry and exit procedures) and command language discipline (command recognition and completion conventions, editing characters, and so on). Hands-on resource sharing thus fails to provide the user with the kind of organized and consistent workshop he requires to work effectively [9].

(2) It provides no basis for bootstrapping new composite resources from existing ones.

Because the network access discipline imposed by each resource is a human-engineered command language, rather than a machine-oriented communication protocol, it is virtually impossible for one resource to programatically draw upon the services of others. Doing so would require that the program deal successfully with complicated echoing and feedback characteristics; unstructured, even unsolicited system responses; and so forth. Hands-on resource sharing thus does nothing to provide an environment in which existing resources can be used as building blocks to construct new, more powerful ones.

These inherent limitations of hands-on resource sharing are removed by a protocol that simplifies and standardizes the dialog between user and server processes. Given such a protocol, the

various remote resources upon which a user might wish to draw can indeed be made to appear as a single, coherent workshop by interposing between him and them a command language interpreter that transforms his commands into the appropriate protocol utterances [10, 11]. The construction of composite resources also becomes feasible, since each resource is accessible by means of a machine-oriented protocol and can thus be readily employed by other processes within the network. 2b4

Standardizing the Inter-Machine Dialog in Specific Application Areas 2c

After the TELNET protocol had been designed and widely implemented within the ARPANET, work began on a family of function-oriented protocols designed for use by programs, rather than human users. Each such protocol standardizes the inter-machine dialog in a particular application area. While TELNET dictates only the manner in which user and server processes are interconnected via the IPC facility, and the character set in which the two processes communicate once connected, each member of this family specifies in addition the syntax and semantics of the commands and responses that comprise their dialog. 2c1

Protocols within this family necessarily differ in substance, each specifying its own application-specific command set. The File Transfer Protocol (FTP) [12], for example, specifies commands for manipulating files, and the Remote Job Entry Protocol (RJE) [13] specifies commands for manipulating batch jobs. Protocols throughout the family are, however, similar in form, each successive family member having simply inherited the physical features of its predecessors. Thus FTP and RJE enforce the same conventions for formulating commands and responses. 2c2

This common command/response discipline requires that commands and responses have the following respective formats: 2c3

```
command-name    <SP> parameter <CRLF>
response-number <SP> text      <CRLF>
```

Each command invoked by the user process is identified by NAME and is allowed a single PARAMETER. Each response generated by the server process contains a three-digit decimal response NUMBER (to be interpreted by the user process) and explanatory TEXT (for presentation, if necessary, to the user). Response numbers are assigned in such a way that, for example, positive and negative acknowledgments can be easily distinguished by the user process. 2c4

FTP contains, among others, the following commands (each listed with one of its possible responses) for retrieving, appending to, replacing, and deleting files, respectively, within the server process' file system: 2c5

Command	Response
RETR <SP> filename <CRLF>	250 <SP> Beginning transfer. <CRLF>
APPE <SP> filename <CRLF>	400 <SP> Not implemented. <CRLF>
STOR <SP> filename <CRLF>	453 <SP> Directory overflow. <CRLF>
DELE <SP> filename <CRLF>	450 <SP> File not found. <CRLF>

The first three commands serve only to initiate the transfer of a file from one machine to another. The transfer itself occurs on a separate IPC channel and is governed by what amounts to a separate protocol. 2c6

Since the general command format admits but a single parameter, multiparameter operations must be implemented as sequences of commands. Thus two commands are required to rename a file: 2c7

Command	Response
RNFR <SP> oldname <CRLF>	200 <SP> Next parameter. <CRLF>
RNTO <SP> newname <CRLF>	253 <SP> File renamed. <CRLF>

A COMMAND/RESPONSE PROTOCOL, THE BASIS FOR AN ALTERNATIVE APPROACH 3

The Importance of Factoring Out the Command/Response Discipline 3a

That FTP, RJE, and the other protocols within this family share a common command/response discipline is a fact not formally recognized within the protocol literature, and each new protocol document describes it in detail, as if for the first time. Nowhere are these conventions codified in isolation from the various contexts in which they find use, being viewed as a necessary but relatively unimportant facet of each function-oriented protocol. "This common command/response discipline has thus gone unrecognized as the important, application-independent protocol that it is." 3a1

This oversight has had two important negative effects upon the growth of resource sharing within the ARPANET: 3a2

(1) It has allowed the command/response discipline to remain crude.

As already noted, operations that require more than a single parameter are consistently implemented as two or more separate commands, each of which requires a response and thus incurs the overhead of a full round-trip network delay. Furthermore, there are no standards for encoding parameter types other than character strings, nor is there provision for returning results in a command response.

(2) It has placed upon the applications programmer the burden of implementing the network "run-time environment (RTE)" that enables him to access remote processes at the desired, functional level.

Before he can address remote processes in terms like the following:

```
execute function DELE with argument TEXTFILE
   on machine X
```

the applications programmer must first construct (as he invariably does in every program he writes) a module that provides the desired program interface while implementing the agreed upon command/response discipline. This run-time environment contains the code required to properly format outgoing commands, to interface with the IPC facility, and to parse incoming responses. Because the system provides only

the IPC facility as a foundation, the applications programmer is deterred from using remote resources by the amount of specialized knowledge and software that must first be acquired.

If, on the other hand, the command/response discipline were formalized as a separate protocol, its use in subsequent function-oriented protocols could rightly be anticipated by the systems programmer, and a single run-time environment constructed for use throughout an installation (in the worst case, one implementation per programming language per machine might be required). This module could then be placed in a library and, as depicted in Figure 2, link loaded with (or otherwise made available to) each new applications program, thereby greatly simplifying its use of remote resources.

Furthermore, since enhancements to it would pay dividends to every applications program employing its services, the run-time environment would gradually be augmented to provide additional new services to the programmer.

The thesis of the present paper is that one of the keys to facilitating network resource sharing lies in (1) isolating as a separate protocol the command/response discipline common to a large class of applications protocols; (2) making this new, application-independent protocol flexible and efficient; and (3) constructing at each installation a RTE that employs it to give the applications programmer easy and high-level access to remote resources. 3a3

Specifications for the Command/Response Protocol 3b

Having argued the value of a command/response protocol (hereafter termed the Protocol) as the foundation for a large class of applications protocols, there remains the task of suggesting the form that the Protocol might take. There are eight requirements. First, it must reproduce the capabilities of the discipline it replaces: 3b1

(1) Permit invocation of arbitrary, named commands (or functions) implemented by the remote process.

(2) Permit command outcomes to be reported in a way that aids both the program invoking the commmand and the user under whose control it may be executing.

Second, the Protocol should remove the known deficiencies of its predecessor, that is: 3b2

(3) Allow an arbitrary number of parameters to be supplied as arguments to a single command.

(4) Provide representations for a variety of parameter types, including but not limited to character strings.

(5) Permit commands to return parameters as results as well as accept them as arguments.

And, finally, the Protocol should provide whatever additional capabilities are required by the more complex distributed systems whose creation the Protocol seeks to encourage. Although others may later be identified, the three capabilities below are recognized now to be important: 3b3

(6) Permit the server process to invoke commands in the user process, that is, eliminate entirely the often inappropriate user/server distinction, and allow each process to invoke commands in the other.

In the workshop environment alluded to earlier, for example, the user process is the command language interpreter and the server process is any of the software tools available to the user. While most commands are issued by the interpreter and addressed to the tool, occasionally the tool must invoke commands in the interpreter or in another tool. A graphical text editor, for example, must invoke commands within the interpreter to update the user's display screen after an editing operation.

(7) Permit a process to accept two or more commands for concurrrent execution.

The text editor may wish to permit the user to initiate a long formatting operation with one command and yet continue to issue additional, shorter commands before there is a response to the first.

(8) Allow the process issuing a command to suppress the response the command would otherwise elicit.

This feature would permit network traffic to be reduced in those cases in which the process invoking the command deems a

response unnecessary. Commands that always succeed but never return results are obvious candidates for this kind of treatment.

A Formulation of the Protocol That Meets These Specifications 3c

The eight requirements listed above are met by a protocol in which the following two messages are defined: 3c1

```
message-type=COMMAND  [tid] command-name arguments
message-type=RESPONSE  tid  outcome      results
```

Here and in subsequent protocol descriptions, elements enclosed in square brackets are optional. 3c2

The first message invokes the command whose NAME is specified using the ARGUMENTS provided. The second is issued in eventual response to the first and returns the OUTCOME and RESULTS of the completed command. Whenever OUTCOME indicates that a command has failed, the command's RESULTS are required to be an error number and diagnostic message, the former to help the invoking program determine what to do next, the latter for possible presentation to the user. The protocol thus provides a framework for reporting errors, while leaving to the applications program the tasks of assigning error numbers and composing the text of error messages. 3c3

There are several elements of the protocol that are absent from the existing command/response discipline: 3c4

(1) RESULTS, in fulfillment of Requirement 5.

(2) A MESSAGE TYPE that distinguishes commands from responses, arising from Requirement 6.

In the existing discipline, this distinction is implicit, since user and server processes receive only responses and commands, respectively.

(3) An optional transaction identifier TID by which a command and its response are associated, arising from Requirements 7 and 8.

The presence of a transaction identifier in a command implies the necessity of a response echoing the identifier; and no two concurrently outstanding commands may bear the same identifier.

Requirements 3 and 4--the ability to transmit an arbitrary number of parameters of various types with each command or response--are most economically and effectively met by defining a small set of primitive "data types" (for example, booleans, integers, character strings) from which concrete parameters can be modeled, and a "transmission format" in which such parameters can be encoded. Appendix A suggests a set of data types suitable for a large class of applications; Appendix B defines some possible transmission formats. 3c5

The protocol description given above is, of course, purely symbolic. Appendix C explores one possible encoding of the protocol in detail. 3c6

Summarizing the Arguments Advanced So Far 3d

The author trusts that little of what has been presented thus far will be considered controversial by the reader. The following principal arguments have been made: 3d1

(1) The more effective forms of resource sharing depend upon remote resources being usefully accessible to other programs, not just to human users.

(2) Application-dependent protocols providing such access using the current approach leave to the applications programmer the task of constructing the additional layer of software (above the IPC facility provided by the system) required to make remote resources accessible at the functional level, thus discouraging their use.

(3) A single, resource-independent protocol providing flexible and efficient access at the functional level to arbitrary remote resources can be devised.

(4) This protocol would make possible the construction at each installation of an application-independent, network run-time environment making remote resources accessible at the functional level and thus encouraging their use by the applications programmer.

A protocol as simple as that suggested here has great potential for stimulating the sharing of resources within a computer network. First, it would reduce the cost of adapting existing resources for network use by eliminating the need for the design, documentation, and implementation of specialized delivery protocols. Second, it

would encourage the use of remote resources by eliminating the need for application-specific interface software. And finally, it would encourage the construction of new resources built expressly for remote access, because of the ease with which they could be offered and used within the network software marketplace. 3d2

A HIGH-LEVEL MODEL OF THE NETWORK ENVIRONMENT 4

The Importance of the Model Imposed by the Protocol 4a

The Protocol proposed above imposes upon the applications programmer a particular model of the network environment. In a heterogeneous computer network, nearly every protocol intended for general implementation has this effect, since it idealizes a class of operations that have concrete but slightly different equivalents in each system. Thus the ARPANET's TELNET Protocol alluded to earlier, for example, specifies a Network Virtual Terminal that attempts to provide a best fit to the many real terminals in use around the Network. 4a1

As now formulated, the Protocol models a remote resource as an interactive program with a simple, rigidly specified command language. This model follows naturally from the fact that the function-oriented protocols from which the Protocol was extracted were necessitated by the complexity and diversity of user-oriented command languages. The Protocol may thus legitimately be viewed as a vehicle for providing, as an adjunct to the sophisticated command languages already available to users, a family of simple command languages that can readily be employed by programs. 4a2

While the command/response model is a natural one, others are possible. A remote resource might also be modeled as a process that services and replies to requests it receives from other computer processes. This request/reply model would emphasize the fact that the Protocol is a vehicle for inter-process communication and that no human user is directly involved. 4a3

Substituting the request/reply model for the command/response model requires only cosmetic changes to the Protocol: 4a4

```
message-type=REQUEST [tid] op-code arguments
message-type=REPLY    tid  outcome results
```

In the formulation above, the terms "REQUEST", "REPLY", and "op-code" have simply been substituted for "COMMAND", "RESPONSE", and "command-name", respectively. 4a5

The choice of model need affect neither the content of the Protocol nor the behavior of the processes whose dialog it governs. Use of the word "command" in the command/response model, for example, is not meant to imply that the remote process can be coerced into action. Whatever model is adopted, a process has

complete freedom to reject an incoming remote request that it is incapable of or unwilling to fulfill. 4a6

But even though it has no substantive effect upon the Protocol, the selection of a model--command/response, request/reply, and so on--is an important task because it determines the way in which both applications and systems programmers perceive the network environment. If the network environment is made to appear foreign to him, the applications programmer may be discouraged from using it. The choice of model also constrains the kind and range of protocol extensions that are likely to occur to the systems programmer; one model may suggest a rich set of useful extensions, another lead nowhere (or worse still, in the wrong direction). 4a7

In this final section of the paper, the author suggests a network model (hereafter termed the Model) that he believes will both encourage the use of remote resources by the applications programmer and suggest to the systems programmer a wide variety of useful Protocol extensions. Unlike the substance of the Protocol, however, the Model has already proven quite controversial within the ARPANET community. 4a8

Modeling Resources As Collections of Procedures 4b

Ideally, the goal of both the Protocol and its accompanying RTE is to make remote resources as easy to use as local ones. Since local resources usually take the form of resident and/or library subroutines, the possibility of modeling remote commands as "procedures" immediately suggests itself. The Model is further confirmed by the similarity that exists between local procedures and the remote commands to which the Protocol provides access. Both carry out arbitrarily complex, named operations on behalf of the requesting program (the caller); are governed by arguments supplied by the caller; and return to it results that reflect the outcome of the operation. The procedure call model thus acknowledges that, in a network environment, programs must sometimes call subroutines in machines other than their own. 4b1

Like the request/reply model already described, the procedure call model requires only cosmetic changes to the Protocol: 4b2

```
message-type=CALL    [tid] procedure-name arguments
message-type=RETURN  tid   outcome        results
```

In this third formulation, the terms "CALL", "RETURN", and "procedure-name" have been substituted for "COMMAND, "RESPONSE", and

"command-name", respectively. And in this form, the Protocol might aptly be designated a "procedure call protocol (PCP)". 4b3

"The procedure call model would elevate the task of creating applications protocols to that of defining procedures and their calling sequences. It would also provide the foundation for a true distributed programming system (DPS) that encourages and facilitates the work of the applications programmer by gracefully extending the local programming environment, via the RTE, to embrace modules on other machines." This integration of local and network programming environments can even be carried as far as modifying compilers to provide minor variants of their normal procedure-calling constructs for addressing remote procedures (for which calls to the appropriate RTE primitives would be dropped out). 4b4

Finally, the Model is one that can be naturally extended in a variety of ways (for example, coroutine linkages and signals) to further enhance the distributed programming environment. 4b5

Clarifying the Procedure Call Model 4c

Although in many ways it accurately portrays the class of network interactions with which this paper deals, the Model suggested above may in other respects tend to mislead the applications programmer. The Model must therefore be clarified: 4c1

(1) Local procedure calls are cheap; remote procedure calls are not.

Local procedure calls are often effected by means of a single machine instruction and are therefore relatively inexpensive. Remote procedure calls, on the other hand, would be effected by means of a primitive provided by the local RTE and require an exchange of messages via IPC.

Because of this cost differential, the applications programmer must exercise discretion in his use of remote resources, even though the mechanics of their use will have been greatly simplified by the RTE. Like virtual memory, the procedure call model offers great convenience, and therefore power, in exchange for reasonable alertness to the possibilities of abuse.

(2) Conventional programs usually have a single locus of control; distributed programs need not.

Conventional programs are usually implemented as a single process with exactly one locus of control. A procedure call, therefore, traditionally implies a transfer of control from caller to callee. Distributed systems, on the other hand, are implemented as two or more processes, each of which is capable of independent execution. In this new environment, a remote procedure call need not suspend the caller, which is capable of continuing execution in parallel with the called procedure.

The RTE can therefore be expected to provide, for convenience, two modes of remote procedure invocation: a blocking mode that suspends the caller until the procedure returns; and a non-blocking mode that releases the caller as soon as the CALL message has been sent or queued. Most conventional operating systems already provide such a mode choice for I/O operations. For non-blocking calls, the RTE must also, of course, either arrange to asynchronously notify the program when the call is complete, or provide an additional primitive by which the applications program can periodically test for that condition.

Finally, the applications programmer must recognize that by no means all useful forms of network communication are effectively modeled as procedure calls. The lower level IPC facility that remains directly accessible to him must therefore be employed in those applications for which the procedure call model is inappropriate and RTE-provided primitives simply will not do. 4c2

SOME EXPECTATIONS 5

Both the Procedure Call Protocol and its associated Run-Time Environment have great potential for facilitating the work of the network programmer; only a small percentage of that potential has been discussed in the present paper. Upon the foundation provided by PCP can be erected higher level application-independent protocol layers that further enhance the distributed programming environment by providing even more powerful capabilities (see Appendix D). 5a

As the importance of the RTE becomes fully evident, additional tasks will gradually be assigned to it, including perhaps those of: 5b

(1) Converting parameters between the format employed internally by the applications program, and that imposed by the Protocol. 5b1

(2) Automatically selecting the most appropriate inter-process transmission format on the basis of the two machines' word sizes. 5b2

(3) Automatically substituting for network IPC a more efficient form of communication when both processes reside on the same machine. 5b3

The RTE will eventually offer the programmer a wide variety of application-independent, network-programming conveniences, and so, by means of the Protocol, become an increasingly powerful distributed-system-building tool. 5c

ACKNOWLEDGMENTS 6

Many individuals within both SRI's Augmentation Research Center (ARC) and the larger ARPANET community have contributed their time and ideas to the development of the Protocol and Model described in this paper. The contributions of the following individuals are expressly acknowledged: Dick Watson, Jon Postel, Charles Irby, Ken Victor, Dave Maynard, and Larry Garlick of ARC; and Bob Thomas and Rick Schantz of Bolt, Beranek and Newman, Inc. 6a

ARC has been working toward a high-level framework for network-based distributed systems for a number of years now [14]. The particular Protocol and Model described here result from research begun by ARC in July of 1974. This research included developing the Model; designing and documenting the Protocol required to support it [15]; and designing, documenting, and implementing a prototype run-time environment for a particular machine [16, 17], specifically a PDP-10 running the Tenex operating system developed by Bolt, Beranek and Newman, Inc [18]. Three design iterations were carried out during a 12-month period, and the resulting specification implemented for Tenex. The Tenex RTE provides a superset of the capabilities presented in the body of this paper and Appendices A through C as well as those alluded to in Appendix D. 6b

The work reported here was supported by the Advanced Research Projects Agency of the Department of Defense, and by the Rome Air Development Center of the Air Force. 6c

APPENDIX A: SUGGESTED DATA TYPES 7

The Protocol requires that every parameter or "data object" be represented by one of several primitive data types defined by the Model. The set of data types below is sufficient to conveniently model a large class of data objects, but since the need for additional data types (for example, floating-point numbers) will surely arise, the set must remain open-ended. Throughout the descriptions below, N is confined to the range [0, 2**15-1]: 7a

LIST: A list is an ordered sequence of N data objects called "elements". A LIST may contain other LISTs as elements, and can therefore be employed to construct arbitrarily complex composite data objects. 7a1

CHARSTR: A character string is an ordered sequence of N ASCII characters, and conveniently models a variety of textual entities, from short user names to whole paragraphs of text. 7a2

BITSTR: A bit string is an ordered sequence of N bits and, therefore, provides a means for representing arbitrary binary data (for example, the contents of a word of memory). 7a3

INTEGER: An integer is a fixed-point number in the range [-2**31, 2**31-1], and conveniently models various kinds of numerical data, including time intervals, distances, and so on. 7a4

INDEX: An index is an integer in the range [1, 2**15-1]. As its name and value range suggest, an INDEX can be used to address a particular bit or character within a string, or element within a list. INDEXes have other uses as well, including the modeling of handles or identifiers for open files, created processes, and the like. Also, because of their restricted range, INDEXes are more compact in transmission than INTEGERs (see Appendix B). 7a5

BOOLEAN: A boolean represents a single bit of information, and has either the value true or false. 7a6

EMPTY: An empty is a valueless place holder within a LIST or parameter list. 7a7

APPENDIX B: SUGGESTED TRANSMISSION FORMATS 8

Parameters must be encoded in a standard transmission format before they can be sent from one process to another via the Protocol. An effective strategy is to define several formats and select the most appropriate one at run-time, adding to the Protocol a mechanism for format negotiation. Format negotiation would be another responsibility of the RTE and could thus be made completely invisible to the applications program. 8a

Suggested below are two transmission formats. The first is a 36-bit binary format for use between 36-bit machines, the second an 8-bit binary, "universal" format for use between dissimilar machines. Data objects are fully typed in each format to enable the RTE to automatically decode and internalize incoming parameters should it be desired to provide this service to the applications program. 8b

PCPB36, For Use Between 36-Bit Machines 8c

Bits 0-13 Unused (zero) 8c1
Bits 14-17 Data type 8c2
- EMPTY =1 INTEGER=4 LIST=7
- BOOLEAN=2 BITSTR =5
- INDEX =3 CHARSTR=6

Bits 18-20 Unused (zero) 8c3
Bits 21-35 Value or length N 8c4
- EMPTY unused (zero)
- BOOLEAN 14 zero-bits + 1-bit value (TRUE=1/FALSE=0)
- INDEX unsigned value
- INTEGER unused (zero)
- BITSTR unsigned bit count N
- CHARSTR unsigned character count N
- LIST unsigned element count N

Bits 36- Value 8c5
- EMPTY unused (nonexistent)
- BOOLEAN unused (nonexistent)
- INDEX unused (nonexistent)
- INTEGER two's complement full-word value
- BITSTR bit string + zero padding to word boundary
- CHARSTR ASCII string + zero padding to word boundary
- LIST element data objects

PCPB8, For Use Between Dissimilar Machines 8d

Byte 0 Data type 8d1
EMPTY =1 INTEGER=4 LIST=7
BOOLEAN=2 BITSTR =5
INDEX =3 CHARSTR=6
Bytes 1- Value 8d2
EMPTY unused (nonexistent)
BOOLEAN 7 zero-bits + 1-bit value (TRUE=1/FALSE=0)
INDEX 2-byte unsigned value
INTEGER 4-byte two's complement value
BITSTR 2-byte unsigned bit count N + bit string
+ zero padding to byte boundary
CHARSTR 2-byte unsigned character count N + ASCII string
LIST 2-byte element count N + element data objects

APPENDIX C: A DETAILED ENCODING OF THE PROCEDURE CALL PROTOCOL 9

Although the data types and transmission formats detailed in the previous appendixes serve primarily as vehicles for representing the arguments and results of remote procedures, they can just as readily and effectively be employed to represent the commands and responses by which those parameters are transmitted. 9a

Taking this approach, one might model each of the two protocol messages as a PCP data object, specifically a LIST whose first element is an INDEX message type. The following concise statement of the protocol then results: 9b

```
LIST (CALL,   tid,         procedure, arguments)
      INDEX=1 INDEX/EMPTY  CHARSTR    LIST                    9b1
LIST (RETURN, tid,         outcome,   results)
      INDEX=2 INDEX        BOOLEAN    LIST                    9b2
```

The RESULTS of an unsuccessful procedure would be represented as follows: 9c

```
LIST (error, diagnostic)
      INDEX  CHARSTR                                          9c1
```

APPENDIX D: A LOOK AT SOME POSSIBLE EXTENSIONS TO THE MODEL 10

The result of the distributed-system-building strategy proposed in the body of this paper and the preceeding appendices is depicted in Figure D-1. At the core of each process is the inter-process communication facility provided by the operating system, which effects the transmission of arbitrary binary data between distant processes. Surrounding this core are conventions regarding first the format in which a few, primitive types of data objects are encoded in binary for IPC, and then the formats of several composite data objects (that is, messages) whose transmission either invokes or acknowledges the previous invocation of a remote procedure. Immediately above lies an open-ended protocol layer in which an arbitrary number of enhancements to the distributed programming environment can be implemented. Encapsulating these various protocol layers is the installation-provided run-time environment, which delivers DPS services to the applications program according to machine- and possibly programming-language-dependent conventions. 10a

The Protocol proposed in the present paper recognizes only the most fundamental aspects of remote procedure calling. It permits the caller to identify the procedure to be called, supply the necessary arguments, determine the outcome of the procedure, and recover its results. In a second paper [19], the author proposes some extensions to this simple procedure call model, and attempts to identify other common forms of inter-process interaction whose standardization would enhance the distributed programming environment. Included among the topics discussed are: 10b

(1) Coroutine linkages and other forms of communication between the caller and callee. 10b1

(2) Propagation of notices and requests up the thread of control that results from nested procedure calls. 10b2

(3) Standard mechanisms for remotely reading or writing system-global data objects within another program. 10b3

(4) Access controls for collections of related procedures. 10b4

(5) A standard means for creating and initializing processes, that is, for establishing contact with and logging into a remote machine, identifying the program to be executed, and so forth. This facility would permit arbitrarily complex process hierarchies to be created. 10b5

(6) A mechanism for introducing processes to one another, that is, for superimposing more general communication paths upon the process hierarchy. 10b6

These and other extensions can all find a place in the open-ended protocol layer of Figure D-1. The particular extensions explored in [19] are offered not as dogma but rather as a means of suggesting the possibilities and stimulating further research. 10c

REFERENCES 11

1. Kahn, R. E., "Resource-Sharing Computer Communications Networks," Proceedings of the IEEE, Vol. 60, No. 11, pp. 1397-1407, November 1972. 11a

2. Crocker, S. D., Heafner, J. F., Metcalfe, R. M., Postel, J. B., "Function-oriented Protocols for the ARPA Computer Network," AFIPS Proceedings, Spring Joint Computer Conference, Vol. 40, pp. 271-279, 1972. 11b

3. Carr, C. S., Crocker, S. D., Cerf, V. G., "Host-Host Communication Protocol in the ARPA Network," AFIPS Proceedings, Spring Joint Computer Conference, Vol. 36, pp. 589-597, 1970. 11c

4. Mc Kenzie, A. A., Host/Host Protocol for the ARPA Network, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, January 1972 (SRI-ARC Catalog Item 8246). 11d

5. Walden, D. C., "A System for Interprocess Communication in a Resource Sharing Computer Network," Communications of the ACM, Vol. 15, No. 4, pp. 221-230, April 1972. 11e

6. Cerf, V. G., Kahn, R. E., "A Protocol for Packet Network Intercommunication," IEEE Transactions on Communications, Vol. Com-22, No. 5, pp. 637-648, May 1974. 11f

7. Thomas, R. H., "A Resource-Sharing Executive for the ARPANET," AFIPS Proceedings, National Computer Conference, Vol. 42, pp. 155-163, 1973. 11g

8. TELNET Protocol Specification, Stanford Research Institute, Menlo Park, California, August 1973 (SRI-ARC Catalog Item 18639). 11h

9. Engelbart, D. C., Watson, R. W., Norton, J. C., "The Augmented Knowledge Workshop," AFIPS Proceedings, National Computer Conference, Vol. 42, pp. 9-21, 1973. 11i

10. Engelbart, D. C., English, W. K., "A Research Center for Augmenting Human Intellect," AFIPS Proceedings, Fall Joint Computer Conference, Vol. 33, pp. 395-410, 1968. 11j

11. Irby, C. H., Dornbush, C. F., Victor, K. E., Wallace, D. C., "A Command Meta Language for NLS," Final Report, Contract

RADC-TR-75-304, SRI Project 1868, Stanford Research Institute, Menlo Park, California, December, 1975. 11k

12. Neigus, N. J., File Transfer Protocol, ARPA Network Working Group Request for Comments 542, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, July 1973 (SRI-ARC Catalog Item 17759). 11l

13. Bressler, R. D., Guida, R., Mc Kenzie, A. A., Remote Job Entry Protocol, ARPA Network Working Group Request for Comments 360, Dynamic Modeling Group, Massachusetts Institute of Technology, Cambridge, Massachusetts, (undated) (SRI-ARC Catalog Item 12112). 11m

14. Watson, R. W., Some Thoughts on System Design to Facilitate Resource Sharing, ARPA Network Working Group Request for Comments 592, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, November 20, 1973 (SRI-ARC Catalog Item 20391). 11n

15. White, J. E., DPS-10 Version 2.5 Implementer's Guide, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 15, 1975 (SRI-ARC Catalog Item 26282). 11o

16. White, J. E., DPS-10 Version 2.5 Programmer's Guide, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 13, 1975 (SRI-ARC Catalog Item 26271). 11p

17. White, J. E., DPS-10 Version 2.5 Source Code, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 13, 1975 (SRI-ARC Catalog Item 26267). 11q

18. Bobrow, D. G., Burchfiel, J. D., Murphy, D. L., Tomlinson, R. S., "TENEX, a Paged Time Sharing System for the PDP-10," Communications of the ACM, Vol. 15, No. 3, pp. 135-143, March 1972. 11r

19. White, J. E., "Elements of a Distributed Programming System," Submitted for publication in the Journal of Computer Languages, 1976. 11s

FIGURE LIST 12

< KJOURNAL, 27201.NLS;1, >, 23-DEC-75 16:00 XXX ;;;; .HJOURNAL="JEW 23-DEC-75 15:45 27201"; Title: .H1="Comments on MIT-DMS' DEC 2-4 1975 Message Services Protocol Proposal"; Author(s): James E. (Jim) White/JEW; Distribution: /SRI-ARC([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: JEW; .IGD=0; .SNF=HJRM; .RM=HJRM-7; .PN=-1; .YBS=1; .PES; Origin: < JWHITE, MTPCOMS.NLS;2, >, 23-DEC-75 15:40 JEW ;;;;####;

.PEL; .PN=PN-1; .GCR;SNDMSGed to Mail Services Committee.

INTRODUCTION

The following comments on the five-section mail protocol document distributed at the last Mail Services Committee meeting are offered with the hope that they will be found useful by the MIT crew that has obviously worked so hard. I have collected what to me are the most important comments in the first major section of the memo. The remaining comments appear in the second section and are keyed to the particular sections of the mail protocol document to which they refer.

MAJOR COMMENTS

Describing the Target System

As I pointed out at the meeting, there exists (to my knowledge) no written description of the distributed mail system we're trying to build. Until this document exists, one really has, in the end, no basis for judging whether or not the proposed mail protocol is adequate. It seems like the cart has gotten before the horse (not that it hasn't before). Before a protocol can be settled upon (designed), the list of mail system capabilities that was started at the meeting and recently distributed must be completed, given much more detail, and agreed to; and a model of the required distributed system defined. This is a substantial task, and not one, I think, that can be pushed off on the necessarily small protocol design team.

S-PGROUPs and s-DGROUPs, for example, appear to be premature and somewhat incomplete solutions to what is in reality a major issue in the mail system design, namely, that of defining and maintaining distribution or other group lists. This is an example of the trouble one gets into by attempting to specify a protocol before the end system has been adequately modeled. The definition of s-USER strikes me much the same way; I don't know whether it's right or not because I don't have enough information about the larger framework in which it will sit.

Recognizing the True Scope of a Structured Data Transmission Protocol

As I suggested at the meeting, the contents of Section II should be advanced within the Network community as a general-purpose, application independent protocol that could become the foundation for a large number of applications protocols (of which the mail protocol is but one). This is not

to say that the Committee should wait for or even actively seek network-wide agreement on this issue, but rather simply that Section II should be rid of its (already very few) references to mail and issued as a separate document (preferably a Request for Comments).

Standardizing Written Descriptions of Structured Data Items

If the structured data transmission protocol indeed becomes widely used, descriptions of structured data items will become a major element of most applications protocols. If we define reasonable and comprehensive syntax conventions, they will probably be employed by others. But if those we define are awkward or incomplete, private conventions will arise on every front and chaos will result. Subsection V of Section II, entitled "printing conventions," therefore warrants considerable thought and probably some expansion.

In DPS, we defined syntax conventions that were incomplete, and constantly found ourselves unable to describe the argument or result of some remotely-callable procedure. Perhaps we should do it right this time. Wasn't John Pickens doing some work in the area of data description languages?

Factoring Out the Request/Reply Discipline

In precisely the same spirit as the structured data transmission protocol, the request/reply framework described in Section V should be structured and documented as a separate protocol of potential utility in many other application domains besides mail.

One of what I personally believe to be the major long-term benefits of such an approach is the following. Just as the structured data transmission protocol encapsulates all the detailed (bit-level) network connection format issues and so elevates our thinking to the more abstract level of INTEGERs, STRINGs, and BOOLEANs, so an application-independent request/reply protocol would encapsulate the mechanics of issuing requests and receiving replies and so elevate us to the even more abstract level of (if you will) the calling sequences of remote functions.

I recommend that we extract the request/reply discipline described in the beginning of Section V and issue it as a separate document (an RFC) proposing an application-independent protocol that might provide a foundation for a variety of applications protocols.

OTHER COMMENTS

Global Comments

1) Although much better in this regard than the previous draft, the protocol still often allows a parameter to be specified either as a p-STRING or a p-INTEGER. This option invariably

complicates both the definition and implementation of the protocol without offering any additional functional capability in return. Some examples are:

Sec II p 17: semantic types.

Here p-STRINGs are allowed for experimental types. An equally powerful but more efficient approach would be to simply define a range of p-INTEGER values for experimental semantic types.

Sec III p 8: group ids.

Here p-STRINGs may be the more appropriate choice, but allowing both p-STRINGs and p-INTEGERs seems to buy nothing while complicating implementations.

Sec III p 10: dynamic group ids (see comment above).

Sec III p 15: field ids.

Sec IV p 5: message fields.

2) There are also a number of cases in which the legal values for certain parameters are defined to be literal p-STRINGs when p-INTEGERs would do just as well and be much easier to decode. Remember that we're defining a protocol for inter-machine communication. A process can easily use the p-INTEGER as an index into a table of mneumonics if and when it becomes necessary to communicate the parameter to a human user. Examples:

Sec IV p 20: handling instructions.

Sec IV p 21: message status.

Comments on Section II (Data Types and Formats)

Global Comments

1) The definition (semantics) of the various primitive data types should, I think, be isolated from the definition (syntax) of the transmission format in which they are to be encoded. At some future date it may prove useful, for reasons of efficiency, to define SEVERAL formats (for 8-bit, 16-bit, 32-bit, and/or 36-bit connections) and include as part of the protocol a negotiatory phase in which the most appropriate format is selected.

2) We might consider thinking some about the interface between the encoder/decoder and the applications program, and include some of that thinking in the document as an implementation guide. For example, how should semantic types be communicated to the applications program? Also, what does the interface look like for b-INTs? The encoder/decoder, for example, might be capable of being placed in a mode in which

it automatically detects and somehow flags as in error, integers whose representation requires more than 32 or 36 bits, depending upon the word length of the machine.

3) Why not define "[x]" as shorthand for "x ! p-EMPTY" and use it wherever possible throughout Sections II-V. The something-specific-or-EMPTY alternative is VERY common, and this convention would, I think, significantly improve the document's readability.

Choice of Data Type Names

1) Primitive and byte-stream data types should be named more consistently. For example:

Why Not ...		Instead of ...	
INTEGER	LINTEGER/SINTEGER	INTEGER	INT/SINT
BITSTR	LBITSTR/SBITSTR	BITS	LBITS/BITSTR

2) The names of primitive data types (as opposed to byte-stream data types, which will never be talked about outside this section) should be fairly carefully chosen because with any luck they will be used in the definition of MANY applications programs yet to come. In that regard, I would suggest that "MULT" be changed to something else (e.g. "STRUCture", a noun) so as not to sound awkward.

3) In the same spirit as (2), this section should explicitly suggest (at the end) that the "p-" prefix need not be employed in subsequent documents. This will make descriptions of applications protocols more readable.

4) How about b-PADDING (a noun) instead of b-IGNORE (a verb).

Typographical Errors

1) Page 3 says that p-BITS are SHORT bit strings. If p-BITS is the external manifestion of both b-BITSTR and b-LBITS, that isn't true.

2) EMPTY is missing from the table on page 5, and booleans are incorrectly defined as 0000000X, instead of 0000001x.

Semantic Data Types

1) At present I have trouble with the notion of semantic types. It seems to muddy the interface between the encoder/decoder and the higher-level program. I may be wrong here.

2) On page 6, it is suggested that 8-bit characters, if found to be necessary, can be introduced as SEMANTIC TYPES. This seems to me a very bad idea; instead, introduce if necessary a b-CHAR8 which produces a p-CHAR or contributes to

a b-STRING. Semantic types should not become a vehicle for application-independent features of the protocol that we find necessary but neglected to define as part of the initial specification. They should rather, as their name suggests, be reserved for truly application-dependent data types. Perhaps we might wish to define a p-EXPERimental or b-EXPER via which candidates for inclusion in the protocol can be tested.

Character Strings

1) P-STRINGs can only be generated via a b-STRUC of b-CHAR7s. This effectively requires of many hosts that the high-order bit of each character be turned on before transmission through the Network, an unnecessary and in may cases expensive operation. One really should define a b-STRING7 data type which allows the string to be transmitted directly as eight-bit bytes but with the high-order bit of each byte off. I realize that defining and using b-STRINGs would thwart use of b-REPEAT in packing repeated characters, but that problem can surely be handled in another way.

2) Defining a p-MULT of p-CHARs to be equivalent to a p-STRING is a needless complication that seems bound to lead to trouble. I have a feeling that this convention was suggested by the fact, noted in (1), that a b-STRUC of b-CHAR7s generates a b-STRING.

3) The mail protocol itself seems to have no need for p-CHAR. If the goal was to provide an efficient means of transmitting single-character strings, this should have been handled via appropriate byte-stream objects so as not to affect the operation of anyone but the encoder/decoder. Why not drop p-CHAR from the protocol and redefine b-CHAR7 to generate a p-STRING of length one.

Miscellaneous

1) Assigning version numbers to every semantic data type seems extravagant. I find it hard to believe that mail processes will really be so sophisticated as to try version n of some data type and, if it's rejected, back off and try n-1. And if they DON'T do that kind of thing, we may find ourselves with a lot of processes that, though very clever, can't talk to each other. Why not instead assign a version number to the protocol as a whole?

2) After reading through their description several times, I still don't understand the utility of b-SEGMENTs. The claim is that they provide a means by which commonly used data sequences can be pre-encoded and then inserted at will into the output stream. But why can't one simply insert the sequence without the b-SEGMENT header? Is the b-SEGMENT envelope just a place to store the length of the sequence so one knows how many bits to copy into the output stream? If so, why not maintain the length information privately, rather

than include it in the output stream with the sequence?

Comments on Section III (Semantics Types for Message Service)

Miscellaneous

1) In the definition of s-INDIRECT, it would be helpful to mention, in the case where the semantic data type denotes a file, that the definition of s-FILE contains information about how the contents of the file are to be decoded to form the data items. This would prevent the reader from thinking, as I did, that the s-INDIRECT definition had a hole in it.

2) Why not combine s-PGROUP and s-DGROUP as additional cases of s-INDIRECT? They all seem to serve similar functions. Furthermore, the names "PGROUP" and "DGROUP" seem to describe one possible use of the data type, not all possible uses; and are therefore somewhat misleading.

3) Since they have identical syntax, we might consider combining s-TIME and s-RTIME into a single semantic type, including an additional parameter to distinguish the two. Perhaps there are other reasons why you didn't do this.

Comments on Section IV (Message Fields)

None

Comments on Section V (Mail Service Requests and Replies)

1) The request/reply discipline seems unnecessarily complex in terms of the number of protocol utterances defined. Why not, for example, define a single reply format incorporating the features of ACK, NACK, LATER, ERROR, and ANSWER? This reply might look something like:

```
REPLY (outcome explanation results)

   outcome      -- p-BOOLEAN with value COMPLETED!FAILED

   explanation  -- (codeforpgm textforuser)
                    p-INTEGER  p-STRING

   results      -- (parameter1 ... parametern)
```

With TEXTFORUSER, CODEFORPGM, EXPLANATION, and RESULTS being optional (i.e. possibly EMPTY). The specific replies defined in the current protocol proposal would have roughly the following representations in this new reply structure:

```
   ACK                              ==> REPLY (TRUE)

   NACK                             ==> REPLY (FALSE)

   ERROR (errcod reason) NACK ==> REPLY (FALSE (errcod
   reason))
```

```
LATER (notbefore at) NACK ==>
   REPLY (FALSE (later) (notbefore at))

ANSWER (code arg1 ... argn) ACK ==>
   REPLY (TRUE (code) (arg1 ... argn))
```

The philosophy apparently embodied in Section V's reply structure is that application-dependent and application-independent information must be isolated in separate MESSAGES, and likewise for information which all mail processes must understand and that which only sophisticated processes need care about. In the structure above, these same concerns are handled in a different way. For example, simple-minded implementations can determine whether a transaction failed by simply consulting the boolean OUTCOME, and ignore any RESULTS that may have been returned. If the transaction failed, they may similarly ignore the reason CODEFORPGM for the failure.

2) Semantically typing the request- and reply-bearing data items seems a bad idea, especially if the request/reply structure is isolated as a separate, application-independent protocol, as discussed earlier. Once an application-independent request message has been defined, it will no longer be necessary to define the specific requests required by the mail protocol as semantic items.

3) S-ABORT may belong as part of the application-independent request/reply protocol.

4) The dialog structure seems to require an inordinate number of inter-process handshakes, even in the simplest and most common cases. To avoid this problem one must, I think, define higher-level composite operations that can be invoked with a single command. To deliver a piece of mail, for example, one will ALWAYS have to execute the SETMSG, RCVMSG, and DLVRTO functions. Why not combine them (SETMSG can be retained for use in dialogs other than delivery)?

5) The protocol contains an inordiate number of functions by use by one process in determining the capabilities or sophistication of the other (i.e. RESTRICT, SETFLDS, DISALLOW, SNDFLDS, HANDLE, and LSTFLDS). Since it costs little or nothing to generate this information, why not require that it be exchanged at the start of every inter-process dialog and define a single function for that purpose?

6) Defining a request (i.e. OVER) which MAY but need not necessarily draw a reply seems like shaky business.

BLUE-PENCIL EDITOR

< IJOURNAL, 27244.NLS;1, >, 2-JAN-76 17:31 XXX ;;;; .HJOURNAL="BEV 2-JAN-76 19:08 27244"; Title: .H1="An Additional Wish for Your ever-growing Wish List"; Author(s): Beverly Boli/BEV; Distribution: /DVN([INFO-ONLY]) DPCS([INFO-ONLY]) KIRK([INFO-ONLY]) ; Sub-Collections: SRI-ARC DPCS; Clerk: BEV; .IGD=0; .SNF=HJRM; .RM=HJRM-7; .PN=-1; .YBS=1; .PES;

.PEL; .PN=PN-1; .GCR;Dirk--Here is my super-traditional editor that we talked about some weeks ago. Its fondest wish is to make the Big List.
Bev

A True Blue Pencil Editor

The True Blue Pencil Editor would perform functions closely resembling traditional blue-pencil editing, rather than making actual changes in the manuscript as the Base "editor" does. To use it, an editor would load the file to be edited, and Goto the True-blue subsystem. This subsystem would allow the editor to read the file and mark comments, corrections, proof-reading marks, etc., on the file such that the author could read the suggested changes, but they would not have actually been made in the file.

It is important that the editor's marks do not look like they are part of the original text, as is now the case. Instead the True-blue subsystem might create blank space on either side of the text where the editor could mark corrections, provide a list of symbols the editor might use to indicate corrections (e.g. a sp character, paragraph character, capitalization marks, etc.), and enable the editor to insert re-writes such that the place to be revised is indicated, and the suggested re-write is shown nearby). A file might look like this when the editor completed it:

(sp)Xanadu	in Zanadu did Kubla Kahn	u.c./K
#stately#	a #noble# pleasure Dome degree	l.c./d
(sp)decree		
(sp)sacred	where Alph the scared river runs...	

A larger screen would be nice for this, wouldn't it?

The subsystem would have an additional important function. After the edits have been made to a ms., the author may review it to decide which changes she would like to keep and which to ignore. The subsystem would allow her to go through the file, signalling the edits she would like to have incorporated into the file permanently. The subsystem would provide commands to aid the author in incorporating or ignoring the edits. (For example, if the editor has indicated that a paragraph [new statement] should begin at a certain point, and the author concurs, the author could give the command "Incorporate (change into file)" and bug the paragraph mark. The subsystem would then break the statement at that point, creating a new paragraph. There might also be an "Incorporate All (edits in) STRUCTURE" command.) A default of either ignoring or making edits could be set; if it were the former, the author would have to indicate individually the edits she wanted to keep. Then when the file was updated, all of the other edits would disappear. Or the reverse could be made the case, and all edits would be incorporated unless specifically deleted (with the "Delete (edit)" command).

< IJOURNAL, 27249.NLS;1, >, 5-JAN-76 12:46 XXX ;;;; .HJOURNAL="ROM 5-JAN-76 14:02 27249"; Title: .H1="Professional Journal subscription"; Author(s): Raphael Rom/ROM; Distribution: /SRI-ARC([ACTION]) ; Sub-Collections: SRI-ARC; Clerk: ROM; .IGD=0; .SNF=HJRM; .RM=HJRM-7; .PN=-1; .YBS=1; .PES; Origin: < ROM, TEMP.NLS;6, >, 5-JAN-76 13:57 ROM ;;;;####;

The follwoing publications have been suggested by ARC members for subscription. (If you have any additional suggestions let me know soon, as we are starting the subscription process.)

Communication of the ACM
Journal of the ACM.
ACM conference proc.
ACM's Computing Survey's.
IEEE Transactions on Software Engineering.
IEEE Transactions on Computers.
IEEE Transactions on Info Theory.
IEEE Transactions on Systems, Man, and Cybernatics.
IEEE conference proc.
IBM Systems Journal.
SIAM journal.
AFIPS proc.
ICCC proc.
Fortune.
Business Week.
Computer News.
Datamation.
Creative Computing.
Computers, Control, and Info Theory (on order).
Modern Data.
Reprographics.
Computer Decisions.
Tek Graphics.
Naval Research Reviews.
Digital News.
Tekscope.
Computer.
Minicomputer News.
Word Processing World.
Computer Networks (on order).
Electronic News (on order).
Proc. of special symposia on reliable software, OS etc., as they occur.

If you would like any of those to be subscribed under your name please let me know.
It has also been suggested that ARC members join a few of the professional groups that are of interset to ARC. If you would like to join such a group let me know and we'll see how it can be arranged. Also let me know if you are already a member of a group so we can avoid duplications.

Week Ending 1/9/75

GENERAL

Things started out slowly this week in that everyone is afraid to make any major starts or changes while they are waiting for "sanction" on any NLS/NSW projects. Major Hearn submitted a proposal (DPD) to Col. Brunner who will probably forward it to Washington when some additional changes are made. Until that proposal gets approval, there are no available funds to support any projects involving NLS at the base. Things can continue to be done as before which means somewhat under the table. By the end of the week, I had uncovered quite a few new projects and as usual am now inundated with additional work.

66-1

I spoke with Col. Simmons who would be the one in charge of 66-1 if it came down to Gunter. He is still waiting for Washington to make a decision on that which could be quite a while. I plan to have Grace and Joann (the two who did the 66-1 rewrites and are well trained in editing, formatting and Proof) start on some small project that is related to their work now so that they do not lose the skills they have developed. Unfortunately, Col Simmons views them as secretaries and is not willing to have them do anything that would distract them from answering the phones.

BASE TOPS IN PR

The input of additional annexes for the Base tops reports is continuing in PR and is going well.

STALOG

I explained some sophisticated retreive and sort procedures to STALOG and with the help of Jay Lowe who is new to them, they will be able to do some fancier things than before. We are still looking at the possibility of expanding their project to make it a more comprehensive data management system, but they will need to find some funds in order to do that.

FISCHER'S MONTHLY REPORT

Bev St. Pierre continues to do Fischer's monthly report and he is pleased with that. Rumor has it that he wants to do another report using NLS. He has begun to write up his documentation proposal for the Base, and it does not look like NLS will be his choice. That report is due to emerge in the next week or so.

BLAP (Base Level ADPS Profile)

I am working on the design of a report that is about 700 pages long. It includes information from all 140 bases around the world and tells about their equipment and projected equipment as related to the B4700 and the H1050. It is a report that has about 5 pages per base and each page consists of only COLUMNS. It is for Major Hearn and some people in his office. It is a major undertaking as far as input, but once entered, the changes should be easy to do. I will know more about this after working on it this weekend. At least it will keep me busy.

EQUIPMENT

we still only have two DNLS terminals on base. There is an ADM2 that is broken and no one knows where it came from so maintenance can not be requested at this point. They do have an lp from us that is for the adm2, but that also sits unused. I am investigating the situation. The RML tip was transfered here this week and Walt Lamia has made the phone lines work so we can dial it directly. That means we will soon?? have nine dialups..5 to

the elf and 4 to the tip. We hope to get all DNLS stations hardwired to the tip. All future installations in the Block House will be hardwired to the Tip.

WEATHER

Snowed two days ago and it was 18 degrees this morning..I thought this was supposed to be the balmy south. Oh well, in Alabama I guess anything can happen!!!!!!

< MJOURNAL, 27356.NLS;1, >, 16-JAN-76 16:35 XXX ;;;; Title:
Author(s): David C. Smith, David S. Maynard, Jan A. Cornish/DAV DSM
JAC3; Distribution: /DBM([INFO-ONLY]) ; Sub-Collections: SRI-ARC
DBM; Clerk: DAV;

LEVEL 1 5

The primitives involved in information retrieval are not all on the same level. Some are "lower level" than others, and are used by "higher level" primitives. We will first discuss what we consider to be the most basic primitives. We will then consider the formulation of higher level primitives. However one man's high level is another's low, so there is no absolute "high" level. Rather it is a continuum, and we will discuss a useful point in the continuum. 5a

BASIC RETRIEVAL PRIMITIVES -- Summary 5b

Sequence creation 5b1

```
sequence _ STATEMENT(statement, viewspecs)
sequence _ BRANCH(statement, viewspecs)
sequence _ GROUP(statement1, statement2, viewspecs)
sequence _ PLEX(statement, viewspecs)
sequence _ FILE(statement, viewspecs)
sequence _ SEQUENCE(statement, address-expression,
viewspecs)
sequence _ SEQUENCE(statement, function, viewspecs)
sequence _ POINTER(sequence)
TRUE,FALSE _ CLOSE(sequence)
```
5b1a

Statement generation 5b2

```
statement _ THIS(sequence)
statement _ NEXT(sequence)
statement _ COMPUTE(statement, address-expression)
```
5b2a

Predication 5b3

```
TRUE,FALSE      _ MATCH(content-pattern, statement)
statement,FALSE _ SEARCH(content-pattern, sequence)
```
5b3a

Attributes of statements 5b4

Names 5b4a

```
string _ GETNAME(statement)
string _ PUTNAME(statement, string)
string _ REMOVENAME(statement)
statement _ SEARCHNAME(string, sequence)
```
5b4a1

Text 5b4b

```
string _ GETTEXT(statement, left-delim, right-delim)
```

Information Retrieval Primitives

There are presently at least six independent efforts by users of NLS to design information retrieval systems: HELP, QUERY, RETRIEVE, FMS, EPOC and CIMROS. This number will certainly grow in the future. Comparing the systems leads to the helpful conclusion that they are all resonably similar. Though this means that currently there is much duplication of effort, it suggests that for the future a small set of primitive functions can be designed to do common information retrieval operations. 1

This memo presents the beginnings of such a set of functions and develops a conceptual framework to guide additions to the set. The primitives will be available to the NLS user for defining his own information retrieval system. First we will simply list the primitives; then we will explain them in detail. 2

The discussions which led to this set of primitives must necessarily be omitted. However, we want to emphasize that we are not presenting people with a Procrustean bed. The primitives here by no means exhaust the possibilities. What we have really developed is a philosophy which has guided the specification of some initial primitives, and which can guide additions to them later. The philosophy can be summed up as: 3

> Present the user with a conceptually clean access to facilities present in NLS. 3a
>
> In this spirit, define functions which manipulate the normal objects of information retrieval systems in useful and efficient ways. 3b

For the first phase, we wanted to build the primitives on top of existing NLS, without trying to change the underlying structure. A second phase is to add facilities to NLS which will improve its information management capabilities. An example is to add properties of statements which, like names, can be put in a (hashed) table and fast lookups done using that table. Most of the primitives presented below are first-phase primitives. 4

```
        string _ PUTTEXT(statement, left-delim, string,
        right-delim)
        string _ REMOVETEXT(statement, left-delim,right-delim)
        statement _ SEARCHTEXT(left-delim, right-delim, sequence)   5b4b1

    Properties                                                      5b4c

        object _ GETPROP(statement, prop-name, prop-type)
        object _ PUTPROP(statement, prop-name, prop-type, object)
        object _ REMOVEPROP(statement, prop-name, prop-type)
        statement _ SEARCHPROP(prop-name, prop-type, sequence)      5b4c1

Others                                                              5b5

    string  _ GETLINK(string)
    integer _ LEVEL(statement)                                      5b5a

BASIC RETRIEVAL PRIMITIVES -- Explained                             5c

    The semantics of these primitives will now be explained in more
    detail.  We will discuss an outline of the implementation
    later.  The descriptions below have the form                    5c1

        <value> _ FUNCTION(arg1, arg2, ...)                         5c1a

    The types of arguments and values are the following:            5c2

        sequence                - the address of an n-word record
        statement               - an NLS stid
        string          - the address of an L10 string
        viewspecs               - the address of a two-word block
        address-expression - the address of an L10 string
        content-pattern         - the address of an L10 procedure   5c2a

    Sequence creation                                               5c3

        sequence _ STATEMENT(statement, viewspecs)
        sequence _ BRANCH(statement, viewspecs)
        sequence _ GROUP(statement1, statement2, viewspecs)
        sequence _ PLEX(statement, viewspecs)
        sequence _ FILE(statement, viewspecs)                       5c3a

            These five are just special cases of the following
            function.  Applying NEXT (q.v.) will yield a sequence of
            statements coming from the relevant NLS structure.      5c3a1

        sequence _ SEQUENCE(statement, address-expression,
        viewspecs)
        sequence _ SEQUENCE(statement, function, viewspecs)         5c3b
```

This is the biggie. ARC'ers will recognize it as just the NLS sequence generator "cleaned up". It sets up a data structure with enough state information to generate a sequence of statements. (We don't know exactly what these entries are yet.) Applying NEXT evaluates either an address expression (e.g. ".u.s.a") or a function with respect to a statement and viewspecs to generate the next statement. 5c3b1

Aside: There will probably have to be two functions, since L10 can't tell whether a function or an address expression is being passed. 5c3b1a

One field in the data structure will tell what "kind" of a sequence it is, i.e. did it come from PLEX, GROUP, etc.? This information will be used, at first, in only one way: since "Jump (to) Name Any" is a fast search in NLS, the particular case 5c3b2

SEARCHNAME(string, FILE(...)) 5c3b2a

will do "Jump (to) Name Any". If other fast searches are added to NLS, these primitive functions will take advantage of them. 5c3b3

These "sequences" are not just instances of sequence generators. Because of the large size of sequence generators, NLS permits no more than four to exist at any one time. Furthermore, the user has to explicitly open and close them, so that, for example, functions which return sequences cannot be passed as parameters to other functions, as in the SEARCHNAME example above. Instead, the "sequences" here are just instances of records, n-word blocks of data. (We don't know what "n" equals yet.) 5c3b4

There are various other implementations of sequences, for example as co-routines. Briefly, they are not: 5c3b4a

co-routines, because co-routines do not exist in NLS 8 or 8.5; 5c3b4a1

sequence generators, because there can be no more than four at any one time, and they must be explicitly opened and closed; 5c3b4a2

a single sequence generator which is "swapped" for each sequence, because this doesn't work. 5c3b4a3

The implementation chosen is to define a duplicate set of sequence generator functions which do not have stacks. The sequence generator is now a set of sub-routines, rather than co-routines. No "sport"s are allowed. All state to be remembered must be stored in the sequence record. (In fact, the current sequence generator does not even need its stack for most cases, only for fancy sequences.) We feel that we can implement all the current information retrieval systems without having to use co-routine sequence generators. 5c3b5

sequence _ POINTER(sequence) 5c3c

This makes a copy of a sequence record. Since the complete state of a sequence is represented in its record, POINTER is a way to remember a place in a sequence before doing a state-changing function like NEXT or SEARCH. 5c3c1

For example, the user can test an hypothesis by trying a SEARCH; if that doesn't work, he can go back to his place in the sequence and try some other SEARCH. 5c3c1a

TRUE,FALSE _ CLOSE(sequence) 5c3d

CLOSE de-allocates the sequence record, sets up a new one-word record that is marked as "exhausted", and returns TRUE if it did this successfully, FALSE otherwise. Sequences which become exhausted are automatically closed (see NEXT), or the user can (but need not) close them explicitly. Sequences which are not closed stay around forever, but this only wastes n-words of space. (If we had a garbage collector, even those words could be reclaimed.) 5c3d1

Statement generation 5c4

statement _ THIS(sequence) 5c4a

A sequence always contains as part of its state the next statement that will be returned when next NEXT is applied. (Hmmm.) THIS peeks at it and returns the statement without advancing the sequence, as NEXT does. 5c4a1

statement _ NEXT(sequence) 5c4b

This is the principle function for extracting the "next" statement from a sequence. It applies the sequencing function (the second argument to SEQUENCE) to a statement

and viewspecs to compute a new statement. In the case where the sequence is a BRANCH, GROUP, PLEX or FILE, it is just like "Jump (to) Next". But the sequencing function may be a more complicated algorithm. (It may not, however, do "sport"s, or call functions which do "sport"s.) 5c4b1

As a side effect, NEXT changes the state of the sequence to point to the next statement after the one it returns, which is the only difference between THIS and NEXT. All sequencing functions must return FALSE when a sequence is exhausted, whereupon NEXT will CLOSE the sequence and return FALSE. 5c4b2

statement _ COMPUTE(statement, address-expression) 5c4c

This evaluates an address expression (e.g. ".u.s.d") with respect to a statement to yield a new statement. This is just a cleaned-up "caddexp". 5c4c1

Aside: This is the function called when 5c4c1a

NEXT(SEQUENCE(statement, address-expression, viewspecs)) 5c4c1a1

is evaluated. 5c4c1b

Predication 5c5

TRUE,FALSE _ MATCH(content-pattern, statement) 5c5a

This applies a content analysis program to a statement and returns TRUE if the statement matches the pattern, FALSE otherwise. 5c5a1

statement,FALSE _ SEARCH(content-pattern, sequence) 5c5b

This searches a sequence for a statement which matches a content analysis program. It returns the statement if it finds one, FALSE otherwise. As a side effect, it sets the sequence to point to the next statement after the one it returns. If it does not find a statement, it CLOSEs the sequence. If the user did not want to lose his place in the sequence, he should get another POINTER to it before doing the SEARCH. 5c5b1

Attributes of statements 5c6

There is a whole class of primitive functions dealing with

attributes of statements. Each involves GET, PUT, REMOVE, SEARCH. Most of these are self-explanatory. 5c6a

All GETs return FALSE if 'name' does not exist in 'statement'. 5c6b

All PUTs replace 'value' if 'name' already exists in 'statement'. Otherwise 5c6c

PUTNAME makes a statement name; 5c6c1

PUTTEXT appends "left-delim string right-delim" onto the end of the statement's text; 5c6c2

PUTPROP appends "prop-name prop-type object" onto the end of the statement's property list. 5c6c3

All REMOVEs return FALSE if 'name' does not exist in 'statement'. 5c6d

All SEARCHes return FALSE if 'name' does not exist in 'sequence'. 5c6e

Names 5c6f

```
string    _ GETNAME(statement)
string    _ PUTNAME(statement, string)
string    _ REMOVENAME(statement)
statement _ SEARCHNAME(string, sequence)
```
5c6f1

These manipulate the name of a statement. 5c6f1a

Text 5c6g

```
string    _ GETTEXT(statement, left-delim, right-delim)
string    _ PUTTEXT(statement, left-delim, string,
right-delim)
string    _ REMOVETEXT(statement, left-delim,
right-delim)
statement _ SEARCHTEXT(left-delim, right-delim, sequence)
```
5c6g1

These manipulate strings within a statement. This is useful for storing multiple fields of a data structure in a single statement.(Other primitives might want to be added which move and insert text at a designated place in a statement, e.g. after a logical field, rather than at the end as PUTTEXT does. 5c6g1a

'Left-delim' and 'right-delim' are strings which

delimit a field within a statement. For example, 'left-delim' might be a field name, and 'right-delim' might be a carriage return. 5c6g1b

Property 5c6h

object _ GETPROP(statement, prop-name, prop-type)
object _ PUTPROP(statement, prop-name, prop-type, object)
object _ REMOVEPROP(statement, prop-name, prop-type)
statement _ SEARCHPROP(prop-name, prop-type, sequence) 5c6h1

These manipulate the property list of a statement. This has a wide range of possible applications, from multiple texts to hashed retrieval. However this will probably not be implemented in the first phase because NLS does not permit named properties of arbitrary type on property lists. In fact, these functions are only included here for completeness. Note: 'prop-name' is a string; 'prop-type' is an integer. 5c6h1a

Others 5c7

string _ GETLINK(string) 5c7a

If a string contains an NLS link, GETLINK will return a string containing just that link. Otherwise it returns FALSE. 5c7a1

integer _ LEVEL(statement) 5c7b

This returns the level of a statement as an integer. 5c7b1

Examples 5c8

seq _ PLEX(1, "d") 5c8a

seq _ GROUP(1, 5, "w") 5c8b

seq _ SEQUENCE(1, ".n", "dbb") 5c8c

seq _ SEQUENCE(1, $mynext, "dbb") 5c8d

st _ NEXT(seq) 5c8e

b _ MATCH("abc", NEXT(seq)) 5c8f

st _ SEARCH("abc", PLEX(5a, "wdb")) 5c8g

```
str _ GETNAME(st)                                                    5c8h

str _ PUTNAME(st, "myname")                                          5c8i

st  _ SEARCHNAME("myname", BRANCH(1, "w"))                           5c8j

str _ GETTEXT(st, "Account: ", EOL)                                  5c8k

str _ PUTTEXT(st, "Account: ", "100", EOL)                           5c8l

st  _ SEARCHTEXT("Account: ", EOL, PLEX(1, "x"))                     5c8m

obj _ GETPROP(st, "DIRECTIVES", texttype)                            5c8n

obj _ PUTPROP(st, "DIRECTIVES", texttype,
        LIST(text-pointer, ".IGR;") CONS
          GETPROP(st, "DIRECTIVES", texttype));                      5c8o

obj _ REMOVEPROP(st, "DIRECTIVES", texttype);                        5c8p

st  _ SEARCHPROP("DIRECTIVES", texttype, PLEX(1a, "db"));            5c8q

str _ GETLINK("Links: <link1>, <link2>, <link3>")                    5c8r
```

LEVEL II+ 6

We will now discuss a higher level of information retrieval primitive, defined in terms of logical "records" and "fields" instead of "strings" and "statements". This is both a concrete suggestion and an indication of the kinds of generalizations to the information retrieval package that we envision. 6a

INTRODUCTION 6b

Several NLS user subsystems, such as FMS, CIMROS and EPOCH, have been written to manage record oriented data bases. The similarities between the subsystems are striking. The following is an exploration of the possibility of developing a generalized system for maintaining record oriented data bases using NLS. The advantage of this generalization would be to greatly simplify the task of implementing a subsystem to manage a record oriented data base. 6b1

The existence of the CIMROS, EPOCH, and FMS subsystems demonstrate that NLS provides a sufficiently powerful set of primitives for performing data management functions on a limited class of data bases. NLS is well suited to low volume, experimental systems in which the user interface, ease of development, and flexibility are of major concern. The NLS limitations on file size and structural complexity imply that NLS is not well suited to production type data bases which have a large volume of data, and high transaction rates. 6b2

Although the functional capabilities of NLS can be extended to allow it to handle a larger class of data bases, I do not believe NLS could (or should) be made into a generalized data management system. The integration of NLS and existing data management systems is a more reasonable path towards full data management capabilities within an AKW environment. However certain extensions to NLS, such as those outlined below for record oriented data bases, may prove to be worthwhile in terms of long term benefits. 6b3

It should be noted that the proposed system is an extension to NLS, and as such is subject to the same limitations inherent in the current implementation of NLS. The file size limitation, statement size limitation and access speed limitations are examples. The extensions proposed herein are concerned mainly with facilitating the use of existing NLS functions rather than increasing the data management capabilities of NLS. 6b4

There are many ways in which the ARC development staff can assist future developers of record oriented data bases

management systems under NLS. The lowest level of support could be to simply publish (and maintain online) the source code for an example subsystem. The definition and implementation of a conceptually clean set of primitives within NLS, as outlined above, represents a higher level of support. A still higher level of support is outlined below. It should be noted that there exists a continuim of levels of support between the three mentioned here. 6b5

PROPOSAL: NLS Mini-Manage 6c

The following is not intended to be a detailed design description, but rather serves to give a rough idea of the kinds of extensions to NLS which are possible. This proposal concerns extensions which could simplify the task of a user who wants to implement a subsystem for maintenance of a record oriented data base. The goal is to automate as much of the process as is possible, leaving only purely application specific code for the implementer to provide. The general approach is to automatically generate as much of the source code as is possible from a user supplied data base description, and to provide hooks for special purpose user supplied code. 6c1

A "record oriented" data base can be characterized as follows: 6c2

The data base consists of a set of records, of a given number of record types. 6c2a

Each record of a given type has an identical format. This format is normally a set of well defined fields. The data (contents of fields) naturally varies from record to record. 6c2b

The structural relationships between records of different record classes are pre-defined. 6c2c

The position of each record in the data base can be computed from the data within the record. 6c2d

All modifications to the data base are performed through the specialized subsystem, which performs data validation upon these modifications. 6c2e

The end user of the subsytem is typically not an experienced NLS user, and normally uses only the data base maintenance subsystem to edit and examine the data base files. 6c2f

The NLS Mini-Manage subsystem I envision can be implemented by completing the following tasks: 6c3

1) Define a formal data base description language for describing a record oriented data base, and implement a compiler which will translate a data base description given in this language into a machine readable data structure. 6c3a

This language will be capable of defining both the structure and format of the records in the data base. Language contructs for the following functions will be provided: 6c3a1

Inversion of a set of records based upon the contents of a field. Inversion is a technique to provide multiple orderings, or multiple access keys, on a set of records without sorting the records. 6c3a1a

There are several possible choices for the internal implementation of the inversion tables. Some are: 6c3a1a1

For each inversion key define a property element in which linked lists defining the inverted order are maintained. 6c3a1a1a

Use the free space in the ring structure blocks of NLS files to maintain linked lists defining the inversion order. 6c3a1a1b

Use separate pages within the file space to maintain a hash table implementing the inversion keys. 6c3a1a1c

Use separate pages within the file space to maintain indexed sequential table implementing the inversion keys. 6c3a1a1d

Field verification content analysis patterns. 6c3a1b

Range limits for numeric fields. 6c3a1c

Hooks for a user supplied field verification procedures. 6c3a1d

Hooks for a user supplied record verification procedures. 6c3a1e

Hooks for user supplied formating routines. The user may supply several formatting routines for each type of record corresponding to several different reports which will be generated from the data base. 6c3a1f

User supplied error messages. 6c3a1g

2) Implement a "grammar generator" which takes as input the data structure describing the data base and produces as output a CML grammar which defines the user interface to the data base maintenance subsystem. The grammar will include at least the following functions. 6c3b

Create Record 6c3b1

Edit Record 6c3b2

Delete Record 6c3b3

Show Record(s) (Criteria) 6c3b4

Generate Report (Report name) 6c3b5

3) Implement a "subsystem generator" which takes as input the data structure describing the data base and produces as output source code for the x-level routines necessary for the data base maintenance subsystem. This subsystem will include calls to the user supplied procedures which are referenced in the Data Base Description. 6c3c

4) Implement commands to compile and load the data base subsystems generated by the Mini-Manage system. 6c3d

The implementation of the above functions would allow a user, without knowledge of CML or of L10 or of the implementation details of NLS, to define a data base using a formal data description language, and automatically have a custom tailored data management subsystem produced for him. This subsystem would contain the functions listed above which provide a basic set of necessary primitives. This approach is called a "generative" approach meaning that a subsystem, custom tailored to the specific data base, is generated from a data base description file. The main advantage of a generative approach over a general purpose system which interprets the data base description at run time is increased efficiency. 6c4

For more sophisticated maintenance procedures the data base designer could supply his own validation, and formating procedures, using the hooks provided by the data definition language. For still more sophisticated maintenance and query functions the data base designer could actually add to the source code produced by the subsystem generator. 6c5

COST ESTIMATES 6d

Time Estimates: 6d1

2 person months design 6d1a

2 person months implementation for the data description language compiler. 6d1b

2 person month for implementation of the "grammar generator". 6d1c

2 person month for implementation of the "subsystem generator". 6d1d

2 person months for debugging. 6d1e

2 person month for documentation. 6d1f

.2 person on a continuing basis for user assistance and maintenance. 6d1g

Thus such a facility would cost approximately one man year to develop. The major benefit gained would be the allow a much larger class of users to define and develop subsystems for record oriented data management. Currently one needs a knowledge of the CML language, the L10 language, the system architecure of NLS, and a working knowledge of the core editing procedures provided by NLS to construct such a subsystem. The implementation outlined here would allow an expert NLS user with a knowledge of a simple data description language to achieve the same result. Whether or not the increased NLS capabilities gained by such a development would be worth the implementation costs is a decision for both the current NLS user community and ARC in general on behalf of the future members of the AKW community. 6d2

< AJOURNAL, 27380.NLS;1, >, 21-JAN-76 22:32 XXX ;;;; .HJOURNAL="DSM 21-JAN-76 19:03 27380"; Title: .H1="NLS File Size Extensions"; Author(s): David S. Maynard/DSM; Distribution: /DBM([INFO-ONLY]) ; Sub-Collections: SRI-ARC DBM; Clerk: DSM; .IGD=0; .SNF=HJRM; .RM=HJRM-7; .PN=-1; .YBS=1; .PES;

The following is a discussion of the file size limit in NLS. The current limitations on file size are discussed , and a proposal for increasing this limit is presented. Historically this NLS limitation is derived from a since removed restriction of the TENEX operating system which limited a file to a maximum of 512 TENEX pages.

As currently implemented NLS files may have a maximum of 95 file pages allocatted to ring structure blocks, and 370 file pages allocated to statement data blocks. The limit to the number of statements which may be contained in an NLS file is a function of average statement size in the file. For files which have an average statement size of less than about 70 characters the limiting factor is the number of ring structure blocks available. The maximum number of ring blocks, and therefore statements, in an NLS file is 9690. For files with an average statement size of more than 70 characters statement data blocks become the limiting factor, yielding an upper limit of less than 9690 statements. The worst case is every statement having the maximum NLS statement size of 2000 characters. In this case the limit is 370 statements per file.

For most NLS applications the file size limitation has not been a major concern, however for some applications this limitation is critical. Increasing the NLS file size limit would be of considerable benefit to the builders of information retrieval systems because it allows larger data bases within a single file, and thus may relieve the implementer of the burdens and inefficiencies of handling multiple data files. There follows some examples of applications which could make use of larger NLS file capacities:

The NLS IDENT file is currently about 75% of the maximum NLS file size. The IDENT file may soon burst at the seams, unless the NLS file size limit is increased.

Currently several implementers of record oriented data management subsystems in NLS have designed their data bases with several "fields", or data elements, within one NLS statement. One reason this is done is to increase the record capacity of the data file. If the limit on the number of statements was larger the data base could be designed with each field being a statement. This allows more efficient field access, using NLS statement names, as opposed to performing string searches within a statement.

The following is a list of what I believe to be the proper set of design goals for any proposed modifications aimed at extending the size of NLS files.

1) minimal implementation cost.

2) minimal run time penalties for users of smaller files.

3) should result in a significant increase in capacity, i.e. an order of magnitude increase.

4) should be invisible both to the user and to all upper level code.

5) If the file formats are incompatible, NLS must handle both formats, allow cross file edits, and provide for automatic conversion from old file formats to the new format at update time.

The following changes are proposed to the NLS file system to satisfy the design goals listed above.:

1) Change the interpretation (but not the length) of the PSID and PSDB fields within the file system, and within stid's.

Current:

First nine bits represent a file page number, and the next nine bits represent a word offset from the beginning of the page to the start of the element (structure or data element).

Proposed:

Conceptually each file page is divided into 64 8 word "chunks". The first 12 bits of an PSID or a PSDB is a file page number. The next six bits represent a "chunk index" at which the associated structure block or data element begins within the page. This requires structure data elements to begin an integral multiple of eight words from the beginning of the page. Note that "chunks" map one to one onto ring structure elements, however a data block may span several chunks.

2) increase the size of the page header blocks from 2 words to 8 words.

3) increase the size of ring structure blocks from 5 words to 8 words.

Note that this change also allows more free space in the ring blocks for possible future expansion.

4) Increase the number of possible structure pages from 95 to 1024, and increase the number of possible data pages from 370 to 4096. This implies increasing the size of the ring block status table, and the data block status table in the file header.

The changes outlined above would result in an order of magnitude increase in the maximum NLS file size. All potential user's of these large NLS files should understand that functions which are time consuming in NLS, i.e. functions which must run the ring blocks or search data blocks, will be correspondingly slower for larger files. The time for this type of function increases approximately linearly with file size.

The estimated costs associated with the possible implementation of this

proposal are as follows:

Implementation Costs:

I estimate approximately one person month for implementing the changes to the file system, another two person weeks for testing and debugging, plus occasional maintainence over a two month acceptance testing period.

Additional Costs:

The proposed file structure is not compatible with existing NLS file structure. Therefore a new file structure version would be defined. The version of NLS (8.6?) with the proposed changes could manipulate both normal NLS files and long NLS files. NLS 8.5 and previous versions could not manipulate long NLS files. There will also be a small (I believe insignificant) run time penalty for manipulation of old files under NLS 8.6. The update file command could (optionally?) convert old files into the new format.

NLS File Size Extensions

The following is a discussion of the file size limit in NLS. The current limitations on file size are discussed , and a proposal for increasing this limit is presented. Historically this NLS limitation is derived from a since removed restriction of the TENEX operating system which limited a file to a maximum of 512 TENEX pages. 1

As currently implemented NLS files may have a maximum of 95 file pages allocatted to ring structure blocks, and 370 file pages allocated to statement data blocks. The limit to the number of statements which may be contained in an NLS file is a function of average statement size in the file. For files which have an average statement size of less than about 70 characters the limiting factor is the number of ring structure blocks available. The maximum number of ring blocks, and therefore statements, in an NLS file is 9690. For files with an average statement size of more than 70 characters statement data blocks become the limiting factor, yielding an upper limit of less than 9690 statements. The worst case is every statement having the maximum NLS statement size of 2000 characters. In this case the limit is 370 statements per file. 2

For most NLS applications the file size limitation has not been a major concern, however for some applications this limitation is critical. Increasing the NLS file size limit would be of considerable benefit to the builders of information retrieval systems because it allows larger data bases within a single file, and thus may relieve the implementer of the burdens and inefficiencies of handling multiple data files. There follows some examples of applications which could make use of larger NLS file capacities: 3

The NLS IDENT file is currently about 75% of the maximum NLS file size. The IDENT file may soon burst at the seams, unless the NLS file size limit is increased. 3a

Currently several implementers of record oriented data management subsystems in NLS have designed their data bases with several "fields", or data elements, within one NLS statement. One reason this is done is to increase the record capacity of the data file. If the limit on the number of statements was larger the data base could be designed with each field being a statement. This allows more efficient field access, using NLS statement names, as opposed to performing string searches within a statement. 3b

The following is a list of what I believe to be the proper set of design goals for any proposed modifications aimed at extending the size of NLS files. 4

1) minimal implementation cost. 4a

2) minimal run time penalties for users of smaller files. 4b

3) should result in a significant increase in capacity, i.e. an order of magnitude increase. 4c

4) should be invisible both to the user and to all upper level code. 4d

5) If the file formats are incompatible, NLS must handle both formats, allow cross file edits, and provide for automatic conversion from old file formats to the new format at update time. 4e

The following changes are proposed to the NLS file system to satisfy the design goals listed above.: 5

1) Change the interpretation (but not the length) of the PSID and PSDB fields within the file system, and within stid's. 5a

Current: 5a1

First nine bits represent a file page number, and the next nine bits represent a word offset from the beginning of the page to the start of the element (structure or data element). 5a1a

Proposed: 5a2

Conceptually each file page is divided into 64 8 word "chunks". The first 12 bits of an PSID or a PSDB is a file page number. The next six bits represent a "chunk index" at which the associated structure block or data element begins within the page. This requires structure data elements to begin an integral multiple of eight words from the beginning of the page. Note that "chunks" map one to one onto ring structure elements, however a data block may span several chunks. 5a2a

2) increase the size of the page header blocks from 2 words to 8 words. 5b

3) increase the size of ring structure blocks from 5 words to 8 words. 5c

Note that this change also allows more free space in the ring blocks for possible future expansion. 5c1

4) increase the number of possible structure pages from 95 to 1024, and increase the number of possible data pages from 370 to 4096. This implies increasing the size of the ring block status table, and the data block status table in the file header. 5d

The changes outlined above would result in an order of magnitude increase in the maximum NLS file size. All potential user's of these large NLS files should understand that functions which are time consuming in NLS, i.e. functions which must run the ring blocks or search data blocks, will be correspondingly slower for larger files. The time for this type of function increases approximately linearly with file size. 6

The estimated costs associated with the possible implementation of this proposal are as follows: 7

Implementation Costs: 7a

I estimate approximately one person month for implementing the changes to the file system, another two person weeks for testing and debugging, plus occasional maintainence over a two month acceptance testing period. 7a1

Additional Costs: 7b

The proposed file structure is not compatible with existing NLS file structure. Therefore a new file structure version would be defined. The version of NLS (8.6?) with the proposed changes could manipulate both normal NLS files and long NLS files. NLS 8.5 and previous versions could not manipulate long NLS files. There will also be a small (I believe insignificant) run time penalty for manipulation of old files under NLS 8.6. The update file command could (optionally?) convert old files into the new format. 7b1

Compiling NLS System Files and Loading New NLS Systems: The use of TASKS and other techniques for System Programmers

This document describes official policy for NLS system programmers as of 22 January 1976. (In fact, most of what is discussed has been used for several years. However, newer programmers have not been told of some of these techniques and documentation has been at best scattered and at worst sparse.) Modifications should be sent to all System Programmers and entered into the Journal. The section on suggested techniques for modifying the system (and bringing up new systems) is subject to approval by the Applications Programmers.

Compiling NLS System Files and Loading New NLS Systems: The use of TASKS and other techniques for System Programmers

Introduction 1

The following describes the facilities for compiling files and loading new systems which have been in effect for some time among the ARC system programmers. Newer programmers have been unaware of these procedures except through the ARC oral tradition and, while this may be of interest to programming anthropologists of the future studying the early Augmentific Age, the abolition of folklore in favor of the written truth cannot help but reduce some chaos. While these procedures have been documented before in earlier Journal messages, this note should be more up to date and collects previously scattered information. Hopefully, modifications will be promptly entered as supplements. Copies of the most recent version should be furnished to all new programmers as part of their orientation. (Important restrictions on the techniques include the necessity of using TENEX syntax for file names, a remnant of earlier days which has never been corrected. Such limitations should be fixed and the changes documented.) 1a

The system commands available are fully described along with the relevant files (e.g., (nls, tasks,)) and rules for modifying the system in a manner which does not overly restrict the actions of others also working on other parts of that system. If these rules are followed, a fixed record of changes is available; this record is often useful in tracking down the causes of side effects and in restoring disappearing code-- an unlikely, but not impossible, situation as any old timer will attest. 1b

These methods may also prove to be useful in the design of similar systems for NSW. 1c

Please note that some commands which I deem to be relatively useless are not described fully (notably "Check Results of Tasks"). This is just a personal prejudice which could be corrected by fans of these neglected commands. Those commands described, however, should be of universal popularity where the universe is made up of the NLS systems programmers. Note that there is no excuse for not maintaining the records in the changes branches! 1d

The Structure of (nls, tasks,) 2

The following branches are contained in the file (nls, tasks,). Some of them are used or created by automatic processes such as the "Run Tasks" command in XXX while others are used by programmers to maintain a record of changes made in any of the evolving versions of the NLS. More detailed discussions may be

found in the remaining sections of this file. Examples are copied from the actual file. My comments are enclosed in per cent signs. 2a

(todo) list of tasks for NLS background process to do. 2a1

Compile inpfbk 2a1a

Compile <nic-nls>inpfbk 2a1b

Print inpfbk 2a1c

% This plex will be read by the program which executes the comands listed here when the "Run Tasks" command is executed. Because it is a program intolerant of syntactical errors, the form, described below, must be followed exactly. % 2a1d

% Changes branches % 2a2

% Changes to systems are listed here along with notes indicating when the system has been brought up as the running system. Changes in only some of the systems (e.g., only NIC-NLS) are listed only in the corresponding branches. When they are added to the other systems, they should also be recorded in the other appropriate branches. The ident and date should also be noted. The most recently added items are closest to the top of the plex. % 2a2a

(NIC-NLS-CHANGES) Record here changes to NIC-NLS source files.... 2a2b

(nic-nls,psuserop,xuoprint) KJM 20-JAN-76 2a2b1

Added code to make tab setting changes take effect immediately on TNLS (already in immediate effect on DNLS). 2a2b1a

(NLS-CHANGES) Changes to files in <NLS> that have not been copied to Nine. 2a2c

(nls, filmnp, freplist) (nls, filmnp, delprop) (nls, filmnp, reprop) (nls, filmnp, freprop) (nls, utilty, filesc) HGL 30-DEC-75 2a2c1

Changed code so replacing a property block does not delete the associated inferior tree. FREPROP no longer frees the inferior tree of the property: it returns the stid of the inferior tree to the calling

procedure which must do this by calling FREINTREE, as is done by DELPROP and FREPLIST, or relink it to a new property by calling INSITREE if desired as is done in REPROP and FILESC. 2a2c1a

(NINE-CHANGES) Record here changes to NINE source files. 2a2d

(nine,auxcod, showfe) LLG 23-DEC-75 15:29 2a2d1

Added this procedure. Performs the PCP call to FE "show" routine. 2a2d1a

(NSW-CHANGES) Changes to files in <NSW-SOURCES> to be documented. 2a2e

(DONE) Tasks which Are Done (CLEAN THIS OUT ONCE IN A WHILE) 2a3

% These statements are created by the "Run Tasks" command and by the "Process Commands" branches for loading files. When Tasks compiles a file on the basis of a command in the TODO branch, it moves that statement from the top of the TODO subplex to the top of the DONE subplex. If the compilation is successful, the date and time of the completion of the compilation is noted. If the compilation is unsuccessful, it notes only the number of errors. 2a3a

Compiler diagnostics may be found in the text file specified by the "Run Tasks" command feedback. (It is unfortunate that the user cannot specify the file name.) The name of the file with the diagnostics is <nls>u-out.txt;. It is generally obvious which version you will be interested in-- hopefully the most recent. It may be examined by copying it to TTY: at the EXEC or by doing a "Copy Sequential" in NLS. If you are running tasks and there is an error in one of the files, the error should be corrected before loading if possible. You may have to contact the person whose file was changed; if it is an obvious syntax error, fix it yourself. 2a3a1

The lastload branches are inserted at the completion of loads driven by the Process commands branches described below. Loader diagnostics may be found saved in the TXT file named in the corresponding Process commands branch. % 2a3b

(LastLoad) Last GNLS Load: 15-JAN-76 23:13 2a3c

Compile inpfbk 2a3d

Completed at 15-JAN-76 23:09 2a3d1

Compile <nic-nls>frontend 2a3e

22-DEC-75 13:02 1 Errors 2a3e1

(LOAD) Process commands branches for loading new systems. 2a4

% The branches with the names listed below are described in detail in a later section of the file. Their names are fairly self-descriptive. If you do a load using them, the loader diagnostics and maps are saved in a file and the load is noted in the DONE branch. Neither of these benefits is available if you do your loads via RUNFIL at the EXEC. Thus your fellow programmer has the benefit of being able to check the load map for information about the experimental system which is being debugged. % 2a4a

(make-gnls) 2a4b

(make-rnls) 2a4c

(make-xporgen) 2a4d

(make-porgen) 2a4e

(make-nls9) 2a4f

(printlist) 2a5

% This branch contains a plex of the following form for every file in NLS. It may be copied to the DONE branch in order to get a complete listing of NLS in the special listing format simply be issuing the "Run Tasks" command. % 2a5a

Print <nls>ADRMNP 2a5b

(nic-nls-printlist) 2a6

% This branch is simialr to the previous one except it is for files in the <nic-nls>directory. There should be a branch created for <nine> and <nsw-sources> as well. % 2a6a

Print <nic-nls>ADRMNP 2a6b

Compiling and Printing Groups of Files-- The "Run Tasks" Command in Subsystem XXX 3

Before describing how to use TASKS I wish to stress the need to update the (nls, tasks,) file whenever you have finished using it, whether to run the tasks command, load a system, insert a command statement in the TODO branch, or enter a record of a change. Others must use the file, and they will not be able to do so if you have left it locked. 3a

To batch requests to compile or print files, commands are placed in the (nls, tasks, TODO) branch. The syntax of these requests is fairly restrictive: Capitalization of the command word but not the file name is important as well as the presence of spaces. If the file to be compiled or printed is in the NLS directory, the directory need not be specified. Otherwise, the directory must be noted in TENEX syntax. Some examples are: 3b

Compile fintnls 3b1

Compile <nic-nls>fintnls 3b2

Print fintnls 3b3

If a file is to be compiled, the FILE statement of that file must be of the following form. Note that ALL NLS system files are in this form. The rel-file name is the one specified in the FILE statement comment. Because of the current state of the TASKS command, the format must be exact. TENEX syntax is necessary. Thus we have the redundancy of an additional comment in NLS link syntax because of the requirements of the commands in the programs subsystem! This hopefully will be fixed in the future. Note also that the compiler is also specified. If no directory is noted, the default system compiler is used. Experimental compilers must have the directory specified. Examples of the FILE statements are: 3c

FILE fintnls % L10 to <nic-nls>FINTNLS %% (L10,)
(nic-nls,FINTNLS,) % 3c1

FILE nislanguage% CML <rel-nls>SYNTAX % % (CML,)
(rel-nls,SYNTAX.rel,) % 3c2

FILE bintnls % <ARCSUBSYS>XL10 to <RELNINE>bintnls %%
(arcsubsys, XL10,) (RELNINE,bintnls,) % 3c3

To get the process started, you must issues the "Run Tasks" Command in subsystem XXX. (XXX should be an attached subsystem for all NLS system programmers; it should be set as such in your user profile. Other useful commands in XXX permit you to enter TENEX DDT; less generally useful commands are also there to start

the Journal process, etc.) It was at one time possible to have the system detach your job once the process started. Thus you could then log in again and not tie up a terminal. (The job would run detached.) I don't recommend this procedure because of various problems in the system. If you just let tasks run, you may see the statements disappear dynamically as the system compiles them one by one (assuming you have the file loaded to the TODO branch on a display). When the process is complete, you will receive the message "Tasks completed". 3d

If the system crashes in the middle of a compilation, it will not have been moved to the DONE branch. 3e

Notations of successful or unsuccessful compilations are inserted down from the commands as noted in the discussion of the DONE branch above. If the source file is locked, it will not be compiled by TASKS (unless it is locked by the person running tasks!) and the message "Cannot open file" will appear in the DONE branch. This feature prevents the compilation of a file which is in the process of being modified. 3f

Compiler diagnostics are available as noted above in the discussion of the DONE branch. 3g

A little used command in XXX may be used to "Check" if there were any errors in the compilations before loading. This command is included in each of the "Process Commands" branches for loading subsystems. I personally prefer to check such information myself and fix syntax errors by hand. I then go on to load the system I want by using the appropriate branch. 3h

Loading System Files-- The Process Commands Branches in (nls, tasks,) 4

One may do a load of an NLS system (or output processor) by doing a "Process (commands from) Group" on the group of commands in the desired branch in TASKS. All of these branches have the same form and accomplish the desired task by starting up a lower fork and doing the equivalent of a RUNFIL. The commands file has the same format as any used in the RUNFIL command at the TENEX EXEC with one important exception: there must be the word QUIT at the end to get you out of the lower level fork and back to NLS. Typing control-I while the process is running will not give you any information since it is directed to the lower level fork. In fact typing anything while the load is in progress is probably dangerous because of limitations in the system which were introduced when it was written several years ago at a time when the TENEX fork manipulation mechanisms were even more primitive than they are now. 4a

while commands in these branches also permit you to run TASKS, I don't use them for this purpose: I prefer to separate the jobs. Thus I do "Process Group" rather than a "Process Flex". 4b

There are several advantages to using this process rather than doing a RUNFIL at the TENEX EXEC: 4c

A permanent record of the load diagnostics and map is available for viewing by other programmers. 4c1

The date and time of the load is recorded in tasks. 4c2

A copy of the load diagnostics is put out to the lineprinter. (This was once useful, but since the loader diagnostics have EOLs rather than CR-LFs, the load maps try to appear all on one line on the ELF printer: You can, however, do a "Copy Sequential" in NLS on the load map file (the name of which is specified in the process commands branch) or view that file on your terminal at the TENEX EXEC by copying to TTY: 4c3

A typical Process commands branch from (nls, tasks,) appears below: 4d

```
(make-gnls)                                                    4d1

   execute programs attach subsystem xxx                       4d1a

   execute xxx run tasks                                       4d1b

   execute xxx check                                           4d1c

   ;; ** DO NOT type ANYTHING while load in progress  **       4d1d

   execute programs run tenex (system,exec,)file
   (rel-nls,load-map-gnls.;,)from (rel-nls,grunlor.txt,)yes    4d1e

   load file (nls,tasks,)                                      4d1f

   insert statement (nls,tasks,done)d(LastLoad) Last GNLS Load: 4d1g

   insert time (nls,tasks,done.date)                           4d1h

   update file                                                 4d1i

   copy file
   (rel-nls,load-map-gnls.;0,)(arcprinter,load-map-gnls.*;*,)  4d1j

   ;  *** LOAD COMPLETED  ***                                  4d1k
```

Similar branches exist for NLS (creates <rel-nls>gnls), NIC-NLS (creates <nic-nls>rnls), NINE (creates <relnine>benls), XPORGEN (creates <xporgen>xoutprc) and PORGEN (creates <porgen>xoutprc). There ought to be one for loading the front end as well! 4e

NLS versions of all the commands files for the RUNFIL exist. If you wish to change the runtil, you should change the NLS version and then do an "Output Assembler". (You probably shouldn't change them blithely because of space restrictios and the necessity for having files in specific locations in the address space.) The relevant files are: 4f

(nls, grunldr,) used to make (rel-nls, grunldr.txt;,) 4f1

(nic-nls, runldr,) used to make (nic-nls, runldr.txt;,) 4f2

(nine, loadnls,) used to make (relnine, loadnls.txt;,) 4f3

(porgen, opload,) used to make (porgen, opload.txt;,) 4f4

(xporgen, opload,) used to make (xporgen, opload.txt;,) 4f5

Again, there should be a pair for the frontend listed here. 4f6

Recording Changes and Additions to the System-- The Changes Branches in (nls, tasks,) 5

For your own benefit as well as for the sanity of your fellow NLS system programmers, you should record relevant changes to the system (from minor bug fixes to major overhauls) in the appropriate change branches in TASKS. Examples of some recent records are contained above. These notes are very useful in the creation of documentation and in analyzing side effect bugs. 5a

Suggested Techniques for Modifying the System 6

Minor bug fixes (changes to the system which take less than a few days) may be made directly to the source code. After the changes have been put in, compile the file using TASKS and then load the system (after perhaps negotiating with others who have files in the same system waiting to be compiled.) After successful testing, note the change in the appropriate branch. Send a message through the Journal to the appropriate documentation person if new documentation is necessary. Send a message to the appropriate Applications programmer to have them bring up the system as the running version on all relevant machines. Changes to <nic-nls>rnls need great care since it goes to OFFICE-1. Changes to <rel-nls>gnls appear in the running system at ISIC. 6a

Major additions or changes should probably be done in your own copy of the relevant source code file which you may debug using the Programs Subsystem. However, because others may wish to fix bugs in the same file, you should maintain a lock on the original so any other programmer will be denied access by the system. After personal negotiations (so the other programmer's changes are also placed in your copy of the file) you could let the other person have access to the system file by updating it. You must then place a lock on the file again. Warning messages may be placed in TASKS to let other programmers know about pending major system disruptions. 6b

As a rule, developments should remain user subsystems until they have been thoroughly checked out. Only then should you incorporate the changes in experimental NLS systems. Further testing should be carried out before the additions or modifications make their way into supposedly more stable systems such as <nic-nls>rnls. 6c

< AJOURNAL, 27387.NLS;2, >, 23-JAN-76 16:25 HGL ;;;; Title:
Author(s): Harvey G. Lehtman/HGL; Distribution: /NPG([ACTION])
SRI-ARC([INFO-ONLY]) ; Sub-Collections: SRI-ARC NPG; Clerk: HGL;
Origin: < LEHTMAN, LOADPROC.NLS;3, >, 21-JAN-76 11:16 HGL ;;;;####;

JEW 27-JAN-76 17:42 27426
Comments from John Day on the NCC DPS Paper
Location: (AJOURNAL, 27426, 1:w)

<<<<<<<<<<<<<<<<< Text of Cited Document Follows >>>>>>>>>>>>>>>>>

MESSAGE

Jim, I read your RFC with great interest. I tend to agree with you about the general utility of the procedure call model for protocol design; however, there are several points in your paper which weaken you argument considerably.

Although I am sure you did not intend it, paragraph 2c1 seems to imply that the Telnet protocol defines connection conventions and the character set, but not the commands. I assume that you are pointing out the lack of a Common Command Language in the Net for often used functions. It is a valid point, but has little to do with the Telnet protocol itself.

A bit later you note that the general format for protocols beyond Telnet have a command followed by a single parameter. From the context, I assume you are referring to FTP and RJE (the only other possiblities are Mail and Graphics) of which only FTP has this format. This hardly seems sufficient to make a generalization from. Also on the point of the single parameter, you seem to indicate (and rightly) that this limitation is restrictive and should be relaxed. However, if I remember correctly from the last FTP meeting in March 1973, the reason for adopting the single parameter form was the difficulties of providing a way to report parameter errors or incompatible combinations of parameters in a machine readable form that was helpful, would not get out of hand, and everyone felt comfortable with. Although several possibilities come to mind to address this problem, I find it a major flaw in your paper that after pointing out the restriction (and implying that the position being taken in this paper is better) you fail to address the issues which lead to the restriction originally. The problems could be dealt with in relatively short order and would lend more credence to you overall argument.

If you would like more evidence on your side with respect to modeling communications as procedure calls, you might look at Balzer's article "An Overview of the ISPL Computer System Design" in CACM Feb. 1973 Vol. 16 No. 2. In which he implements ports by means of co-routines.

A real nit-pick. I was a little disappointed in the generality of your data types. Why should integers and indices be restricted to 32 and 16 bits respectively? For that matter why distinguish the two? There are several good data type formalizations floating around in the literature that you might want to look at.

All in all you may find that these are rather minor points, but my own feeling is that they detract rather heavily from the flow of the argument that is being made in favor of the procedure call model. Its almost like one has just read the proof of a theorem; and you feel that the theorm is true, but your not sure you believe the proof.

Take care, John

REPLY

John-- Thanks for your comments on my NCC paper. Assuming the paper

is accepted for inclusion in the Proceedings, I will try to incorporate your comments along with the referees'. In response to some of your specific points:

(1) You're right, I'm not criticizing the TELNET protocol; it does what it was designed to do, i.e. provide a vehicle by which a user can address arbitrary commands to a remote process. Rather, I'm simply noting that effective process-to-process communication (with no human in the loop) requires that the syntax and semantics of commands be fixed.

I chose to talk about process-to-process protocols as if they were command languages because that's the way they in fact developed in the ARPANET (i.e. TELNET was chosen as their foundation). Although I have never bought this design strategy and believe it has hindered protocol development, it is the one that was chosen.

(2) The family of protocols to which I refer in the paper consists of FTP, RJE, and Mail (both as it exists within FTP and as (for example) I proposed in RFC 524). Each of these protocols employs the single-parameter-per-command model. Although the family is small (just as the total number of official network protocols is small), I'm sure the philosophy behind it would tend to be adopted in other application domains (at least those amenable to it), were there any such work being funded these days within the ARPANET.

There are, of course, as you note, other applications protocols (e.g. graphics) which are structured entirely differently; I will make this clearer in the paper.

(3) I don't recall the difficulty of reporting parameter errors being a factor in limiting the protocols to one parameter per command, although that was a long time ago and I may have forgotten. The reason I do remember was the difficulty of delimiting one parameter from another, i.e. of finding a delimiter that would never appear in a parameter. The decision was therefore made to allow just one parameter per command, from which only CR LF were excluded. This, of course, was simply laziness on our part, since there are simple ways of surmounting the difficulty (e.g. preceeding occurences of the delimiter within the parameter by an escape character). The insistence that the protocol be typeable by a TIP user may also have had something to do with the decision. In any case, the data typing scheme proposed in the paper solves this problem, of course. But since I didn't even describe the problem in the paper, I, of course, didn't make a big thing about its solution. Perhaps the paper should contain some more background information in this area (but I find it embarrassing to even talk about).

Regarding the difficulty of reporting errors in a multi-parameter-per-command model, I don't understand what the problem is. FTP permits an error number to be returned to the invoking process, in which the specifics of the error (e.g. which parameter was at fault) can easily be encoded; the protocol proposed in the paper does likewise. Every programming environment with which I have ever dealt takes roughly this same approach. There error numbers are employed to report the failure of procedures whose calling sequences can be complex indeed, and the mechanism has always seemed perfectly adequate.

(4) The problem of imposing ceilings on the length of a character

string, the size of an integer, etc. is one that can only be solved by simply biting the bullet and picking numbers. Whether 32 bits as the width of an integer was the right choice, I don't know. If one instead picks 36, thereby allowing a 36-bit machine to send such an object to (for example) a 16-bit machine, the latter will have major problems trying to manipulate it. Thirty-two seemed a nice choice at the time, since objects of that size can easily be manipulated by both 32- and 36-bit machines, and require an even doubleword for their representation in 16-bit machines.

Type INDEX was NOT intended simply as a more efficient way of encoding small INTEGERs, a goal that can be achieved much more cleanly by defining two transmission formats for INTEGERs, one for use with very small numbers and the other for use at other times. INDEX was introduced as a distinct data type as a means of formalizing what I have found to be a very common class of data objects, namely, small positive integers (the 2**15-1 upper bound being somewhat arbitrary) used as handles or identifiers for other objects. What I was aiming at was a convenient way of restricting certain procedure arguments or results to such objects. Perhaps the right thing to do is simply describe such parameters as "an INTEGER in the range [1, 1**15-1]", although "an INDEX" is certainly more convenient.

Jonn, you may be interested to know that Al Vezza and others at MIT have also designed a set of data types and transmission formats (for a mail protocol they're working on) that I believe will be described in a forthcoming RFC. Their protocol takes much greater pains to accomodate large objects and to minimize representation sizes, and is to my proposal what a Cadillac is to a VW. --Jim

<<<<<<<<<<<<<<<<<<<<<<< End of Cited Document >>>>>>>>>>>>>>>>>>>>>>

Comments from Carl Sunshine on the DPS Papers

018 MESSAGE

019 Jim, I just had the pleasure of reading your two DPS papers which I found most lucid and enlightening. A couple of questions occurred to me, however.

020 In your NCC paper, you state that "unlike the substance of the Protocol, however, the Model has already proven quite controversial in the ARPANET community." (par 4a8) Could you point me to some RFC's or whatever that relate to this controversy? I am most interested in what people find palatable or distasteful about your formulation. In particular, your selection of procedure-like models rather than process- or command-like models bears further discussion in my mind. Is that what the controversy was about?

021 In your second paper (27250), it took me a while to understand where the extensions you discussed in part 3 get implemented. If I have it straight, the first extensions discussed in each section are of a "server" nature -- that is they are part of the run time environment, and respond to requests from remote processes (actually the remote process' RTE) to do something to local processes. INIPROCESS, ALOPORT, etc. are in this class. The second type procedures you described exploit these "server" type procedures to provide a "user" type service to their local processes. So a process calls CRTPROCESS in its local RTE which talks to INIPROCESS in the appropriate remote computer's RTE to do the right stuff. Is this all correct? If so, I think you should make the distinction between the server and user type RTE procedures clearer. You also might make it very clear that all the extensions to the model you present in part 3 are "system procedures" as you say in part 2, either of the server or user type.

022 I'm looking forward to our session at NCC. Carl Sunshine

023 REPLY

024 Carl-- Thanks for your comments on my DPS papers. Assuming the papers are accepted for publication, I'll try to incorporate your comments along with the referees'. In response to your specific questions:

025 (1) The major point of controversy within the ARPANET has indeed been the procedure call model. See:

026 (a) "A Commentary on Procedure Calling as a Network Protocol"
(RFC 684) by Rick Schantz of BBN.

Comments from Carl Sunshine on the DPS Papers

027 (b) "Some Comments on the Procedure Call Protocol" by Raj Kanodia of MIT (to my knowledge, this was never published as an RFC; perhaps Raj can supply you with a copy). 027

028 (2) In the paper, I tried to carefully distinguish between the "primitives" that a run-time environment offers its local applications program, and the "system procedures", which one run-time environment offers to another via the Protocol, via which those primitives are implemented. As you finally concluded, CRTPROCESS, for example, is a (local) primitive while INIPROCESS is the (remote) system procedure required for its implementation. All the primitives (which correspond to your term, "user type RTE procedures") are summarized in Appendix C, all the system procedures (your term, "server type RTE procedures") in Appendix B. 028

029 As indicated in paragraph 2d4, I employed the same format for describing both primitives and system procedures, hoping that the category into which particular primitives or system procedures fell would be apparent from the context. Sorry for the confusion; I'll try to devise some notational difference to distinguish the two. 029

030 Carl, thanks again for your comments. After proof reading something 500 time yourself, it begins to make sense no matter what it says. --Jim 030

Comments from Ken Pogran on the NCC DPS Paper

04 MESSAGE

05 Jim, I was just browsing through RFC707, and came up with the following comment: In 2c7 on page 5, you describe the RNFR/RNTO FTP command sequence, saying that "Since the general command format admits but a single parameter, multiparameter operations must be implemented as sequences of commands". That's generally true; I regard it as an example of narrow thinking on the part of the folks who first specified the FTP, which has been blindly perpetuated by all who followed. Note that in the latest go-round, we have some mode-setting commands which take TWO parameters! (I forget which, but it's there).

06 The example of RNFR/RNTO can only prompt the reader to say "Oh, what a bad design!". Which, of course, is the point, for you pick up on this in 3a2 (1). But I think it's the weakest point, for we could design the two-command sequence, with its attendant round trip delays, out of the protocol. We CAN'T design "parameter types other than character strings" into the FTP, nor can we "return results in a command response". I think these are by far the more substantive problems with the FTP's crude command/response discipline; RNFR/RNTO is merely a rough edge that, I think, isn't worth mentioning in a technical paper.

07 Good paper, by the way!
Regards, Ken

08 REPLY

09 Ken-- Thanks for your comments on the DPS paper.

010 The most recent FTP spec contains two commands with more than one parameter: SOCK, which permits both host address and socket number to be specified; and ALLO, which accepts both maximum file size and maximum record size. Both are "easy" commands to define, since both of their parameters are numeric and therefore a space can readily be employed as a delimiter between them. I considered mentioning these two commands in the paper, but decided not to (perhaps a mistake), since I was interested in describing the general philosophy behind the protocols (i.e. the rule, not the few exceptions to it).

011 Needless to say, RNFR/RNTO is not an isolated case, since in FTP we also have:

012 USER/PASS/ACCT instead of LOGIN (user, pass, acct)

013 RETR/TYPE/STRU/MODE instead of RETR (pathname, type, stru, mode)

Comments from Ken Pogran on the NCC DPS Paper

014 STOR/ALLO instead of STOR (pathname, filesize, recsize) 014

015 But you're right that this defect of the protocol would be the easiest to remove. --Jim 015

WHAT DPS IS

The Distributed Programming System (DPS) developed by SRI's Augmentation Research Center (ARC) is a software framework that encourages resource sharing within a computer network by facilitating the construction of distributed systems. It involves the following three elements:

A MODEL A high-level, application-independent model of distributed systems lays the foundation for a network-wide virtual programming environment in which hierarchically arranged processes can communicate with one another at the procedure call level.

A PROTOCOL A network-wide Procedure Call Protocol (PCP) implements the model by appropriately regulating the dialog between two processes interconnected by means of a network inter-process communication channel.

INTERFACE SOFTWARE Interface software provided by each installation and link loaded with (or otherwise made available to) an applications program employs the protocol to provide the program with high-level process manipulation primitives, thus effectively extending its local operating system to embrace processes anywhere in the network.

FOR MORE INFORMATION

The Distributed Programming System is described in a set of four documents, of which the present document is one:

INITIAL MOTIVATION The first paper provides important initial motivation for developing a procedure-call-oriented interface between distant processes:

A High-Level Framework for Network-Based Resource Sharing, December 23, 1975 (SRI-ARC Catalog Item 27197).

MODEL DESCRIPTION The second paper, a companion to the first, more fully develops the DPS model, suggesting standards for other common forms and aspects of process interaction:

Elements of a Distributed Programming System, January 5, 1976 (SRI-ARC Catalog Item 27250).

PROTOCOL DEFINITION The third document is intended primarily for systems programmers and provides a detailed description of the Procedure Call Protocol:

DPS-10 Version 2.5 Implementer's Guide August 15, 1975 (SRI-ARC Catalog Item 26282).

TENEX INTERFACE SOFTWARE DESCRIPTION The final document, intended primarily for applications programmers, contains a detailed description of interface software implemented by ARC for PDP-10s running the Tenex operating system:

DPS-10 Version 2.5 Programmer's Guide August 13, 1975 (SRI-ARC Catalog Item 26271).

JEW 28-JAN-76 19:09 27446

This one page document will be inserted as a preface into each of the four DPS documents to be distributed. Output process for best results.

This file is designed to give you a substantial, concrete introduction to NEWRETRIEVE commands. Provided you load a similar file with your directory as a plex you may actually "execute" the examples. Of course, you can change what content is being searched for. I reccommend that these examples be executed on a TI to get a hardcopy listing of the noise words.

First, you must load the NEWRETRIEVE subsystem. Currently, it has the nickname RULES (Retrieval User Language for Evaluating Sets) <cornish, rules.subsys, >.

Assuming you are in the RULES subsystem, to "execute" a group, branch or statement of RULES commands, simply enter "Execute(command in) Base Process (commands) ...." indicating the desired group, branch or statement. The commands will be executed, leaving you still in RULES. All the retrievals will appear in this file.

Thus, the second function of this file is to be a scratch pad for the various example commands.

Finally, this file serves as an excellent test base.

Summary of NEWRETRIEVE

NEWRETRIEVE has been evolving now for several months.

Here are brief descriptions of the commands to date.

Define

The Define command acts as a declaration. You may define either a form or a criterion. Within a define command, you may reference previously defined forms or criteria. NOTE THAT WHEN YOU QUIT THE SUBSYSTEM, YOU LOSE ALL DEFINED FORMS AND CRITERIA. Thus, I have found it exceeding convenient (modulo the hassle of writing command branches) to collect all my declarations into command branches.

Evaluate

You may call for the evaluation of a previously defined form or criterion. When you evaluate a criterion at a statement, TRUE or FALSE is displayed in the TTY simulation window depending whether or not the statement "satisfies the criterion". When a form is evaluated at a statement, a structure is generated which is inserted to follow a statement you select with a level adjust you select.

Evaluate form-name (at) SOURCE (to follow) DESTINATION LEVADJ CONFIRM

Evaluate criterion-name (at) SOURCE CONFIRM

Viewspecs

Note that now, viewspecs are considered part of a form. When you define a form you are prompted for the viewspecs you wish associated with the form. At a later time you may invoke

Set Viewspecs (for form) form-name VIEWSPECS

should you wish to alter the viewspecs. Setting these viewspecs does NOT alter the viewspecs of your current display area.

Show

To display the declarations of the various forms and criteria you have defined, I have include the command

Show (definition and complexity of) Form form-name CONFIRM

Show (definition and complexity of) Criterion form-name CONFIRM

"Complexity" is a measure of the hairiness in some sense of a form or criterion. It is the number of sequences required at any one time to evaluate the form or criterion.

Recall that in NLS 8 and 8.5, no more than 4 sequences can be open at any one time.

Flush

I regard this command as an interim solution.
Under rare circumstances, errors will leave open a sequence or two. Thus I have included the command

Flush (all unclosed forms) CONFIRM

I plan to include a command which will institute the evaluation of a specified criteria as the "content analyzer" for the current display area. Also, I plan a command which will institute the evaluation of a specified form as the "sequence generator" for the current display area.
This would allow the easy (fingers crossed) use of forms and criterias in other areas of NLS, such as Output Quickprint.

Consider the following examples

A search across files for content. The northwest passage!!

Suppose that by doing a Copy Directory UNDER the statement (files) I get a plex of links to all my (unarchived !!) files. How can I extract from this plex, all links which point to files "about" NEWRETRIEVE? Here you may of course substitute your own content.
First off, I must extract from the plex all the links to NLS files. Play along and pretend you don't know (I didn't) that you could get them with an option to the Copy Directory command.
Thus I define one criterion which is simply a pattern based on the extension of an NLS file.
Then I define the form nlsfiles which "extracts" from ANY plex all statements containing somewhere the content ".NLS;".
Finally, to perform the extraction on the PARTICULAR plex I "evaluate" the form nlsfiles at the statement (files.d).
Again, this means that the SPECIFIC plex to extract from is the plex at (files.d).

Note that while forms names must be unique as well as criteria names, RULES allows the name of the form and the criterion and the destination to be all the same.

(nls)

Define Criterion nlsfiles<CA>Pattern [".NLS;"] ;<CA>
Define Form nlsfiles<CA>Extract Plex <^N><CA>Name nlsfiles<CA><^N><CA>
Evaluate Form nlsfiles<CA> files.d<CA> nlsfiles<CA>d<CA>

Secondly, let's abritarily define "file is about NEWRETRIEVE" to mean that the file contains the content "NEWRETRIEVE" in at least one level 1 or level 2 statement. While this might be somewhat artificial, it demonstrates the use of viewspecs with a form.
First, I define a criterion named nwcrit. When nwcrit is evaluated at a specific statement, the link in the statement is taken and the branch at "the other end" under viewspecs "eb" is searched for a statement containing the content "NEWRETRIEVE". If one is found, the criterion nwcrit is TRUE. Otherwise, the criterion nwcrit is FALSE.
Secondly, I define a form named nwform. When nwform is evaluated at a specific statement, in this case (nlsfiles), an "extraction" is performed on the plex at (nlsfiles.d). The criterion used in this extraction is the criterion newret.

Recall that the plex at (files.d) is a plex of LINKS. When nwform is evaluated at the statement (nlsfiles), nwcrit will in turn be evaluated at each of the statements in the plex

at (nlsfiles.d). If for one of the statements nwcrit is TRUE , then that statement is handed back as part of the extraction. Otherwise, it is not handed back.

Finally, the structure handed back as a result of the extraction is inserted UNDER (expand). Recall that what each statement in this structure will be is a link to a file containing some level 1 or 2 statement with content "NEWRETRIEVE". Another example follows in which the ACTUAL STATEMENT with content "NEWRETRIEVE" is handed back.

```
(extract)
   Define Criterion nwcrit<CA>Some Branch .1<CA>eb<CA>Pattern [
   "NEWRETRIEVE" ] ;<CA>
   Define Form nwform<CA>Extract Plex .d<CA>w<CA>Name
   nwcrit<CA><^N><CA>
   Evaluate Form nwform<CA> nlsfiles<CA> expand<CA>d<CA>
(expand)
```

Expanding a plex of links

Assume that you have a plex of links to all your files, such as you would get if you did a Copy Directory. Do this to go under statement (files). Since you only want to expand links to NLS files, you first must "extract" the links to NLS files. I assume this has also been done already in the first example. If not just execute the branch at (nls).

Once the plex under (nlsfiles) as been constructed, the links are expanded to two levels by executing

```
Define Form reach<CA>Branch .1<CA>eb<CA><CA>
Define Form loopreach<CA>Loop Plex <^N><CA>Name
reach<CA><^N><CA>
Evaluate Form loopreach<CA> nlsfiles.d<CA> expand<CA>d<CA>
```

Alternatively, you may execute

```
Define Form onestep<CA> Loop Plex <^N><CA> Branch
.1<CA>eb<CA><^N><CA>
```

```
(structure)
   mary had
      a little lamb
      she was white
   harold was a barrel
      roll on
         over the falls
   yeah yeah
      we all have substructure
(handle)
(files)
   < CORNISH, AFMFORMAT.SUBSYS;39, >
   < CORNISH, AFMFORMAT.CML;11, >
   < CORNISH, CAPABILITIES.NLS;9, >
   < CORNISH, COBOLEIGHT.CML;4, >
   < CORNISH, COBOLEIGHT.SUBSYS;2, >
   < CORNISH, COBOLEIGHT.NLS;2, >
   < CORNISH, CODEOUTLINE.NLS;8, >  [ Being Modified By CORNISH (JAC3) ]
   < CORNISH, DECIMAL.CML;2, >
   < CORNISH, DECIMAL.SUBSYS;38, >
   < CORNISH, DMPAPERS.NLS;3, >
   < CORNISH, FORMALDEFINITION.NLS;10, >
   < CORNISH, JAC3.NLS;16, >
   < CORNISH, JAC3.NLS;15, >
```

```
    < CORNISH, MESSAGE.TXT;1, >
    < CORNISH, MOTIVATION.NLS;5, >  [ Being Modified By CORNISH (JAC3) ]
    < CORNISH, MOTIVATION.NLS;4, >
    < CORNISH, NEWRETRIEVE.NLS;78, >  [ Being Modified By CORNISH (JAC3)
    ]
    < CORNISH, NEWRETRIEVE.NLS;77, >
    < CORNISH, NEWRETRIEVE.CML;54, >
    < CORNISH, NEWRETRIEVE.SUBSYS;117, >
    < CORNISH, OBJECTLANGUAGE.NLS;3, >
    < CORNISH, PLINX.NLS;2, >  [ Being Modified By CORNISH (JAC3) ]
    < CORNISH, PLINX.NLS;1, >
    < CORNISH, PRIMITIVES.NLS;5, >  [ Being Modified By CORNISH (JAC3) ]
    < CORNISH, RULES.NLS;4, >  [ Being Modified By CORNISH (JAC3) ]
    < CORNISH, RULES.NLS;3, >
    < CORNISH, RULES.SUBSYS;5, >
    < CORNISH, RULES.CML;2, >
    < CORNISH, RULESDEFINITION.NLS;1, >  [ Being Modified By CORNISH
    (JAC3) ]
    < CORNISH, TEST.NLS;10, >  [ Being Modified By CORNISH (JAC3) ]
    < CORNISH, TEST.NLS;9, >
    < CORNISH, TESTCOBOL.NLS;1, >
    < CORNISH, <^V>]ARCHIVE-DIRECTORY[.;1, >
(nlsfiles)
(groups)
   Define Form front<CA> Group <^N>.h<CA><CA>
   Define Form back<CA> Group <^N>.t<CA><CA>
   Define Form switch<CA> Follow Name back<CA> Name front<CA>
   <CA><^N><CA>
(loops)
   Define Form roundabout<CA> Loop Branch <^N><CA> Statement
   <^N><CA><^N><CA>
```

1. Introduction .YBS=1;
This document discusses the design of a particular information retrieval system, HELP. The requirements of HELP are a subset of the general information-retrieval and data-base-management situation. This document is in a format which satisfies HELP's needs. It both presents a new formulation of HELP, to replace the existing one, and provides a concrete example of the issues to be dealt with in designing information retrieval systems.
The philosophy of this new HELP is to use a simplified subset of existing NLS. The features in this subset are explained in information retrieval terms to make them accessible to the HELP user without requiring him to know NLS. For example, a concept called "views" is introduced, which NLS users will recognize as simplified viewspecs. The HELP features are implemented using available NLS code, since in reality they already exist in NLS.
Note: the numbers here (e.g. 2a) are used to indicate which statements are "titles" and which "paragraphs" (explained below); the numbers would not appear in actual text.

2. Requirements for HELP .YBS=1;.PBS;

2a. Online/Offline
The same document is to be used for both online interrogation and offline printing.

2a1. Online
For online use, the document is divided into conceptual chunks which the user can retrieve and display. These chunks may be of any size. Most HELP documents will organize their chunks in some sort of hierarchy because of the hierarchical nature of commands. For example, if the user types "insert statement" and then hits the HELP button, the retrieval system will take him to the sub-chunk "statement" under the chunk "insert". Other documents will be more linearly structured.

2a2. Offline
For offline use, the document is to be capable of being printed meaningfully by an algorithm. This might be the Output Processor, or it might be some other subsystem. In any case, no human editing is to be required to get a readable hardcopy. The document should be linearly coherent and clearly organized, so that if users only look at the hardcopy, they will still have a useful reference document.
It is also reasonable to expect that the document will get "Output Quickprinted" during its lifetime, so it ought to be a pretty standard NLS file (or files).
Since most people are more familiar with hardcopy documents than with online ones, each document should probably be written first in offline form, as if it were to be printed. It can later be modified to make it more useful online, in the ways outlined in the next section.

2b. Skilled NLS users
Skilled NLS users are to be able to examine and move around in online documents in all the flexible NLS ways developed during the past ten years. The general principle is: the more you know, the more you can do.
In particular, the standard NLS sequence and portrayal generators are to be used, rather than special-purpose ones.

2c. Novice and non-NLS users
Users unfamiliar with NLS, and those from whom we want to hide

NLS, will have a simple (minded) set of commands, as is currently done with HELP and the NIC's QUERY.

2d. Opening documents

There are to be several ways to specify which documents to search. A good metaphor is a library of books: to look up something, a reader selects a book from the library and opens it to a certain page.

who does the 'catalog'

2d1. The HELP button

The HELP button provides a specific initial entry into a document. It "opens a book" at a designated place. For example, if the user partially enters a command and then hits the HELP button, HELP will show him the entry dealing with that command in the relevant documentation file.

2d2. Other entry

The user may explicitly specify a term to retrive and/or a document to search. If he does not specify a document, an algorithm computes a default document. For example, the current NSW HELP computes a tool HELP file from the tool name. Once a document is open, subsequent terms are retrieved from that document. This amounts to turning pages in a book once you have selected it from the library.

If a term is not found in a document, links may be included in the document naming other documents to search. This amounts to looking in a book's bibliography, selecting other books from the library, and searching them.

2e. Viewing documents

Since logical structure does not necessarily follow physical structure in a file, HELP is to provide logical ways to view a document. Examples are presented below.

2f. Multi-file documents

Retrieval requests across file boundaries are to be permitted. As mentioned above, documents may contain links to other documents, which may contain links to other documents, etc., forming a network. It is a network rather than a tree because document "A" may contain a link to document "B" which may contain a link back (eventually) to document "A". HELP will take precautions to avoid loops.

2g. Use existing HELP documents

Since so much effort has gone into the CORE, BASE and several smaller HELP files, they are to be used "as is", with at most some automatic conversion by an algorithm. However, these files are not in hardcopy form.

2h. Use existing NIC documents

Since a large number of NIC files exist which are of general interest to the user community, they are to be accessed by HELP. These files are already in good offline form, and require little conversion to online form.

The principle requirement is that links to NIC documents be placed in an index accessible to the retrieval system, such as LOCATOR.

2i. Use existing hardcopy documents

A large number of hardcopy-format documents exist. This number will grow as ARC people continue to produce documentation -- mostly in hardcopy form -- and a NSW tool builders begin furnishing their tools. These documents are to be accessed "as is", or perhaps with automatic conversion by an algorithm. Again, the principle requirement is that links to these documents

be placed in an accessible index, such as LOCATOR. All the documents already in LOCATOR are also to be accessed by HELP.

2j. Stepwise refinement of offline documents

When accessing a document designed for offline use, HELP cannot do better than NLS, since it knows nothing about the content of the document. However, HELP is to have the property that the more online-type information one adds to a document, the more flexibly it can be viewed and accessed. With this approach, limited retrieval on any hardcopy document can be achieved immediately; more advanced retrieval can be obtained in proportion to the number of features added.

If one designates certain statements as being "titles" (see below), then an outline view can be generated showing only the titles in the document, to any specified level. Also if titles are named statements, name searches can be done, which are much faster than content searches.

3. Document Organization .YBS=1;.PBS;

To satisfy the requirement that every document be usable both for hardcopy and online reference, documents must be linearly organized (i.e. readable from front to back in a coherent fashion) and also divided into manageable "chunks" which can be printed online. NLS gives us a good handle on this, since documents are already divided into statements and branches. Thus we have a start toward solving the problem of online access of existing hardcopy documents. The following model of a document further enhances the online capabilities.

A document consists of two types of statements: "titles" and "paragraphs". Titles are distinguished from paragraphs by some means, e.g. by making them named statements or by giving them a property. The title/paragraph distinction enables the system to display a document's LOGICAL structure (it titles), in addition to its physical NLS structure.

The assumption is that documents contain a set of concepts (the titles) which can be retrieved and a set of statements (the paragraphs) which explain those concepts. The two sets are not necessarily disjoint.

3a. Titles

Titles are ordinary NLS statements, usually no more than one line long. They form the conceptual organization of documents. Every title may have paragraphs associated with it, forming an NLS branch. A title branch completely explains one topic.

A title is the smallest unit HELP will retrieve.

3b. Paragraphs

Paragraphs are also ordinary NLS statements, possibly having sub-statements. They are of arbitrary length, and there may be arbitrarily many of them -- whatever is necessary to explain the title.

Paragraphs are introduced because the old HELP/QUERY menus are too restrictive. They disable NLS viewspecs and Jump commands. They limit to one (1) the number of ways documents can be viewed -- a statement followed by a numbered menu of one-line statements -- which has been the source of dissatisfaction. Removing the menu constraint permits a more flexible style of reading and writing.

4. Accessing Mechanism .YBS=1;.PBS;

4a. Views

HELP will provide three logical "views" of a document. These are

implemented using NLS viewspecs and/or content analyzer filters. Since these "views" use standard NLS portrayal features, they may be manipulated like any NLS structure. For example, a view of a document may be printed on a teletype, output to a line printer, copied to a file, bugged on a display, etc.

4a1. Outline view

Show just the titles, to a level specified. This view replaces the old HELP/QUERY menus. It presents the conceptual organization of a document, enabling the user to find out WHAT to ask about. It is implemented using a content analyzer filter and the viewspecs "ebtQ".

Link taking

If the user asks to see a specific statement in an outline view, and the statement contains a link, then the link will be TAKEN and the CONTENTS of the link will be shown. This automatic link-taking feature is an extra facility provided by HELP.

4a2. Full view

Show a complete topic: a title and all of the paragraphs under the title, i.e. an NLS branch. It is implemented using viewspecs "wdQ". This is the viewing mode into which the user is put if he types the HELP button while he is in the middle of a command.

4a3. Short view

This shows all the lines in the statement the user is currently at, plus one line of each statement one level down -- the current HELP view. It is implemented using a new NLS viewspec.

4b. Retrieving views

4b1. Retrieval by title

Show part of a document in an outline view. In other words, the titles themselves are the information retrieved. If the document does not have titles, then this retrieval cannot be done.

4b2. Retrieval by content, based on NAME lookup

A topic may be associatively retrieved by specifying its NLS NAME. This is the way HELP/QUERY currently work. It is also the way the Jump (to) Name command works. The reason for limiting content searches to name lookups is efficiency; if NLS ever implements other fast searches, then HELP can use them as well.

Therefore all titles should be named with the string which will be used to retrieve them. For example, the title dealing with "insert statement" should have the name "statement" and occur under a title named "insert". (We could use multi-word names if NLS implemented them.)

If the statements in a document are not named, as is the case with some hardcopy documents, then the search can still be done on the content of statements, rather than on their names. However, adding a feature to a document -- names and/or titles -- increases the accessing capabilities of the document.

4b3. Retrieval by address

A title can be retrieved by pointing to it or addressing it. HELP's convention is that if an outline view is being shown when a title is addressed or bugged, then that title will be shown in a full view. This is similar to displaying a document under viewspec "x" in NLS, then jumping to a statement with

viewspec "w" or "wq" on. In fact, that can also be done.

4b4. Next view

A "Next" button will show the next view of any kind -- i.e. a logical scroll. Thus a hardcopy document may be read "as is" by letting HELP open it, using the "full view", and stepping through it with the Next button. Or the titles in a document could be displayed and, if they don't all fit on the screen, the Next button used to see the rest. The same with class views.

This is similar to NLS's Jump (to) Next with viewspec "i" on.

4c. User interface

TYPEIN / BUG

User TYPEIN or BUGs will be looked up without requiring a command word.

If in outline mode when a BUG or menu number is given, HELP will show the designated topic in full view mode. If a word rather than a menu number is typed, HELP will look up the word without changing view modes.

The same applies to full view mode, with the exception that BUGs are interpreted as designating words rather than titles, and the word is looked up just as if it had been typed.

<SP> Outline (CONFIRM / TYPEIN)

Sets the viewing mode to "outline". If some TYPEIN is given, HELP goes to the word or address given. Otherwise, the user is left at the same place in the document.

<SP> Full (CONFIRM / TYPEIN)

Sets the viewing mode to "full". If some TYPEIN is given, HELP goes to the word or address given. Otherwise, the user is left at the same place in the document.

<SP> Short (CONFIRM / TYPEIN)

Sets the viewing mode to "short". If some TYPEIN is given, HELP goes to the word or address given. Otherwise, the user is left at the same place in the document.

<SP> Next <"view">

Prints the next view in the current viewing mode. This lets the user step through a document. If the full view is on, he can read an entire document using just this command.

<SP> Last <"view">

Prints the last view shown. May have to restore the viewing mode. The previous views are kept in a ring, as with NLS's Jump command.

<SP> Higher <"view">

Prints the "up" of the current topic in the current viewing mode. This lets the user see the context surrounding a topic, particularly if he is in outline mode.

OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFF
OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFF
OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFFICE-1 OFF

27 532

MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD
MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD
MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD

(DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (D
(DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (D
(DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (DSM)27532 (D

SATURDAY, NOVEMBER 13, 1976 10:30:58-PST SATURDAY, NOVEMBER 13, 1976 10:30:58-
SATURDAY, NOVEMBER 13, 1976 10:30:58-PST SATURDAY, NOVEMBER 13, 1976 10:30:58-
SATURDAY, NOVEMBER 13, 1976 10:30:58-PST SATURDAY, NOVEMBER 13, 1976 10:30:58-

< IJOURNAL, 27532.NLS;1, >, 6-FEB-76 13:41 XXX ;;;; Title: Author(s):
Robert Louis Belleville/RLB2; Distribution: /SRI-ARC([INFO-ONLY])
DPCS([INFO-ONLY]) ; Sub-Collections: SRI-ARC DPCS; Clerk: RLB2;
Origin: < BELLEVILLE, VIDEO-GENERATION.NLS;3, >, 5-JAN-76 14:26 RLB2
;;;;####;

VIDEO GENERATION TECHNIQUES FOR THE NLS WORKSTATION

This document describe the techniques used to generate tv pictures from a digital computer. Anyone will be able to read and appreciate the information discussed; however, this is a very technical paper and may not be of interest to many. Also it is about 17 pages and there are 6 pages of drawings which have not been journalized, but can be obtained from the author.

VIDEO GENERATION TECHNIQUES FOR THE NLS WORKSTATION 1

INTRODUCTION 1a

Video has captured the imagination of most computer display designers. Why is this? 1a1

First, video displays are low cost. Complex parts such as the CRT itself, which would be prohibitively expensive to produce in small quantities, are cheap and plentiful because of the impact of broadcast tv. Moreover, the technology is well worked out, standards have be adopted, and many manufactures compete in an open market. While the home tv has relatively low picture quality, broadcast studios and the closed circuit tv market has produced a demand for very high quality tv equipment at reasonable cost. 1a2

Second, the video waveform is attractive for several reasons: 1a3

1) The image is coded in a standardized way. 1a3a

2) This code provides for the creation of an image from a single stream of data on a single wire. No complex interface is required between the source of the video signal and the tv monitor. Part of the intelligence needed to describe the picture is contained in the monitor itself. The video production system need not concern itself with the mechanism need to trace out the picture, it need only provide the coded information. 1a3b

All the information to construct the picture is contained in the low energy video channel. Random or calligraphic displays distribute the information into 2 high energy position channels and a low energy intensity channel. Video on the other hand uses two positioning channels which operate at fixed frequencies so that the driving circuitry can be optimized to minimumize the complexity and power requirements. 1a3b1

3) The code provides for the creation of not only line drawings but also pictures containing grey levels or color. In its simplest form, computer generated video need only contain a binary level - on or off; but if necessary, the computer video generator can produce grey levels and color. 1a3c

4) Although there is no logical information in the video signal, the waveform is rich in spatial information. This information can be used to mix video waveform from several

sources into a single video output to provide split screen and overlay characteristics. 1a3d

5) The bandwidth of the source and the monitor will control the density of information displayed to the user. The information will be displayed without flicker or drift. 1a3e

Third, video technology is based on memory. No matter which of the several video production techniques are used, the designer must find relatively large quantities of memory. Video is thriving today because memory cost is dropping dramatically, memory speed is increasing, memory density is increasing, and the power consumption is dropping. In short, the vital element needed for video production is becoming the simplest and cheapest part of the system. 1a4

Fourth, related technologes are advancing. 1a5

1) The creation of the data structures needed to convert pictures described as computer based data structures (both lines and surfaces) is relatively well understood. The process of scan conversion requires considerable processor speed, but here again, the speed of processors is increasing while the price drops. 1a5a

2) Electrostatic printer/plotters are both fast and reliable. And the price is quite reasonable. Mechanical plotters are obsolete for the production of working drawings and documentation. (Large mechanical plotters are still required for the production of circuit masks and so on; however, these machines should be considered as numerically controlled tools - not computer peripherials.) The technology needed to drive the electrostatic printer is identical to that needed for the production of video. 1a5b

3) The use and storage of pictures, that is photographs, is increasing and the technology for the encoding of these images relates closely to the problems of video production. Moreover, the analog storage of pictorial images on video tape and microfilm is possible and its use will grow. The video monitor included in the workstation can be used directly to view these images because the input waveform is the same for both sources. Optical Character Recognition (OCR) is just beginning to capture and create, markets and technology for the input of all sorts of printed matter, pictures and drawings. As you may have guessed, the scanners used to capture the data for OCR produce data structures which, in their raw form, are completely compatible with video production. 1a5c

As you can see, computer generated video has many advantages. In fact no, technology available today provides the flexibility and effectiveness of video. In the sections below, we shall look in detail into the problem and processes of video. 1a6

CONSIDERATIONS FOR VIDEO 1b

VIDEO STANDARDS 1b1

In this section we shall look at the time critical nature of video production. Three EIA standards apply to the video waveform: 1b1a

1) RS-170, Electrical Performance Standards for Monochrome Television Studio Facilities. (figure i) 1b1a1

2) RS-330, Electrical Performance Standards for Closed Circuit Television Camera 525/60 Interlaced 2:1. (figure ii) 1b1a2

3) RS-343, Electrical Performance Standards for High Resolution Monochrome Closed Circuit Television Camera. (figure iii) 1b1a3

These standards set the refresh rate to 60 per second. Once set in motion, the video waveform must be maintained with relentless precision. 1b1b

ORGANIZATION OF THE VIDEO WAVEFORM 1b2

In order to create the video waveform, the picture is converted into a series of horizontal strips, starting at the top and continuing to the bottom of the screen. These strips, called horizontal scan lines, are made by measuring the whiteness of the picture at each point from left to right. Since the strip is determined by scanning sequentially from left to right, the horizontal displacement (x dimension) is represented by the time from the beginning of the scan line. In the same way, the vertical dimension is determined by the time from the beginning of the first scan line. 1b2a

The complete picture could be represented by the horizontal scan lines taken one right after the other; however, it is not. The picture is scanned in two passes called fields. The even numbered scan lines (0, 2, 4, 6, ...) are placed in the first field; and the odd numbered scan lines (1,3,5,7, ...) are placed in the second. Together the first and

second field are called a frame. The fields are said to be "interlaced 2:1." 1b2b

Since each field must be completed in 1/60 th of a second, and two fields comprise a frame, the complete picture is repainted 30 times a second. This is comfortably above the threshold for persistence of vision so that the viewer does not perceive any flicker. 1b2c

Each horizontal scan line is separated from the next by a pulse called the horizontal sync. The diagram below shows the horizontal sync pulses separating the video (intensity) information. As the diagram shows, the video signal starts just above 0 volts with black, continues with successively lighter shades of grey, and ends with white at the highest level. Near the horizontal sync pulse, the signal level drops below black (called blacker than black) to a level which insures that no image is produced. The process of suppressing the video signal is called blanking. The actual horizontal pulse is located slightly off center of the horizontal blanking pulse and is negative going. 1b2d

The horizontal sync pulse resynchronizes the horizontal oscillator in the tv monitor so that each of the scan lines starts at the same point on the monitor screen. In addition to resynchronizing the horizontal oscillator, the horizontal blanking period allows the beam to retrace back to the left side of the screen before it launches off for the next scan line of the field. 1b2e

The horizontal oscillator is used to deflect the CRT beam horizontally from left to right in a consistent linear way then snap the beam back to the left during the horizontal blanking period. The vertical oscillator performs the same function for the vertical axis but at a much slower rate. The video waveform provides a sync pulse similar to the horizontal one to resynchronize the vertical oscillator of the tv monitor. Retracing the beam from the bottom right of the screen to the top left takes about 1.25 milliseconds. Horizontal scan lines which would ordinarily fall into this time period are lost. 1b2f

Interlace is accomplished by delaying the start of the second field. This phase change is accomplished by continuing the horizontal sync pulses during the vertical sync period. These pulses are called serrations. Near the middle of the vertical period the horizontal sync pulse train is shifted by one half of a horizontal time period. The vertical oscillator has remained constant, but the video

information is delayed by one half period. The vertical oscillator causes the beam to move down the face of the CRT by two scan lines for each horizontal period. As a result, the beam of the tv monitor will be lower on the crt by exactly one scan line. For this field the scan lines will be between those of the preceding field. On completion of the second field the delay is removed and the process is repeated. 1b2g

PERFORMANCE REQUIREMENTS 1b3

Most video waveforms are produced by image tubes in tv cameras. The vidicons and image orthocons used in these devices are essentially analog devices. The precision of the deflection circuitry of the camera and the bandwidth of the video amplifiers determines the maximum resolution and image quality of the video system. For digitally produced video, the speed of the digital hardware determines the resolution of the video produced. 1b3a

Broadcast television uses 525 lines with a 2:1 interlace. Closed circuit systems may use higher scan rates. In the table below the number of scan lines actually available for image use and the time available for the production of video is given for several common systems. (us = microseconds) 1b3b

Line Rate	Time Per Horizontal Scan Line	Video Time Per Scan Line	Lines Available
525	63.5 us	53.0 us	487
675	49.4 us	42.8 us	625
729	45.7 us	39.1 us	675
875	38.1 us	31.5 us	811
945	35.3 us	28.7 us	875
1023	32.6 us	26.0 us	947
1200	27.8 us	21.2 us	1112

1b3b1

For example, if an 875 line system is to be used for the computer production of video, then a scan line must be produced every 38.1 microseconds. The time available for video on the scan lines will be the horizontal time minus the horizontal blanking time in this case 31.5 microseconds. Because some scan lines are lost during the vertical retrace only 811 scan lines may be used for the production of an image. 1b3c

The amount of data transmitted on each horizontal scan line is not completely independent of the number of lines in the

system. The aspect ratio (height/width) for television is 3:4, that is the screen is wider than it is deep. In order for the image to be linear (undistorted) the number of points on a scan line must be 4/3 of the number of horizontal scan lines. The table below contains the bit width and the time per bit for the systems listed above. The Approximate Bandwidth indicates the minimum bandwidth of the tv monitor needed to reproduce a picture with every other point on. 1b3a

Line Rate	Number of Points	Time per Bit	Approximate Bandwidth
525	700	75.7 ns	6.6 MHz
675	900	47.5 ns	10.5 MHz
729	975	40.3 ns	12.4 MHz
875	1167	27.0 ns	18.5 MHz
945	1260	22.8 ns	22.0 MHz
1023	1364	19.0 ns	26.2 MHz
1200	1600	13.2 ns	37.8 MHz

1b3d1

In the 875 line example, there would be 1167 points on each scan line. The production system must place a bit on the scan line every 27 nanoseconds (ten to the minus 9th) and the tv monitor must have a bandwidth in excess of 18.5 MHz. (million cycles per second) 1b3e

SUMMARY 1b4

The production of video by digital hardware is constrained by the figures given in this section. Conceptually the production of video is very straightforward. The image must be coded in some way and stored into a memory which will be used to refresh the monitor 60 times a second - the refresh memory. In addition, the data must be extracted from the refresh memory and converted into the video signal at a rate high enough to meet the the requirements set out in this section. Finally the unit must produce synchronizing signals (or accept them) to control the horizontal oscillator and the vertical oscillator of the monitor and to keep the video generation hardware in step with these signals. In the sections below the various production techniques are discussed. 1b4a

FONT POSTING 1c

INTRODUCTION 1c1

Font posting is the most common form of digital video

production. In this technique, the characters to be displayed are stored in the refresh memory. To refresh the monitor each character is taken from the refresh memory and the value is used to address another memory which contains the bit pattern for each row of the character. This "font memory", and the system which looks up the font definition from the font memory and places the bits into the video output, is called a character generator. 1c1a

FONTS 1c2

The figure below shows the essential features of the fonts stored in this type of system. Horizontal bit positions are called rows and represent slices through the character. Each element of the row is called a bit and corresponds to the value of the video signal (on or off) for this particular position in the character matrix. 1c2a

Two items deserve special mention: base line, and descender. The base line is an arbitrary line designated by the font designer and represents the line upon which the capitol letters sit. Descenders are the parts of the character which lie below the base line. In system with both upper and lower case characters, descenders are needed to enhance the readability of the image. 1c2b

Character sets are defined on a matrix which is a table of bits designated as width by height, for example, 5 X 7 is very common. Fonts are posted on the display in an envelope around the character which allows for the inter-character and inter-line spacing. For many systems the 5 X 7 fonts are posted in 7 X 10 envelope. In general the bigger the character definition, the more readable are the characters. 1c2c

CHARACTER GENERATOR AND FONT MEMORY 1c3

The organization of the character generator is shown below. Basically the device is just a memory whose width is determined by the font definition and whose size is the number of rows per character times the number of characters supported. The font memory for a 5 X 7 character set containing 96 characters will be 7 X 96 = 672 X 5 (bits wide) = 3350 bits. 1c3a

Font memory may be constructed of read only memory (ROM), programmable read only memory (PROM), or random access memory (RAM). 1c3b

ROM's are cheap, fast, large and unalterable font

memories. ROMs of 16K bits are available today at relatively low cost. Character ROM chips such as the Motorola MCM6571 contains the definition of 128 characters (Ascii plus lower case Greek symbols) on a 7 X 9 dot matrix with provision for the production of shifted descenders (pseudo 7 X 16 definitions). This chip contains 8192 bits, has a cycle time of less than 500 nanoseconds, and costs less than $12 in single quantity. 1c3b1

Most ROMs like these are mask programmable, that is the user can provide the manufacturer with the bit pattern required. There is a one time charge for the mask layout (about $600) and a requirement that the customer purchase a relatively large quantity (100 units). 1c3b1a

Currently there are two kinds of PROM memories - reprogrammable and field programmable. Field programmable PROMs are permanently programmed by blowing a microfuse associated with each bit of the memory. Once programmed, these devices cannot be changed. Reprogrammable devices can be programmed in much the same way as the field programmable units; however, a exposure to a strong dose of ultraviolet light will reset the device to the original state. Field programmable units are available to about 8K bits with speeds below 60 nanoseconds. New reprogrammable units contain 8K bits, have a cycle time of about 500 nanoseconds, and cost about $100. 1c3b2

Semiconductor RAM memory is currently the focus of a great deal of excitement. The greatest attention has be given to MOS (metal oxide semiconductor) especially N-channel. Though the technology varies there are two general classes - static and dynamic. 1c3b3

Static RAM can be written and read at any rate up to the maximum cycle time of the device while dynamic RAM must be "refreshed" at some minimum rate to insure that the information (which is stored as a charge in a tiny capacitor) is not lost. Static RAM is very easy to use but is slightly more expensive in large systems. Dynamic RAM, because of its simpler internal structure, is available in relatively high densities (bits per chip). A 4K dynamic RAM with a 450 nanosecond cycle time is less than $20 today and 16K dynamic RAMs will available in 1976. The current cost is about 0.5 cent per bit or less for small memories

and memories of 250,000 bits cost about 1 cent per bit including necessary interface logic. 1c3b3a

Font memories, built from RAM, have no initial state so a display constructed with writable font store must have the ability to initialize the font memory from its own permanent memory or form its host computer. 1c3c

The cost of a RAM based font memory can be estimated. Suppose a system is to contain a writable font store of 128 characters constructed to include descenders on a 7 X 16 matrix. If the font is implemented with Intel 2102A static RAMs (1024 by 1 bit) then 127 X 16 = 2048 rows of 7 bits are required. Since the memory is 7 bits wide then 7 X 2 = 14 (1024 bits per chip) chips are required. The 2102A costs $470 per 100 which yields $65.80 for a computer writable 450 nanosecond character generator. 1c3d

(The cost of the 2102A dropped 100% in 1975 and the new 16K RAM is expected to cost $10 by 1977 or 1978.) 1c3d1

REFRESH MEMORY 1c4

The CRT screen is organized as shown below as a matrix of m columns by n lines of characters. In order to strip the bits from the font memory for the each line, each character must be given to the character generator in order and the sequence repeated until the number of rows per character is exhausted. 1c4a

The refresh memory contains several kinds of information. 1c4b

1) The character code value. For reasonable systems this will be the 7 bit Ascii character code with the control characters mapped into special characters. (These symbols are called "Pi" characters.) 1c4b1

2) A number of bits from none to several to select the font if more than on font memory is available. For example a single bit might be used to point either a fixed ROM character generator and a single bank of writable font memory. 1c4b2

3) A few bits controlling special marking functions such as underlining or reverse video. 1c4b3

For example, a refresh memory with reverse video, underlining, two fonts, and a 7 bit character code will need to be 10 bits wide. 1c4c

There are at least two approaches for the organization of the refresh memory. First, the traditional scheme is called M by N and uses two recirculating shift registers to hold the character data. The M register is the larger and slower of the two. To create the video, a line of characters is placed into the N memory which is smaller and faster. The N memory is cycled enough times to paint all the rows required for the font. The M by N scheme is relatively cheap and flexible and is found in most commercial units. The major drawback is that a specific character cannot be written randomly into the M memory, but must wait until the character position passes through the one character window provided by this sort of memory arrangement. 1c4d

Second, a true random access memory can be used to represent each matrix location of the screen. In this form the cycle time of the memory must be as fast as the character time required for output; however, any character can be written at random and the refresh memory can be manipulated at very high speed by the terminal computer. 1c4e

At present, random access memory is not particularly more expensive than shift registers, and the flexibility of the random approach will be explored here. A good description of the M by N system can be found in National Semiconductor Application Note AN-40, "The Systems Approach to Character Generators", June 1970. 1c4f

PERFORMANCE REQUIREMENTS 1c5

In font posting the number of characters per line, the number of columns, is the controlling parameter which controls the speed of the character generator and the refresh memory. The table below shows the number of nanoseconds available for each character of a 80 character line for the systems under consideration: 1c5a

Line Rate	Nanoseconds per Character
525	662 ns
675	535 ns
729	489 ns
875	394 ns
945	358 ns
1023	325 ns
1200	265 ns

1c5a1

Both refresh and font memory speed are determined by the figures given in the table above. For example, in the 875 line system the font memory must cycle in less than 394 nanoseconds. The refresh memory must make a character value available to the character generator in the same time period. (The presentation of the character value may be overlapped with the access of a row of bits from the font memory by delaying the transmission of a character by one character time.) 1c5b

Suppose we wish to build a 64 line display with 80 characters per row. If the envelope is selected as 10 X 13 (the usual size for a 7 X 9 character), then 64 X 13 = 832 scan line are required. From the tables above we find that a 945 line system will be needed. Horizontally, 10 X 80 = 800 points are required and the 945 line system provides 1260. The font memory must be able to produce a row every 228 nanoseconds (a bit slower if the characters are allowed to be a bit wider than they are tall). Such a system could be built today by using ROMs for the font memory and high speed RAM (MOS RAMs below 200 nanoseconds are available for about 2 or 3 times the price of slower units discussed above) for the refresh memory. By portraying 80 X 64 = 5120 characters, this system would represent about the limit of the current state of the art. The cost of the memory and related hardware would be about $2000 for the heart of the unit. 1c5c

SYSTEM BLOCK DIAGRAM 1c6

The figure below shows the design of a font posting video generator. The system is designed with a single static font. The refresh memory is constructed from RAM and the basic elements of the computer interface are shown. 1c6a

BIT MAPS 1d

INTRODUCTION 1d1

Although font posting is the most common technique, bit mapping is the simplest. The image to be portrayed is represented as a large matrix of bits which are taken from the refresh memory and sent to the tv monitor. 1d1a

The simplest form of the bit map uses only one bit per picture element (pixel). The number of points per screen is a measure of the image quality. For reference, the table below shows the number of bits required to store bit maps of several sizes. 1d1b

Size Width X Height	Number of Bits
128 X 128	16,384
256 X 256	65,536
512 X 512	262,144
1023 X 1024	1,048,576

1d1b1

The number of bits rises as the product of height times width. Until recently, the cost of memory made the use of bit maps too costly. Now, the cost of a million bits of memory is less than $10,000 and a 512, by 512 memory matrix is well less than $2,000. 1d1c

By using more than a single bit for each screen point, color and grey scale can be produced. 1d1d

PERFORMANCE REQUIREMENTS 1d2

For the systems detailed above, the time per bit is less than the cycle time of all but the fastest memories; however, this is not a problem. The refresh memory (that is the bit map) is organized to deliver several bits in parallel to a high speed shift register to produce the serial stream of bits. 1d2a

For example in the 525 line system 487 lines are available and a bit map of 512 by 487 bits can be portrayed. Each bit must be delivered to the video amplifier in 75.7 nanoseconds; however, if the refresh memory is organized as 16K words of 16 bits, then 16 bits are obtained in each memory cycle. To find the minimum cycle time of the refresh memory we multiply 75.7 X 16 = 1211.2 nanoseconds are available. Most semiconductor memory has a cycle time less than 1000 nanoseconds. 1d2b

SOFTWARE REQUIREMENTS FOR BIT MAPS 1d3

The software to control the bit map cannot be overlooked without the risk of a system design of relatively poor performance. There are two basic problem areas: 1d3a

1) Font posting must be performed by software. 1d3a1

2) Since the bit map provides the ability to portray line drawings, the conversion of the lines into a bitwise representations must be considered. 1d3a2

The access of the terminal computer to the bit map will

determine the speed of the system. Systems have been implemented in which the bit map is on the same bus structure as the terminal computer. This implementation has two drawbacks. First, if the address space of the terminal computer is limited, then the bit map takes away from the the area available to run program material. Second, the video generation equipment has high priority access to the memory. As a result, a large percentage of memory cycles are lost to the terminal computer. 1d3b

If the bit map is not implemented as part of the terminal computer then the unit must be addressed a an I/O device. This means that loading or reading will take place no faster than the maximum bus rate. DMA (Direct Memory Access) can be used to update the display quickly; however, the bit map will be, in general, as large as or larger than the memory of the terminal computer so the size of the buffer available will be too small to be of much use. Under program control the speed of I/O will be determined by the minimum meaningful program loop which can place data on the output port. For example, an LSI-11 could do some kinds of output using: 1d3c

```
        MOV #buffer,R0          ;start of data to be sent
        MOV n,R1                ;number of words to go out
LOOP:   MOV (R0)+,@#device      ;output to bit map 16 bits
        SOB R1,LOOP             ;continue for n words
```
1d3c1

This loop takes about 12 microseconds for each word. If a 512 X 512 bit map was to written by this technique 16,384 X 12 = 196,608 microseconds or about .2 seconds are required. Of course, the only use of a loop like this would be to clear the screen (CLR @device instead of the MOV). By comparison the Tektronix storage tube requires 0.5 seconds to clear the screen. 1d3d

If a buffer contained the data for a line of text to be written on the bit map the time would be 32 (words wide) X 10 (rows per character) X 12 (microseconds) = 3.84 milliseconds. Creating the buffer from incoming Ascii characters would take considerably more time; however, at 9600 baud about 72 milliseconds would be available to convert a text line containing 72 characters. 1d3e

Moving a line of text from one line to another would require a read of 320 words from the bit map followed by a write of 320 words at the new location. This will take about 7.68 milliseconds. This rate seems slow; however, it is consistent with the speed of existing alphanumeric displays.

(moving chunks of the bit map around is a discussed in the section of refresh memory management.) 1d3f

Converting lines from an endpoint representation to a stream of bits is called scan conversion. The algorithm to fit a line from points in a bit map space was developed by Bresenham in 1965 to control a mechanical plotter. (J. E. Bresenham, "Algorithm for Computer Control of a Digital Plotter", IBM System Journal, Vol. 4, No 1, 1965.) The algorithm is called a digital differential analyzer (DDA) because it solves a difference equation which results in a best fit. Although the algorithm requires only addition, subtraction, and a sign test to operate, writing a random line into the bit map would require nearly 100 microseconds per point. A line across the screen would require 512 X 100 = 51.2 milliseconds to complete. A single line could be transmitted to the terminal in 8 Ascii characters (8 milliseconds at 9600 baud). 1d3g

Unlike a storage tube, single lines can be deleted. However, the so called point in common problem limits the use of this feature to some extent. If two lines cross they have one or more points in common. If one line is erased, the common points cannot be detected so they are erased as well leaving a hole in the remaining line. 1d3h

Although bit map software is more complex than font posting, the rewards are greater. Character size and style can be easily controlled. Linework, while slow, is possible; and the direct use of linework graphics greatly enhances the flexibility of the terminal display. 1d3i

SYSTEM BLOCK DIAGRAM 1d4

The figure below shows the design of a bit map generator for 525 line applications. The simplicity of the organization is striking. 1d4a

OTHER TECHNIQUES 1e

INTRODUCTION 1e1

There are several other techniques for the digital production of video. Font posting and bit maps represent two ends of a spectrum which contains various methods of image encoding. Font posting can use compact image coding because the class of pictures (that is printed pages) is limited. Bit maps use the least compact form of image encoding because the range of images is not limited. 1e1a

From the point of view of information theory, the maximum amount of information in a two dimensional picture is obtained when the value of each point is chosen at random. If such a picture is displayed, the result is a grey patch. Pictures which are meaningful to a human observer contain some order, and thus less information than the random picture. As a result the number of bits needed to represent the picture is less than the bit map. In general, photographs contain the most information, line drawings much less, and finally text contains the least pictorial information. The techniques described below employ various coding techniques to store and display the image. 1e1b

RUN CODING 1e2

Run coding is a popular technique which works because the video data on a scan line usually does not vary at every point. Usually there are large areas of a single video level. For example, the background is the same shade of grey and occupies much of the surface area of the image. Instead of coding each point, as in the bit map technique, runs of the same grey level are encoded as a value and the number of times to apply the value. Memory bandwidth can be a problem for those parts of the picture in which the value of the video level does change rapidly, as is the case for closely spaced vertical lines. Encoding information for a run code generator is more complex than for a bit map; however, the simplicity of the generator circuit and the compression of the data may favor the use of run coding. 1e2a

A run code processor is described by Ben Laws in "A gray-scale Graphic Processor Using Run-length Encoding", Proceedings of the conference on Computer Graphics, Pattern Recognition, and Data Structure, May 14-16, 1975. (XDOC #32489) 1e2b

FOURIER TRANSFORMATION 1e3

Partly as a result of the work on broadcast television bandwidth compression, and partly as a result of the work on automatic image recognition; signal theory has been expanded to encompass two dimensional "signals". All the work on filter theory, fourier domain, and so on has be generalized to apply to pictures. This work will be expected to have an impact on digital video generation in the next decade; however, at the present the techniques are directed toward image compression and recognition. 1e3a

REAL TIME GENERATION OF LINEWORK 1e4

Linework is best stored as endpoint coordinates. Not only does this representation provide the most compact form of storage, endpoints provide the most logical way of manipulating the linework. Realtime linework generators seek to convert a data structure consisting of endpoints into the video waveform. 1e4a

The algorithm used to write into the bit map is implemented in very fast hardware to operate on "y sorted" vectors. As each new scan line is started, the hardware must find all the lines which begin and end at this y position. Exiting lines are removed from the linework generator. Lines are processed to determine the intersections with the current scan line. One or more buffers are computed ahead of the actual video output to insure that the output bit rate requirements are met. 1e4b

As the discussion of bit maps and video considerations shows, the speed of the hardware needed to set the horizontal bits on the scan line must be very great in order to allow for a reasonable number of lines which cross a scan line. While the speed is great, the technique is possible and we can expect some commercial units to become available in 1976 or 1977. 1e4c

HYBRIDIZATION 1e5

One of the advantages of the video system is that any of the techniques above can be mixed together at the final video amplifier. For example, if a bit map is used, then rubberbanding lines will be a problem. A real time generator for a single line is simple to construct, so a hybrid system with a single rubberband line and a bit map can be built to remove the limitation. All the techniques have advantages, and the choice of technology for a given application will dicate the best choice; however, specific limitations can be removed by added additional hybrid capacity. 1e5a

REFRESH MEMORY MANAGEMENT 1f

INTRODUCTION 1f1

The refresh memory for either the bit map or font posting can be implemented as a matrix whose size is dictatd by both the physical implementation and the required size, or an additional memory can be used to map the lines and rows from arbitrary portions of the total address space. 1f1a

MEMORY CONTROL FOR FONT POSTING 1f2

The direct use of a random access memory for font posting can require some waste of available memory. For example, suppose a 80 X 32 refresh memory is to be constructed from 1K chips. The total number of characters required for this implementation is 80 X 32 = 2560 which is 2560 - 2048 = 512 character positions in excess of 2k. This means that 3K of memory will be needed and 512 character positions will be lost to the system. 1f2a

Placing another memory between the line address counter and the refresh memory, allows the terminal computer to map the lines from the refresh memory into the order required for display. This mapping allows for fast scrolling and it allows the full memory of the display to be utilized even if all the lines in the refresh memory cannot be displayed. 1f2b

The number of characters displayed on each line can be controlled by three techniques. 1f2c

1) The lines can be of fixed length. 1f2c1

2) The refresh memory can contain an end of line character which is used to discontinue font posting when it is encountered. 1f2c2

3) The memory mapping register can contain the line length. 1f2c3

If the latter two techniques are used, then in general the refresh ememory can contain more lines of text than in the fixed length scheme. 1f2d

MEMORY CONTROL FOR BIT MAPPING 1f3

Bit maps for screen sizes other than even multiples of two will benefit from the use of a memory map for the display of each row of bits. For example in the case of the 525-line monitor only 487 lines are available horizontally; however, about 700 points are available on each line. If a 256K-bit memory serves as a refresh memory in an unmapped system then 9 bits will be used to address each axis (512 X 512). (If say 7 bits were used to address the horizontal rows then the bit map would be 1024 X 256, and 487 - 256 = 231 scan lines would be lost.) Since only 487 scan line can be seen, 25 are lost. If a memory map is used the width can be increased (by 25 points) and the full memory used. 1f3a

In bit maps, the slowest operation is the bulk transfer of groups of rows. In the discussion above, the time to move two lines of text would be about 8 milliseconds. If memory mapping is available the time would be reduced to a few microsceonds. As a result, scrolling text is simplified. 1f3b

SYNC GENERATION 1g

SYNC GENERATION 1g1

Sync generation for 525 line systems has been greatly simplified by the introduction of several sync generator chips. These units produce the full EIA sync waveform and provide separate signals for vertical and horizontal synchonization. The chips cost less than $20 and replace commercial units costing several hundred dollars. 1g1a

For high resolution systems we can expect similar developments to be available in the future. 1g1b

2

USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC
USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC
USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC

MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD
MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD
MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD

(DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM
(DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM
(DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM

SATURDAY, NOVEMBER 13, 1976 10:41:02-PST SATURDAY, NOVEMBER 13, 1976 10:41:02-P
SATURDAY, NOVEMBER 13, 1976 10:41:02-PST SATURDAY, NOVEMBER 13, 1976 10:41:02-P
SATURDAY, NOVEMBER 13, 1976 10:41:02-PST SATURDAY, NOVEMBER 13, 1976 10:41:02-P

USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-
USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-
USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-ISIC USC-

MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD
MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD
MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD MAYNARD

(DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DS
(DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DS
(DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DSM)NSF (DS

SATURDAY, NOVEMBER 13, 1976 10:41:02-PST SATURDAY, NOVEMBER 13, 1976 10:41:02-
SATURDAY, NOVEMBER 13, 1976 10:41:02-PST SATURDAY, NOVEMBER 13, 1976 10:41:02-
SATURDAY, NOVEMBER 13, 1976 10:41:02-PST SATURDAY, NOVEMBER 13, 1976 10:41:02-

< MAYNARD, NSF.NLS;1, >, 12-NOV-76 04:22 DSM ;;;;
(todo)
Get previous versions of NSW proposal from NINA. Extract separable chunks.
Boiler plate intro.
Select Set of optimal, separable?, projects:
Video Terminal
Debugger, DAD.
FE Task areas?
Single user system
Study on ???
Implementation work other O.S. other machines.
Transportability Study
Distributed Journal system
Help system
Library support system
Graphics R&D see Bellville
Forms subsystem
Locate likely project areas within NSF and identify key people.
Investigate previous contacts with key people.
Draft think pieces.
Contact (phone or letter?) key people with the aim of making them receptive to at least reading our think piece.
Send think pieces
Follow upd
Decide on proper proposal format.
Sumbit proposals
(done)
Printed (28442,) Suggested Projects for RADC by HGL and KEV
Requested retrievel of (36992,) VIDEO TERMINAL Preliminary Specifications from archieve.
(documents)
(HJOURNAL,28777,) RADC Proposal by HGL and KEV
O-1: Online ISIC: online
DAD Do All Debugger
Introduction (HJOURNAL,28777,3a)
Demonstration (HJOURNAL,28777,3b)
Study (HJOURNAL,28777,3c)
Software Engineering
(28442,) Suggested Projects for RADC by HGL and KEV
O-1: Online ISIC: online
DAD Do All Debugger
Introduction (hjournal,28442,3a)
Study (hjournal,28442,3c)
Graphics maintenance
Software Engineering
(27532,) VIDEO GENERATION TECHNIQUES FOR THE NLS WORKSTATION
O-1: Online ISIC: not there
(arcdocumentation,nsw-proposal,)
VIDEO TERMINALS
(arcdocumentation,nsw-proposal,1d4)
COMMAND SEQUENCE FACILITY
(arcdocumentation,nsw-proposal,1e1)
DEBUGGER SUPPORT
(arcdocumentation,nsw-proposal,1e3)
QUALIFICATIONS

(arcdocumentation,nsw-proposal,1f)
(HJOURNAL,35953,) EXTENSIONS TO QUERY2 SUBSYSTEM
QUALIFICATIONS
(HJOURNAL,35953,6)
REFERENCES
(HJOURNAL,35953,9)
(bjournal,36932,) KNOWLEDGE WORKSHOP UTILITY SERVICE IN SWEDEN
QUALIFICATIONS
(bjournal,36932,2) Good for user issues, community approach, technology transfer
II BACKGROUND .CEN=1;.GCR;
A. General
B. ARC's Community Plan
C. Objective
D. Applications of the System and Technology Transfer Aspects
REFERENCES
(bjournal,36932,4) Good Overview
IV SELECTED REFERENCES .CEN=1;.GCR; .Pbs; .IOvr=2;
1 ARC 34111, N. D. Meyer, "Executive Information Tools," Augmentation Research Center, Stanford Research Institute, December 1975.
2 ARC 34137, J. C. Norton, "Workshop Utility Service III," Augmentation Research Center, Stanford Research Institute, 31 December 1975.
3 ARC 35672, J. H. Bair, "Seminar on the Augmented Knowledge Workshop," Augmentation Research Center, Stanford Research Institute, 11 June 1976.
4 ARC 36931, D. C. Engelbart, J. C. Norton, J. H. Bair, "AKW System Capabilities and Features," Augmentation Research Center, Stanford Research Institute, September 1976.
5 ARC 36944, J. C. Norton, "Status of Workshop Utility Service Subscriptions," Augmentation Research Center, Stanford Research Institute, 30 September 1976.
(HJOURNAL,36044,) NETWORK OPERATING SYSTEM STUDY
Change RADC to NSF and resubmit

(commands)
Go Programs <CA>
Load Program xprograms,include.sg<CA>
Load Program maynard,feedit.proc-rep<CA>
Load Program maynard,addressfilter.ca<CA>
Quit <CA>

(files) %used by printing system%
(maynard,addressfilter,) %source code for address generator%
(maynard,nsf-addresses,) %data base for NSF addresses%
(maynard,references,) %collected references %
(maynard,ref-index,) %includes to references by subject area%

(Cover) % A sample cover letter%

The Augmentation Reseach Center of Stanford Research Institute is interested in exploring the possibility of obtaining National Science Foundation funding in order to continue some of our ongoing research efforts. This document consists of three major sections; An introduction to SRI and the Augmentation Research Center, a high level description of the proposed research effort, and a bibliography of pertinent references. If you have any questions about SRI, ARC,

or the proposed research effort, or you would like additional information please contact me. If you think your program within NSF might be interested in funding some level of effort in this area I would be glad to submit a formal proposal. We would welcome any comments you could give us to aid the production of a mutually satisfying proposal. If your particular NSF program is not interested in funding any effort in this area, but you know of another program within NSF that might be interested, please feel free to forward this document to them. Thank you very much for your time.

for your time.

< HJOURNAL, 27715.NLS;1, >, 8-MAR-76 20:36 XXX ;;;; .HJOURNAL="JMB 8-MAR-76 20:16 27715"; Title: .H1="USER SERVICES REPORT: Training & Assistance at Bell-Canada - March 1-5 1976"; Author(s): Jeanne M. Beck/JMB; Distribution: /ARC-APP([ACTION] Please read, this is meaty and contains something for everyone in Applications) ARC-DEV([INFO-ONLY]) DCE([INFO-ONLY]) ; Sub-Collections: SRI-ARC ARC-APP ARC-DEV; AccessList: ARC-APP ARC-DEV DCE JMB; Clerk: JMB; .IGD=0; .SNF=HJRM; .RM=HJRM-7; .PN=-1; .YBS=1; .PES;

USER SERVICES REPORT: Training & Assistance at Bell-Canada - March 1-5 1976

Warning: this report is long and meaty.

1. TIME: March 1-5 1976 - 5 person-days by JMB

2. TRAINING, ASSISTANCE, and PERSONS CONTACTED [directory names in uppercase]

There was a schedule of times and subjects of planned training that Inez distributed to Bell users (27649,) but as Inez warned me, there was no way to get users to actually come to scheduled "courses". They were often out of town, in meetings, or working from home on any particular day. So each morning and afternoon, I went around to users checking on what topics I could get 2 or more people together for--sometimes I would work with individuals--and delayed topics for which all interested parties were not available; some topics, like MSG, were still postponed on Friday when I had to leave. The rest of this section describes the training sessions I did have with users.

March 1 - Discussed Output Processor and COM news with Inez MATTIUZ. She showed me some of their COM'ed documents, and we discussed their attitudes toward the quality of the documents they'd produced (the quality varies widely in my opinion). They are pretty disillusioned with trying any fancy COM stuff; they stay away from two columns. I suggested some things I'd learned about columns and Tabto's from working on Dave Potter's recent COM project. I taught advanced Output Processor stuff to Inez and Huguette MEADE. They were happy to hear about the PlexNum, PxI, PxF, PxP, PxIFirst, PxIRest, Numdash (for their Terminet printer), and other directives they didn't know much about. Gave a beginning Output Processor course to Lina NARDI, which Inez and Huguette sat in on (These three people are the ones who do the input and COM production of reports for all the other Bell users). Inez commented that Larry Day has ordered that any report or paper that is put into NLS at any stage has to be published through COM; she felt that this expressed a very definite committment to NLS by the management of the group.

March 2 - AM - Individual session with Mike KATSOULIS reviewing basic NLS stuff and getting his directory and initials file back together after months of inactivity.

He was especially interested in how to file away or delete journal items he has already read (he uses Print Journal). He is also interested in how to handle Snamsgs--delete, file elsewhere, read later--so wants to know MSG (the class we could never get people together for). I would have liked to be able to give a journal mail handling class to several users, but ended up helping many of them with the same stuff individually. I feel that most people need a mail-handling course several

months after they become users, and that we should put more effort into this (we concentrate more on sending mail than on receiving it).

Like many other rusty TNLS users at Bell, Mike understands about NLS structure and uses it successfully in moving, deleting, etc., in his initials file, but doesn't want to know about editing because his memos and papers go to Huguette or Lina for transcribing. He's very interested in the quality of the production of his stuff thru COM; I tried to talk him into indicating structure on his handwritten drafts, but he'd rather leave that up to Huguette. Usually she returns him an unstructured draft, then he tells her which points should be formatted below which, and then she restructures it if that seems necessary while formatting it for COM. Most other people who write papers don't like to read a draft until it has been formatted through the OP to look somewhat like the final version will look.

March 2 - PM - I gave Inez and Huguette an introduction to Content Analyzer patterns. I did not teach everything--only mentioned a little about combining elements and didn't mention ENDCHR or scan direction or negation, etc. They need to practice this somewhat before learning more features. They learned to invoke the Content Analyzer with the Set Content To command and to use Print, Output, or the Filtered Editing commands such as Copy & Move, with Viewspec 1.

Answered their questions about markers, file and directory protection, and Edit Statement. Does anyone in ARC have any comments about the status and use of the Edit Statement command? Some of the control characters <CTRL-A><CTRL-Q> haven't been working in this command.

March 3 - AM - Worked with Larry DAY and Gwen Edwards (GEDWARDS) on some Addressing, some Viewspecs, and on Journal topics.

March 3 - PM - Spent most of the afternoon working with Gwen and Huguette and Don ATKINSON (who came by off and on) on Editing, advanced Addressing, sending sequential files with SNDMSG, how to transfer sequential files into NLS and vice versa, and how to use the Message subsystem. This session held their attention well probably because they had a specific problem to solve and the topics I covered spoke to and solved that problem (it didn't seem like merely "learning").

March 4 - AM - Don ATKINSON showed me his work habits in TNLS and DNLS; he wasn't interested in devoting much time to "training", but I did show him some shortcuts while he was passing back and forth. He is a very restless and quick-learning person. He said that he has learned all he knows about NLS (which is quite a lot tho he has unusual methods) from the Glossary at 2 am.

March 4 - PM - Continuation of March 3rd afternoon session with Huguette and Gwen; covered how the journal files are organized and how to access old journal items, which led to topics on Archiving, getting back archived files (and their partial copies!), and dealing with file space allocation problems.

March 5 -

Worked with Mike BEDFORD on various specialized topics. Phil WEINTRAUB was a good example of the situation many users get into in terms of their learning NLS. Inez thought he greatly needed training for his development as a user. When I

asked him which sessions he was going to attend or if I could go over anything individually with him, he said that he didn't see the need to take time now; when he next had some problem or project, he would ask Inez to help him. Inez complains to me that they come to her with problems, she tries to train them, but they only want their particular problem solved and are not willing to invest time in learning.

3. APPLICATION (This is by no means the major application at Bell; I just include it because it's new to most of us):

The DATES Subsystem - a Management Tool

On Thursday morning Lina NARDI (who operates it) and Penny NAPKE (who wrote it) showed me Bell's Dates subsystem. Lina, or anyone else in Bell, goes into this subsystem of NLS and enters the dates a member of the Business Planning Group will be out of the office and why and when they will return. [Only Lina, the secretary, can change entries for another person.] This info is written on a file in the <BELL> directory. The subsystem can also be interrogated for the next ten mornings or afternoons all of a given group of people will be available in the office, used for planning meetings.

Penny also wrote a Process branch which Lina runs: it is used to send a message to everyone in Bell listing all the members who will be out of the office tomorrow and today. This Process branch uses the AUXCHAR procedure-replace to prompt the user for input at certain places, and it includes the running of a content analyzer program Penny wrote.

I have documentation for these capabilities if anyone's interested. My major conclusion is this relatively simple-minded effort shows that if Penny put her mind to it, and the resources were given, she could design a really powerful management application for Bell; I feel that the management backing is potentially there (also see--4f1).

4. ISSUES:

DEX:

Larry DAY and his boss, Don ATKINSON, both asked me to look into Bell's continuing problems with DEX. Partly because Inez was not in the office during most of the week, it was difficult to determine exactly what the problems were and how long they had experienced them. Evidently, DEX has not really worked well for them for over a year. During that time they have complained about, and we have fixed, various problems such as the up-to-date versions of cassette and dex programs getting archived, and the dex program introducing <NUL> characters. It was not clear to me until this trip, however, that it was still unreliable for them in spite of these fixes.

DEX was a major issue of NLS effectiveness for the managers of this group, especially Atkinson; their concept is that there was a period of time quite a while ago when DEX worked perfectly, and that for at least a year now, something hasn't worked and we have been quite unresponsive to their complaints.

[To be fair, however, I should point out that clear info about many of them did not come through to ARC; what often happens is that we tell users that we have fixed something and later ask them if things now work, they have not tested the process to find that they there are other problems, so we assume that all is now cool, they find out much later

about their troubles, conclude that we failed to fix things, etc.; this can go on forever.]

Day and Atkinson both are very aware of the potential for really efficient use of NLS resources that DEX represents, and it can be a lifeline when there is a real crunch in direct access. I also personally feel that DEX should be thought of as one of the major aspects of our service to users who are concerned with cost-effectiveness.

So, I tested DEX extensively myself on Thursday evening and Friday morning (a low load time and a high one) in their environment.

The DEX program, which converts a sequential file into an NLS file, works fine. The problems lie in the process of the Cassette program reading the tapes to make the sequential file.

The equipment they use are two cassette units which are quite similar to our ICP Termicettes, each of which is connected to a VU-COMP, a display terminal used for TNLS. I made several test tapes according to the instructions in our DEX USERS' GUIDE FOR THE ICP TERMICETTE (32952,), except for manually pushing the buttons to go into record mode and to record a gap on the tape.

My various experients suggest that the problem is that the equipment is running in continuous mode rather than in line-at-a-time mode (in line mode the program will stop reading at every <CR>, wait, and then start reading the next line; in continuous mode the whole stream of characters on the tape are shoved into the system as fast as they are read--it does not even pause between files), as the system is designed for. [MEH & JCP say that the design is characteristic not of the Cassette program but of input to the computer itself; I am looking into some suggestion that this is not immutable.]

I could find no equivalent to the Stop Control knowb on our ICP's which is used to select Line mode.

With Continuous mode the Cassette program can SOMETIMES get all the text read into the sequential file IF you are running during a time of low computer load, there is only one file on the tape, AND you stop the the tape manually and the Cassette program with <CTRL-C>. The rest of the time the result is NOTHING on the sequential file (and therefore you cannot use the DEX program).

When I explained the problem to Atkinson on Friday, he took the position that he wants the system to work with Continuous mode because that would cut down drastically the amount of connect time it would take someone to read in the tapes. The Casset:e program DOES take a long time to read a tape; evidently the pauses at <CR>s have to be long.

While we decide what to do about the Line/Continous problems (their equipment, our programs and computer) I'll try to work out with Inez some suggestions which MAY increase the possibility that tapes will be made into files. To that end, the secretaries should be trained to use the input instructions in our manual; I have the impression they're used to the standard straight input procedures for offline use of cassettes, and they don't clearly understand the conventions

DEX uses.
I collected other problems to solve for Bell users during the week. I will be working with Feedback and others on these.
Future training and assistance; these are things I did not get a chance to do:
Huguette wanted to learn all about making Tables of Contents.
Batch facility (especially when the new revision is formally announced)
MSG course

5. DISCUSSION:
While I was at Bell, it was announced that Phil FELDMAN was leaving the Business Planning Group and that Penny NAPKE was joining it. BPG is the main Bell group using NLS, and is responsible for it; sometimes people in other neighboring groups use NLS also (Penny is an example).
Huguette MEADE is the most sophisticated NLS user in Bell next to Inez MATTIUZ. Her job consists entirely of working for other Bell members on getting things into and out of NLS. In fact, there is a tendency to rely on her almost too much--other users do not bother to learn what is going on so they do not know how best to set up the work they give her. Also, when she is away for the whole summer (every year), the work just piles up. See also SGR's description of her role in her course report on the last Bell visit (32888,1e4).
Attitudes Toward Training: One type of comment I heard from several people seemed to partly explain why Bell users seemed not very highly motivated to attend training sessions. I would paraphrase it something like this: "We're not very susceptible to training because we have really taught ourselves NLS. We got very little training before SGR's visit last June, so we had to fend for ourselves, and feel that our use habits are too well ingrained to change." Thus it was difficult for me to help them to develop their proficiency. They do show more interest in brand new capabilities we offer, though (Batch, MSG).

OWW 11-MAR-76 21:15 27726

Information about ARC Documentation Production Research

Oliver Whitbey (OWW) put these questions to DvN as part of his effort to inform himself about NLS for the Editorial Procesing Center project. Dirk plans to provide brief answers with supporting references next week. He may come to some of you for help in this matter.

Information about ARC Documentation Production Research

I should appreciate the following type of information from you: 1

Any data you can unearth to relate the jobs we foresee in journal publication in the areas of: 1a

- executive editing 1a1
- redaction 1a2
- typography 1a3

to the load factor on NLS, assuming all users are doing the same kind of work. Will any of the experience with other ARC clients, such as Gunter AFS, enable us to get a handle on this relationship? 1b

A short explanation of the function of the full screen proofing capability now available through Office-1 (?) and an exposition of what you believe may be the changes needed to provide an interactive redaction capability, such as MAE's, for NLS using the Tektronix display. 1c

A general perspective on the experience ARC has now built up in document production with its clients. I'd much appreciate this description containing some background information on each client so that I can understand the viewpoint from which each regards NLS and the broad needs each is attempting to satisfy. 1d

A feel for the way in which each ARC client doing document production has handled the transition from a manual to an automated mode. 1e

A short catalog, with narrative descriptions, of the major improvements that are planned for NLS (Office-1) and that may influence the capabilities of an NLS-based test vehicle for this project. 1f

The way in which the load factor on NLS varies throughout the day and the week for the Office One machine and, if possible, for the BBN machine. In other words, what is the windfall % of each machine at times of the day when an East-Coast, Mid-west, and Far-west journal is likely to be using a "slot"? The variability in the estimates of the mean would also be useful to have, if they can be ascertained. 1g

If ARC has a written evaluation of UNIX for the document production job--I understand such an assessment was made--I would find it most useful in coming to a fair judgement of the suitability of the various possible test vehicles. 1h

Naturally, if ARC has extended or revised the Vallee report, I would find the results of such a study most interesting. 1i

In talking with Doug Engelbart about the advantages to the scientific community, I haven't yet formed a clear impression of the areas of communication on which his thoughts focus first and most heavily. I believe we all acknowledge that more rapid and facile access to records of past work could be of great use to, say: 1j

lawyers 1j1

scientists 1j2

administrators 1j3

legislators 1j4

Certainly getting these data (citations, abstracts, etc.) into a machine readable form should be done when the original work is published. However, I have a fear that putting everyone on-line to one another could be more disruptive than is discourteous use of the phone. It certainly is no substitute in its current implementation for close personal contact among collaborators. I should like to distinguish between groups the members of which are related in a systematic way by their responsibilities to a single organizational entity to solve a problem and those in which the members are really each seeking to solve different problems but are joining efforts in hopes of getting insights into their own particular problem. 1k

While we are personally intrigued by the technical nicities and social possibilities of computerized conferecing and information sharing, I feel we must recognize that non-computer scientists are interested first & foremost in their own work. Quite often they have to make sacrifices to share information with others. The benefits they reap are usually rather indirect and long in coming while the penalties are direct and immediate. My basic question is "What are the drawbacks to information sharing and how are the incentives to cooperate handled?" I'd like to have a balanced perspective on this problem, which is multifaceted, far reaching, and really significant. I'm not willing to accept that computer information sharing is an unalloyed benefit to the scientific community. 1l

assisted systems the operator is already entering the next job, while the operator at the magnetic tape equipment is watching the first tape play out..." (italics added)

This example illustrates the considerable benefits to be gathered from office automation, not only in word processing, but also in communication, information management, and document production. The office of the future is here today -- if we want it.

.Grab=10;TEXT PROCESSING AT THE CLERICAL LEVEL

The considerable costs savings and productivity increases described by Tartaglia are based on applications at the secretarial/clerical level. His analysis of automated word processing breaks word processing systems into three basic elements: the originator, the transmission element, and the keystroking element. These examples all involve cases in which the three elements are discrete; they are clear examples of the benefits which can be realized simply by automating current procedures.

This sort of automation leaves the executive comfortably ensconced in a swivel chair, dictating or writing a draft of a document. These words are then transmitted (on paper, on dictation equipment, over facsimile equipment, etc.) to a secretary or typist who enters the text into the word processing system, after which it is played back and returned to the originating executive for review and modification. All changes are then made by yet another keyboard operation.

As Tartaglia demonstrates, office automation even at this level can produce tremendous gains in productivity, with commensurate cost reductions, simply by reducing or eliminating redundant keyboard operations. Logically, however, only one step separates managers from full exploitation of current technology. Executives who, instead of writing or dictating, enter their thoughts and ideas from a deskside terminal, not only eliminate the redundancy inherent in the separation of the three elements; they also gain far greater control over the flow and form of their words.

Despite the widespread use of automated systems in other functional areas of the organization, the office -- particularly at the management level -- operates much as it did years ago. The primary medium for the flow of words and ideas whithin the organization is still the printed page; and although the ideas on the page may have been dictated into a central dictation system and transcribed by a member of the typing pool, the basic process has changed little. Origination, transmission, and keystroking (not to mention the redundant cycling through these steps during editing and revision) remain separate.

It is not the lack of adequate tools for office automation which guards the status quo. An excellent example of the scope and power of applications of technology to office operations is provided by NLS (oN-Line System), which is maintained and marketed by Stanford Research Institute.

For thirteen years the Augmentation Research Center (ARC) at SRI has been developing computer-based tools to enhance the abilities of individuals and groups working with knowledge. This research, under the direction of Dr. Douglas C. Engelbart, has been sponsored by the Defense Advanced Research Projects Agency, the U.S. Air Force, and other government agencies.

The set of integrated tools developed at ARC is termed the "Augmented Knowledge Workshop (AKW)." The focus has not been on

problem-structuring methods or on analytical capabilities, but on an overall environment in which people can work with information and ideas. This effort has been guided by an orientation toward the nontechnical user, such as the business executive. These tools have been applied with considerable success to the work of a diverse set of organizations throughout the United States and Canada. The growth and further evolution of this information environment looks highly promising.

The change implied here is far greater than simply the automation of the way we work. Techniques like those developed by Engelbart and his colleagues hold the potential for actually changing significantly the procedures and structures of organizational life. According to George Pake (as quoted in the June 30 issue of .Slant;Business Week.Slant=Off;), "There is absolutely no question that there will be a revolution in the office over the next 20 years. What we are doing will change the office like the jet plane revolutionized travel and the way that TV has altered family life." And he continues, saying that within 20 years his office will be completely different: "I'll be able to call up documents from my files on the screen, or by pressing a button... I can get my mail or any messages. I don't know how much hard copy I'll want in this world."

.Grab=10;AUTOMATION AT THE EXECUTIVE LEVEL

An organization's success depends on the effectiveness of its executives in guiding the organization through its complex environment. The scarcity and cost of executive talent imputes a high value to its efficiency. Organizations are becoming increasingly interested in providing a supportive working environment for their executives.

A large part of the executive's day involves working with information and ideas. Executives must be aware of the organization and its environment, through a wide range of contents and many levels of information specificity and permanence. They must solve unstructured as well as structured problems. They must work with others on ideas. Essentially, they are employed to think and to communicate. Advanced information-handling tools can play an important role in improving the effectiveness of the executive's use of valuable time.

Many organizations provide management with tools which are designed to help coordinate the flow of information. A management information system may help executives to get necessary information, no more, no less, pretty much when it is needed. Financial or accounting systems may keep the books, while still other systems are used to analyze business or market trends and forecast probable futures. To date, however, these systems have remained separate, and typically require specialized, trained personnel to run them. No one system either provides executives with the various information-handling tools they need, or even a coherent context through which they can reach these tools. Moreover, current systems tend not to be user-oriented -- that is, they are relatively difficult to learn and to use, sufficiently so as to discourage all but the most determined from aquiring sufficient skill in their use to garner any significant benefits.

The executive might use tools like those developed at SRI in many ways. Collecting information about the organization, communicating with peers, superiors, and subordinates, collaborative work without regard to the locations of the collaborators -- and yes, even text processing and document production -- all are tasks well within the

scope of currently available tools.

.Grab=10;WHAT KIND OF AUTOMATION?

.Grab=10;Magnetic tape (or disk) equipment

The vast majority of the word processing systems currently on the market fall into this category, which includes such equipment as IBM's memory typewriter, the near-classic Magnetic Tape Selectric Typewriter (MTST), and VYDEC's screen-oriented editor. These units are completely self-contained; to date, in fact, few if any provide even the option of communicating with computer-assisted systems.

Tartaglia (.Slant;Journal of Systems Management.Slant=Off;, August 1974) estimates that such equipment can usually cut overall costs betwen 25 and 30 percent. For some applications, this may be more than adequate; not all text processing tasks require or can even use the power and flexibility afforded by more sophisticated systems. Before committing an organization to the purchase of .Slant;any.Slant=Off; system, however, careful study should be devoted to the flow of text processing and document production through the organization. Generally speaking, magnetic tape equipment is relatively limited in its ability to grow and flex with the constantly changing needs of the company; moreover, its capabilities limit even its current uses.

Characteristically, these "stand-alone" devices are not designed for massive editing jobs. With the MTST, for example, basic editing is limited to the substitution of character strings (text) of equal length. Larger-scale editing changes require that the document be copied onto a second tape, manually stopping and starting the respective tapes when areas to be edited are reached, a process which closely resembles the way in which audio or videotapes are edited. Their major advantage is their cost; in terms of short-term costs, they represent the least expensive first step toward office automation.

.Grab=10;Computer-assisted systems

Until recently, the thought of using a computer as the heart of a word-processing system would have been discarded as hopelessly impractical. The rapid growth in the availablity of increasingly powerful and increasingly economical interactive computers, however, has brought radical changes to the field. William Vetter, Director of Advanced Planning and Research for the Franklin Life Insurance Company (.Slant;Best's Review.Slant=Off;, October, 1973) describes the result:

> At Franklin Life, a computer (word processing) system has been used for over five years. It followed an extensive use of automatic typewriters and it immediately lowered the unit cost to less than half. With practically no training, each operator began producing three times the volume produced on an automatic typewriter.
>
> To compare:

Automatic Typewriter	Computer
Equipment --$285/month	Keypunch --$110/month CRT --$75/month Computer --$60/hour
Output, words per minute --150	15,000

Cost per thousand words --$.73	$.066
Operator input, letters per day --150	450
Input cost per letter -- $.153	$.051

.Grab=10;As though unsure of the ability of these figures to convince the skeptic, Vetter goes on to list other advantages of the computer over the magnetic tape units:

1. Preworded letters and paragraphs can be stored on direct access economically..YBS=0;
2. There is no need to group letters by type.
3. The computer will "put on" the master letter in a few microseconds.
4. The work load can be smoothed out -- letters can be requested when the information is available and let the computer print and date the letter when it's time to send it..YBS=1;

Van Dam and Rice, in a survey of on-line text editing systems published five years ago in .Slant;Computing Surveys.Slant=Off;, summarized the advantages of the computer-assisted systems available at the time (it should be noted that the speed and power of these systems increased tremendously, with equally large cost reductions, in the five years which followed):.PlexNum=101;

easy access;.YBS=0;
immediacy of response;
ease of making hard copy without intermediate stages of typesetting, proofreading, resetting, reproofing, etc.;
reduced "turnaround" time for any type of file research and writing task;
common access to the same data base (this is useful for a pool of researchers or documenters working in the same area, or for common access to updated (project) management information;
"constructive plagiarism": the easy modification of previously written materials for present purposes (as in the writing of papers, contracts, proposals, or brochures);
great simplification of document dissemination and storage -- no hard-copy bulk, but any degree of archival protection desired;
far greater flexibility for browsing and linking text fragments than with manual methods of working with hard copy; and
relatively modest cost for all this increase in activity and efficiency, when compared with all aspects of present systems: writing delay, retyping, proofreading, typesetting, revision of galley and page proof, printing, binding, distribution, storage (and subsequent inaccessibility due to distance, lack of shelf space, poor indexing, borrowed or lost copies, etc.). The user cost includes the machine time used (typically a 5% or smaller rate of CPU utilization/user time) and the rental or purchase cost of the terminals employed..YBS=1;

Of course, selection of a computer-assisted system is no easier than choosing a hardware-oriented system. ATS, Wylbur, Hpertext, Script, TECO, and a host of others wait in an array of resources more likely to bewilder than to inform the potential user of their relative merits.

.Grab=10;COST-EFFECTIVENESS

Clearly, computer-assisted systems have tremendous advantages over "stand-alone" systems. They offer greater power, speed, and flexibility; in addition, they hold the promise of improving productivity not only at the clerical level (by automating the traditional word-processing cycle) but also at the professional or executive level. Imaginative applications of such systems can provide not only vastly improved text processing and document production; they can also give executives the tools they need to move at ease through all the varieties of information handling, retrieval, and dissemination tasks that in the final analysis constitute their work. Management information systems, financial planning and budgeting, word processing, communications, and record keeping can be merged, with a desk-side terminal providing the professional with easy access to whatever tools are needed for the job at hand. Equally obvious are the short-range economies of the "stand-alone" system. Not only do such systems require a smaller initial investment; they also exert less impact on organizational structure, personnel, and procedures. The full potential impact of computer-assisted tools was described years ago by Douglas Engelbart, Director of SRI's Augmentation Research Center, whose system, NLS, is much more than just a text editor; Engelbart's goal is to provide a new way of thinking and working by fully exploiting the power of current technology:

> We are concentrating fully upon reaching the point where we can do all of our work on line -- placing in computer store all of our specifications, plans, designs, programs, documentation, reports, memos, bibliography and reference notes, etc., and doing all of our scratch work, planning, designing, debugging, etc., and a good deal of our intercommunication, via the console.

Dr. James Emery of EDUCOM describes two components of system cost-effectiveness:.PlexNum=101;

> .Slant;Do the right thing.Slant=Off;: that is, get the job done. The more power and flexibility a system has, the more it can accomplish; today's best computer-assisted systems can handle a tremendous range of jobs of virtually any size, scope, and complexity -- jobs which simply cannot be approached with "stand-alone" tools.
>
> .Slant;Do the thing right.Slant=Off;: that is, get the job done with maximum efficiency. Although high-power tools like SRI's NLS can do both large and small jobs, the smaller jobs can be performed just as well, and at lower cost, by less powerful, "stand-alone" tools.

The optimum system, if it is to meet reasonable criteria of cost-effectiveness, must be able to provide each user (from the clerical to the executive level) with precisely the power needed for the job at hand -- no more, no less. "Stand-alone" systems, lacking the power for large jobs, cannot meet this criterion; computer-assisted systems, because of their cost, also generally represent a compromise in this respect.

Two approaches, both currently feasible, provide a resolution to the conflicting requirements of power and economy -- hierarchical systems and hierarchical networks. Both approaches rest on a common principle: they are multi-level systems which allow the user to reach out from a single piece of equipment (usually a deskside terminal) to use precisely the power needed to efficiently perform the job at hand. Such systems allow any user to perform tasks

ranging from basic text entry (the initial keyboard operation required by all systems) up through tasks as demanding as complex search and retrieval operations, heavy document editing and revision, and photocomposition, and to do so in a cost-effective manner.

.Grab=10;Hierarchical Systems

Hierarchical systems are essentially hybrids. They represent a melding of computer-assisted and "stand-alone" systems; that is, they use relatively low-power systems (e.g., magnetic tape units) for initial text entry and simple editing, and rely on the computer to supply both efficient long-term storage and retrieval and the power needed for large-scale editing and formatting operations.

Efficient hierarchical systems can be built primarily through careful selection of the equipment that will comprise its component parts. The basic rule for equipment selection is simple: choose only equipment that provides a communication capability. Such equipment (e.g., IBM's Communicating Mag Card units, Vydec's screen-oriented system) provide local text capture and reasonable editing capability; in addition, they allow the user to connect (via dial-up phone connections) to the computer when additional resources are needed.

Ideally, such a system should be as modular in design as possible. That is, the frontend equipment -- the equipment which provides the basic interface with the user -- should be as simple and self-contained as possible. This is necessary if the user is to perform economically those jobs (e.g., interoffice memos) which require little if any automation. Although higher-powered units like the Vydec system will of course perform such tasks quickly and pleasantly, costs will be higher than they need to be.

Equally important is the need for the user to be able to use this same frontend equipment to reach out for additional power when it is needed. This process should be as simple and straightforward as possible; in fact, the ideal system would automatically provide precisely the power needed by the user at any given moment. Such flexibility is beyond the capabilities of a modular or hierarchical system (but within those of a hierarchical network -- see the discussion below); nevertheless, the process of getting extra power should be a simple one. Here again the answer lies in the selection of relatively self-contained frontend equipment.

Although the equipment comprising a hierarchical system would vary from one organization to another, it would generally be oriented around typewriter terminals connected to tape cassette units. Such terminals are relatively inexpensive, can provide excellent print quality at up to thirty characters per second, and, when linked to tape units, can be used for local text capture and simple editing. Anderson-Jacobson, for example, markets terminals expressly designed for text processing applications; these terminals not only support the functions just described, but can also be used as sophisticated office typewriters when no automation whatsoever is needed (for example, for inter-office memos).

Other useful equipment includes portable terminals to be used from remote locations, display (CRT) terminals for more powerful editing and efficient browsing through on-line files and data bases, and high-speed printers for high-volume, large scale document production. All of this equipment is readily available;

indeed, many organizations already have made substantial investment in terminals and related equipment. When used in conjunction with powerful computer systems like SRI's NLS, such tools combine to form a true hierarchical system.

.Grab=10;Hierarchical Networks

Hierarchical networks (c.f. .Slant;Datamation.Slant=Off;, February 1975) are conceptually similar to the hierarchical systems described above. Instead of local magnetic tape (or disk) units providing a low-level editing and storage capability, however, the hierarchical network uses minicomputers to perform the same functions.

Basic user interface with the system is provided by the same types of terminals described above, but without self-contained magnetic storage and editing facilities. Minicomputers providing these capabilities are connected to central facilities which provide whatever additional supporting power is needed on a time-shared basis. The major difference to the user is that, while text entry and editing in a hierarchical system generally differs considerably according to whether the process is in a "stand-alone" or computer-assisted mode, with the choice made in every case by the user, the individual using a hierarchical network may be entirely unaware of the system level at which he/she is working. Initial typein, editing, retrieval, composition, and communications, may all be handled by a hierarchical network using one basic set of commands in a manner that is virtually transparent to the user.

In other words, the hierarchical system is a set of related tools, having a common purpose and supported by a common systemic framework, but requiring different procedures according to the level of power and flexibility needed. The hierarchical network, on the other hand, appears to the user as a single coordinated system, providing him/her with all basic information handling functions.

Although several hierarchical networks are currently operational, none supports the capabilities needed for office automation. Current technology, however, is fully capable of supporting such an application; in fact, developmental efforts now under way at SRI's Augmentation Research Center are directed toward this goal.