****

The Design and Implementation of NewsPeek; An Interactive News Gatherer for Future Communications Systems

by

Steven Ford Labalme

****

Submitted in Partial Fulfillment

of the Requirements for the

Degree of Bachelor of Science

at the

Massachusetts Institute of Technology

August 1981

Signature of Author.....
Department of Electrical Engineering and Computer Science, 7 August 1981

Certified by.....
Thesis Supervisor

Accepted by.....
Chairman, Departmental Committee on Theses

The Design and Implementation of NewsPeek; An Interactive News Gatherer for Future Communications Systems

by

Steven Ford Labalme

Thesis Supervisor

Nicholas Negroponte, Professor of Computer Graphics

Abstract:

This thesis describes work done at the Architecture Machine Group at MIT in the field of Media Technology. An interface to an information intensive network is discussed and proposed. A prototype implementation of such an interface has been designed and built and is currently being used as a test bed for the development of a future Home Communication Set.

This work was supported by a grant from the Office of Naval Research.

Table of Contents

We are, as inhabitants in the modern world, being continuously bombarded by news and other information. As concerned citizens, we try to weed through this vast amount of data in order to become better informed about the issues of current social and personal interest. Today, we accomplish this goal by watching network news broadcasts, listening to the radio, reading one or several of the various printed forms of media generally published daily or weekly, or by word of mouth from friends and aquaintances. Each of these methods of communication has its advantages and disadvantages, but none of them is (comprehensively) adequate to do the job required today of information retrieval, communication, and dissemination.

* Television news brings us up to the minute pictures and visual information, and along wth Radio carries current news and stories.

* Newspapers and magazines generally can give a broader perspective, or perhaps a more detailed, although often personally biased, view.

* Libraries provide an excellent way to access printed media, and offer the user the advantge of being able to access past as well as current information.

* Word of mouth is still the most powerful method of communication available to us, but it is heavily restricted

by the number of people we come into contact with, and the generally limited scope of world and cultural interest available in an average persons circle of friends.

I say that word of mouth is powerful mainly because it is generally the best method of quickly and accurately conveying ideas. This is so because of the immediate feedback that one receives from a listener, and because of the *two-way* path of the communication that exists and is conspicuously absent from the other methods of communication/information retrieval mentioned above.

This thesis proposes the development of a new form of information retrieval which incorporates hopefully the best of all the possible avenues of communication. It would consist of a personal computer of fair to subsantial power connected over some communication channel (telephone lines, light guides, satelite) to electronic libraries, schools, shopping centers, banks, and other users, as well as, of course, entertainment stations. In essence, it would be an extension of both the television set and the telephone, facilitating a bi-directional interchange of data. Such a system could be called an HCS, or Home Communication Set, a term coined by John Wicklein, the current president of the Public Broadcasting Network in his book Electronic Nightmare.

Recently, the philosophy of System Design has been

changing along with new advances of technology that are serving to dramatically lower the cost and space requirements of computing power. An awareness of the need to supply user flexibility without tieing up channels of communication with redundant or simply useless information has helped pave the way for the current dogma of distributed processing. This concept incorporates a small, perhaps personal computer with user needs built in, or better, specified by the user, and a clear, quick, and precise communication protocol between it and other computers which are connected to it through a network. This allows the user to be in control of the way she is interacting with the system, and thus gain confidence and efficiency in its use.

Furthermore, it is becoming increasingly economically productive for a company to compile and provide to the consumer large data bases of information, such as news, weather, legal codes, or even classroom instruction. This last is made possible, and the others more useful, do to the bi-directional transmission capabilities of the system. This exciting prospect, that of being able to transmit a reply to an article or make a statement on an issue, as well as receive information from a greater variety of sources, will help to open up a whole new world of learning and information.

However, a tool which can provide unlimited access to informational resources can be used as well as terriby

abused. To insure that malicious misuse of such a system is kept to a minimum without placing undue and possibly dangerous restrictions on how it is to be used, there are several important issues which should be kept in mind during the design of a large scale integrated system.

* The user interface must be easy to use, comprehensible, and easily customized to user preference. It must also be low cost and available to all.

* Access to public information must be guaranteed to all. No one should be denied access due to reasons of class, race, political beliefs, sex, et cetera.

* Private information, e.g., personal letters sent by means of an electronic mail, must be guaranteed private and untappable. [note: This can be accomplished using current state-of-the-art public key encryption methods.]

* No single interest group, such as the government, military, or private corporation, should have the ability to control the flow of information on the system, or have any influence upon what is and what is not generally available.

All of these guidelines should be completely adhered to or we lay ourselves bare to manipulation and exploitation for the good of some special interest group. As people begin to learn how to use the system for learning and

dissemination of information, and with these precepts as a foundation, we will be able to move towards a better informed public, decentralization of decision making, and perhaps a more coherent public policy along socially useful lines.

This thesis describes the design and implementation of an interface between a person wishing to become more informed about some subject and a large computer-indexed data-base of news and legal information. The interface has been implemented at the Architecture Machine Group at the Massachusetts Institute of Technology in a generalized fashion so as to allow easy extensibility and user customization. One of its goals has been to seemingly increase the bandwidth of the communication channel by providing an "intelligent" interface which can store the intermediate results of a path of inquiry or automatically extend it, and by providing automatic (key-word) access to an optical videodisk which will have a large number of high-resolution color photographs, charts, and maps which are displayed in mixed format along with quality (two-bit font) text. Another goal, as with all systems designed at Arch Mac, is total ease of use, achieved through integrated use of color monitors with touch sensitive displays, two-way voice communication, and other human-oriented input and output (communication) methods.

The data base that we have interfaced to is a high-technology key-word access information retrieval system provided by Mead Data Corporation. It contains on the order of a trillion characters of text in original format drawn from the three major domestic wire services, major domestic and Japanese papers and magazines, (with the notable exception of the New York Times) a plethora of legal case

reports, and the entire Encyclopedia Brittanica. Their system is relatively pleasant to use with their standard terminal, but it lacks pictures and the generality to connect to other users or data banks. Also, in general, it has a very formalized set of commands, and thus lacks the flexibility we are striving for.

An important feature of the interface is the ability to comment on (or even annotate a personal copy of) an article that one feels strongly about, and then mail it to one's friends. An extension of this which is incorporated in my system is the ability to store annotations on the Mead Computer so that anyone interested can access them. In essence, guaranteeing that one's "letter to the editor" is published. This feature is the first step toward decentralization of information -- a desireable goal in this new age of information.

The first and most important design consideration is that the system have complete two-way communication capability. This allows for an interactive, building approach to information retrieval where one makes a generalized request for information, gets feedback in the form of text and/or graphics, and is then able to, on the basis of this, continue the line of inquiry deeper by describing a more specific search goal or by modifying the current request in order to explore a new avenue of thought.

This is accomplished through the use of an interactive touch-sensitive display, a computer controlled videodisc system, other state-of-the-art interactive technologies, such as voice recogntion and speech production, gesture amplification, contextual modification, and, (yes, Martha) a keyboard.

Suppose one wishes to find out the cost of carrying cargo on future space shuttle flights. First one selects a library to draw upon. With the LEXIS/NEXIS system supplied by MDC, we might decide to restrict our search to "WIRES" which is a combination of stories carried by AP, UPI, and REUTERS wire services. Restriction of libraries to search will save time and money, for every story found will carry a surcharge on both. Then one enters the general goal, say, "Space Shuttle and cargo" from a keyboard and the computer "answers" by displaying on the monitor the first page of text from the most recent addition to the selected library, and at the bottom will be a command area which will tell the user that she is at the first of (say) 37 stories found, and where she can ask to see the next or previous page or story, to modify or quit from the search, or to move to a different search keeping present information in a temporary storage cache.

Now, as one browses through the first few stories, we find one which mentions "consumer costs" and "touch" this phrase which informs the system to restrict the search to

stories which have this key in them also. By iterating this approach, one finally arrives at a concise search stratagem, which is useful if perchance our selected library falls short of satisfying our curiosity. At such a point, we can simply broaden our horizens by including other libraries to search from.

A nice feature made available because of the power of one's personal HCS is the storage of search topics. If one runs short on time, or just wishes to be kept up to date on some topic, a search strategy can easily be stored, and re-entered at a later date. This could, for the case of being kept up to date, even be done automatically by the HCS every day, such that when one wished to find out "what's new," the information is already waiting.

An extension of the idea of carrying over search topics lies on thefield of research, which would become a hobby of anyone who has access to an HCS (which eventually should become guaranteed by law to everyone, see below). Once fully integrated, with many private and public libraries on line, one could hopefully find out just about "anything about anything," but it would take time not anly to learn where to search (i.e., who has the information) but also what exactly it is that one is interested in. It would be quite an interesting, and rewarding learning experience.

Communications between the user, the Nexis daemon, MDC's LEXIS/NEXIS, and various other routines in the system usually occur as operations upon one or more of four segment-sized data bases. [see Appendix A] Two of them, nexsuper and nexinput are 'owned' by the Nexis daemon which has total impunity in their usage. They are also initiated by the user-controlled nex$ routines, which write to nexsuper to inform the daemon of user presence or to initiate a search, and reads from nexinput when the daemon signals that a page is ready to be read.

As one can see by a cursory inspection, nex_page and nexinput are very similar, and are indeed somewhat redundant, for proper usage dictates that a page in nex_page be allocated by the user process before a request for information be made, so it would seem easiest for the daemon to write directly into nex_page as data comes in. But this would require that the daemon have every user's instance of nex_page initiated, limiting the number of users to around five, because of the restrictions MagicSix imposes on the number of segments which can be initiated in an address space. So when a page (in nexinput) is filled, the daemon signals the user that his page is ready, and the user, who has nexinput initialized (there is only one) copies it into her own nex_page buffer.

Every user has her own copies of the other two, nex_user and nex_page, are the user's sole property, and

describe the current state of the user's line of inquiry and act as an intermediate buffer for page information as received by the daemon, respectivly. As one can see by a cursory inspection, nex_page and nexinput are very similar, and are indeed somewhat redundant, for proper usage dictates that a page in nex_page be allocated by the user process before a request for information be made, so it would seem easiest for the daemon to write directly into nex_page as data comes in. But this would require that the daemon have every user's instance of nex_page initiated, limiting the number of users to a maximum of around five, because of the restrictions MagicSix imposes on the number of segments which can be initiated in an address space. So when a page (in nexinput) is filled, the daemon signals the user that his page is ready, and the user, who has nexinput initialized (there is only one) copies it into her own nex_page buffer using nex$save_page or nex$mangle.

There are three major parts to the design of the interface:

1) The Hardware. This consists of an auto-dialer to make the link with Mead Data Central's LEXIS/NEXIS system, and a half-duplex RS-232C (standard) interface to one of the Architecture Machine Group's Interdata 3220 or 3230 computers running the MagicSix Operating System.

2) The Nexis daemon and I/O buffers. A special free-running process known around MIT as a daemon) has been designed to

handle the allocation of communication, i.e. telephone, lines, buffering, formatting and transmitting of user requests (queuing them until a free line can be found) buffering returned page information, and signalling the appropriate user when a reply is complete and ready to be copied into the user space. The Nexis daemon utilizes two special segment sized shared data bases to aid in interprocess communication.

3) The NewsPeek user interface. NewsPeek is a generic term for a series of programs to facilitate comfortable user interactions. It is within the domain of NewsPeek to handle the formatting of an output display (with or without videodisk supplied images), the interpretation of data gathered from a touch sensitive display, and other user interfaces.

As the system now stands, NewsPeek has only a minimum of functionality, but is designed in a modular, object oriented manner such as to make the addition of new features as straightforward as possible.

A rather large collection of routines exists for communicating intent from NewsPeek to the Nexis daemon. These are known within the system as 'nex$' routines, and one exists for every command that can be entered from a standard LEXIS/NEXIS terminal (transmitting), along with a smaller collection for inspecting and modifying the daemon's

state (inquiry), as when a new user enters into communication with the system. [See Appendix B]

[important note: The user should NEVER attempt to touch the daemon data bases nexsuper or nexinput other than through the use of nex$ routines.]

An important feature of nex$ routines is that they may be used in either of several fashions, according to personal preferance and efficiency requirements. The various formats of a 'transmitting' nex$ routine are as follows:

```
     dcl  nex$transmit entry options (variable)
1)   call nex$transmit (uname, xid, "foo")
2)   call nex$transmit (uname, xid, "foo", rid)
3)   call nex$transmit (uname, xid, "foo", page_num, rid)
or
     dcl nex$transmit entry options (variable) returns (fix)
4)   rid=nex$transmit (uname, xid, "foo")
5)   rid=nex$transmit (uname, xid, "foo", page_num)
```

[where uname is the user's name, xit is the transmit id, "foo" is the search request, page_num is an index into the user's page buffers, and rid is the temporally unique request id.]

Method 1 is the simplest to use, but does not return rid, which acts like a general code in that if negative, something went wrong. The rid (request_id) is the user's handle as to where the Nexis daemon is buffering her requests, and in fact may be used to help identify a particular search. However, use of $save_that_page and $mangle (used to either save or kill a page on the signal 'bufr_rdy') preempts the need for the user to have to worry about rid's value. Many find it comfortable to have the code's reassurance that all went well with the last

operation, though.

Methods 2 and 4 are thus slightly safer, and when rid is positive, it points to the entry in nexsuper.xbuf which is buffering the request.

Each of these methods (1,2,and 4) assume that the user does not want to save the returned page, although the daemon will still signal "bufr_rdy" to the user when it is done.

[note: calling nex$dont_bother will keep the daemon from signalling the user if she is not interested. Call nex$bother_me will reset this, and is the default]

If, however the user is interested, she should subsequently call nex$save_that_page (abbreviated $stp) with an index into the nex_page buffers as an argumant. Alternatively, she could use methods 3 or 5 which accept a page_number as part of the original call. If the page_number is less than or equal to zero on the call, the page is not of interest and will not be buffered; otherwise, a call to nex$mangle will save it in the right spot.

Use of the daemon increases the potential performance of MDC's LEXIS/NEXIS system in several ways. First of all, it allows type ahead, for it buffers all input and output requests. This also allows for multiple users on a single system, although if only one line were actve performance would degrade considerably.

Another important feature is that we will no longer be tied to a standard text terminal. Instead, we will display results of searches on a color monitor, in quality, two-bit fonts, with key words highlighted in color. A touch sensitive display will provide dirict control over moving through stories, or expanding or compressing the scope of the search. Other I/O techniques will also be explored in future work, such as voice (using the NEC voice recognition device and the Votrax talker), gesture, and contextual specification.

Multiple lines can be accessed at once so that parallel searches can be made on different data bases. For example, we will soon be installing an AP picture description data base in an MIT private file at MDC. Then, when we make a request for a story about, say, Kissenger and Iran, we will simultaneously access the AP data base which will return the frame numbers of a map of Iran and a recent photo of Henry Kissenger on a AP videodisc. Thus, the final display of a particular search might well be a collage of text provided by NEXIS and graphics provided by a local

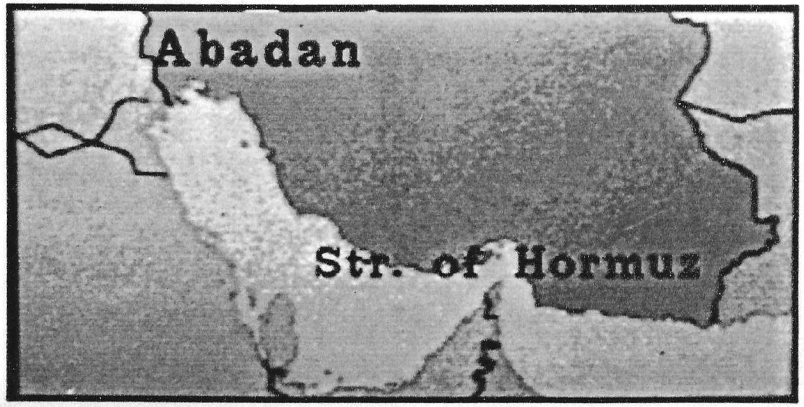
computer controlled videodisc system. [See graphic, next page.]

AP Washington 1/19/81

That would have required an appropriation by Congress because there is no legal way for the U.S. government to attach the shah's money. Moreover, the Carter administration's own estimate of the shah's wealth was only a small fraction of Iran's figure.

The agreement also pledged the policy of the United States will be "not to intervene,

Henry Kissinger - AP photocolor



Southern Iran - R. R. Donnelley

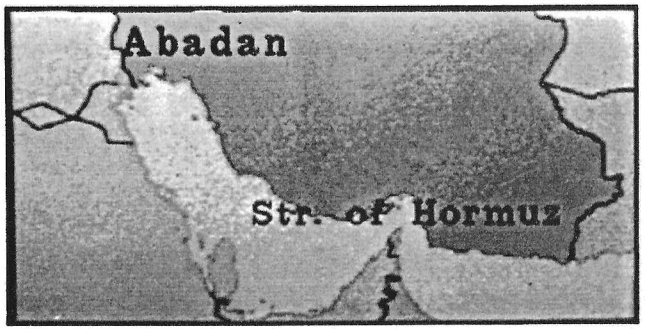
directly or indirectly, politically or militarily in Iran's internal affairs."

Some analysts, former Secretary of State Henry Kissinger among them, say the United States had no business bargaining in the

AP Washington 1/19/81

That would have required an appropriation by Congress because there is no legal way for the U.S. government to attach the shah's money. Moreover, the Carter administration's own estimate of the shah's wealth was only a small fraction of Iran's figure.

The agreement also pledged the policy of the United States will be "not to intervene,



Southern Iran - R. R. Donnelley


Henry Kissinger - AP photocolor

directly or indirectly, politically or militarily in Iran's internal affairs."

Some analysts, former Secretary of State Henry Kissinger among them, say the United States had no business bargaining in the

Variable descriptions: [See Appendix A]

Each variable in the data bases is commented, but some deserve further explanaton.

The first thing to note is that each is based on a linkage$foo external pointer; that is, each data base has its pointer made an external static variable, so that variables within it may be referenced without having to refer to its pointer. This is possible because there is at most one instance of any of these data bases in a process.

Looking at nexsuper first, note that the data base is organized into four main sections which correspond, respectively, to a table of user information, statistics of id number usage, phone numbers and other related stream information, and the circular transmit buffer and its associated control variables.

```
* ulock bit(32)
* ilock bit(32)
* llock bit(32)
* xlock bit(32)
```

Each of these sections has a read/write lock which is used before any process (other than the daemon) can write to its associated section.

```
* max_xxxs  fix
*  xxx_count fix
```

Each of the first three sections has its maximum size

set by Ninit (the Nexis daemon's initialization procedure) and xxx_count reflects the number of xxx's currently active.

The first section in nexsuper is the user information area which keeps track of who is logged into the system. It also contains fields which allow for easy user access to the nexinput page buffers.

* user[<unum>].ridx fix

The ridx(request_id) is used by the daemon and the user to keep track of pending requests. When a user initiates a request, it goes into the next available spot in the transmit buffer (xbuf, see below) and its index is returned as the request_id. When the daemon finishes reading in a page of information, it checks user[<unum>].ridx. If it is negative, (which implies that no buffers are pending) we set it to the page's ridx, which was set when the page was allocated, and then 'signal user[<unum>].name "bufr_rdy"' which alerts the user to the fact that a page is ready to be read.

* user.[<unum>].crid fix

Used by nex$mangle is set to the "current" rid which corresponds to the xbuf entry whose input is ready to be read by the user.

* user[<unum>].bother bit(1)

This is, by default, set to true. What this variable

controls is whether or not the user wishes to be signalled that a page has been received even if the user is not interested, (i.e., not planning on saving the page).

The id numbers and records of the user comprise the second section of data in nexsuper. One hundred id numbers are used so that multiple search contents can be saved while moving from one contour to another, or while multiplexing users along a single telephone line. Most of these are self explanatory, but additional clarification may be helpful with a few.

* id[<inum>].in_use bit(64)

This is set to all zeros when the id is freed by a user. Other than that, it contains the time, in bit(64) format, of when this id was last used.

The third section is the phone number list. In a future version of this system, MDC will supply us with a dedicated 9600 baud trunk line which we will then time-demultiplex into up to 16 separate 1200 baud lines. In this manner we could run parallel searches, one of the major advantages of this system. The values stored under line[lnum] are generally self-explanatory, except for io_ptr, ioname, device, devptr, and handle which are used by the daemon and MagicSix to facillitate I/O stream communication.

The final and most critical section of nexsuper, is the xbuf (transmit buffer) and its associated control variables.

* xlock bit(32)

Another write lock used by the nex$ routines to insure that only one process writes to the buffer at a tiem.

* xsize fix

xsize is set to 255, which thus allows 256 entries in xbuf, for it starts at zero.

note: xsize should always be a power of two minus one.

* xmask bit(8)

This is used to aid in modulo (xsize) addition. It is defined on xsize and is ANDed with xhead or xtail when these are incremented. Thus xbuf is a circular buffer. Note that, because based storage cannot use the "defined" attribute, xmask is actually commented out and is defined in programs that use it. The same holds for

```
xhack defined xhead
xtack defined xtail
xcack defined xcidx
```

* xhead fix
* xtail fix
* xcidx fix

These point to the head, tail, and current index of the transmit buffer, respectively.

* xhwrp bit(1)
* xtwrp bit(1)

These flags are (were) used to inform the daemon of wrap around, but are currently not used.

* xbuf[0:1]

The transmit buffer xbuf is dynamically allocated. On MagicSix, one cannot define a segment over 16k bytes in size, but dynamic allocation can go to 64k. So xbuf is defined here of size two, but it will grow to xsize (255).

* dserved bit(1)

dserved is not used in the current version of the Nexis daemon. However, it will provide the control necessary to implement intelligent look-ahead in search request serving. What this means is, in a fully implemented system with multiple lines to MDC, one might wish to skip the lowest (oldest) available request to transmit so that another may be transmitted on the current id, for increased efficiency. 'dserved' would then let the daemon know whether or not a particular entry has been served.

* uname char(8)vary
* u_num fix
* command char(128) vary

These are all set by the nex$ routine so the daemon knows what to do and for whom.

* idnum fix

idnum, also known as the xid (transmit_id) is also provided by the nex$ routine, and is a number from 1 to max_ids that points to the id number to use.

* ipage fix

ipage points to the page in which to save results of a search. If ipage=0, then the user is not interested. If ipage<0, then the page is full and ready to be read.

Appendix A

```
/*
 *   Nexis Daemon database      *******> DON'T TOUCH !!!!!!! <*******
 */

dcl   1 nexsuper based [linkage$ns_ptr],

/*
 *   login names of users and pointers to their respective user_contours
 */
        2 ulock bit(32),              /* lock for adding users        */
        2 user_count fix,             /* logged in users              */
        2 max_users  fix,             /* originally set to five (5) */
        2 user [16],                  /* MagicSix can't handle more */
            3 name char(8) vary,        /* user login name          */
            3 when bit(64),             /* when user logged in      */
            3 rid  fix,                 /* request id (=> xbuf)     */
            3 crid fix,                 /* current rid              */
            3 bother bit(1),            /* bother me with every
                                           end of transmission */
/*
 *   keeps track of id number usage
 */
        2 ilock bit(32),              /* lock for id allocation       *
        2 id_count fix,               /* ids in use                   */
        2 max_ids  fix,               /* number of legal ids          */
        2 id [100],                   /* for parallel operation       */
            3 in_use bit(64),           /* true if id busy                */
            3 uname char(8) vary,       /* ...tells us by whom.           */
            3 line_num fix,             /* 0 if not current               */
            3 id_number char(7),        /* id number (eg, 981125b)        */
/*
 *   phone number list
 */
        2 llock bit(32),              /* line lock                    */
        2 line_count fix,             /* logged in lines              */
        2 max_lines  fix,             /* originally set to one (1)    */
        2 line [16],                  /* up to 16 lines via T16 mux */
            3 active bit(1),            /* true if active                 */
            3 id_num fix,               /* transmitting on which id       */
            3 x_ridx fix,               /* pointer to xbuf entry          */
            3 inpage fix,               /* input page buffer offset       */
                                        /* = 0 if not interested          */
            3 io_ptr ptr,               /* iocb pointer                   */
            3 ioname char(8),           /* stream name                    */
            3 device char(8),           /* device name                    */
            3 devptr ptr,               /* device ptr                     */
            3 handle ptr,               /* receive handle                 */
            3 phone_number char(10) vary,/* hello? pronto! pronto!        */
```

```
/*
 *   this area is the circular command queue
 */
        2 xmit_hdl ptr,                 /* transmit hane for ring0 stuff     */
        2 xlock bit(32),                /* buffer lock                       */
        2 xsize bit(16),                /* xbuf size originally = 256        */
/*      2 xmask bit(16) defined xsize, */
        2 xhead fix,                    /* top of circular buffer  ( FI )    */
/*      2 xhack bit(16) defined xhead, */
        2 xtail fix,                    /* end of circular buffer  ( FO )    */
/*      2 xtack bit(16) defined xtail, */
        2 xcidx fix,                    /* process_message index pointer     */
/*      2 xcack bit(16) defined xcidx, */
        2 xhwrp bit( 1),                /* true on head overflow (modulus) */
        2 xtwrp bit( 1),                /* true on tail overflow (modulus) */
        2 xbuf [0:1],                   /* command queue (max size = 256)*/
            3 dserved bit(1),               /* been sent to Dayton       */
            3 u_num fix,                    /* user number               */
            3 idnum fix,                    /* which id ?                */
            3 ipage fix,                    /* nexinput page number      */
            3 command char(128) vary;       /* xmit moves user buf here  */
```

```
/*
 *   common area for input page buffers
 */

dcl   1 nexinput[1] based (linkage$ni_ptr),

          2 free bit(1),                /* huh ?                        */
          2 read bit(1),                /* ready to read ?              */
          2 ridx fix,                   /* xbuf entry number (rid)      */
          2 lrcs char(2) vary,          /* LRC count (transparent)      */
          2 blen fix,                   /* buffer length                */
          2 bufr char(1920) vary;       /* actual page buffer           */

dcl (linkage$ns_ptr,
     linkage$ni_ptr) ptr ext;
```

```
/*
 *   Each user will have her own contour set.
 *   Contours will allow a user to "detach" from one search,
 *        then initiate another with the ability to return
 *        to the original search (with varying levels of accuracy).
 *   [ this structure is 3200 bytes long -- will allow up to 20 contours ]
 */

dcl   1 nex_user based [linkage$nu_ptr],

        2 my_name char(8) vary,          /* Who...who are you??              */
        2 user_number     fix,           /* all you are is a number to me    */
        2 contour_count   fix,           /* number of active contours        */
        2 max_contours    fix,           /* maximum number of contours(20)*/
        2 max_level fix,                 /* maximum number of levels (6)     */
        2 ccontour fix,                  /* starts at one +(to set nex_page)*/
        2 contour [1],
          3 desc char (32) vary,         /* contour descriptor               */
          3 xid fix,                     /* let's keep track of who we call */
          3 libr char (10) vary,              /* nexis, etc    */
          3 file char (10) vary,              /* wpost, etc    */
          3 clevel fix,                       /* current level */
          3 level [6],
            4 request char (128) vary,             /* request text          */
            4 rid fix,                             /* points to xbuf entry  */
            4 no_stories fix,                      /* number of stories     */
            4 story    fix,                        /* story num in clevel   */
            4 mode     fix,                        /* display mode          */
            4 no_pages    fix,                     /* # pages in this mode */
            4 page     fix,                        /* page  number          */
            4 roll     fix,                        /* roll index ( + or - )  */
/*
 *   quick and dirty return to old search
 */
            4 doc_number    char ( 50) vary,   /* nexis file and doc numbers */
            4 np_idx  fix;                     /* first page index in nex_page */

dcl linkage$nu_ptr ptr ext;
```

```
/*
 *   this stores the page information
 */

dcl max_nex_pages fix init(30);

dcl   1 nex_page [1] based (linkage$np_ptr),

        2 contour_num fix,              /* 0 if free, else contour      */
        2 level_num fix,                /* points to level              */
        2 rid fix,                      /* unique request_id            */
        2 max_graphics fix,             /* number of graphics           */
        2 graphic_ptr [10],             /* save up to ten pictures      */
          3 disc char(3),                   /* which disc               */
          3 frame fix(31),                  /* what frame ?             */
          3 pic_name char(32) vary,     /* keyword or caption           */
        2 next fix,                     /* index to next logical page */
                                        /* 0 incomplete chain           */
                                        /* > 0, points to next page     */
                                        /* < 0, negative of first idx */
        2 bufr char (1920) vary;        /* stick the stored page here */

dcl linkage$np_ptr ptr ext;
```

Appendix B

Nexis Daemon Documentation

This is the documentation for the Nexis daemon which comprises the heart of the NewsPeek NEXIS/MagicSix interface.

The nexis daemon is composed of five subprograms which live in >dd>Nexis>source and are bound together in >dd>Nexis>Nexis. The top-level program is called Ndaemon.pl1, and is very simple indeed. On being run (or launched) the daemon first calls $Ninit which sets up the two daemon data bases (nexsuper and nexinit, see Appendix C), and initializes the structures within. Then a (deferred) sleep condition "xmit_sig" is set to wake only after control passes to a catnap statement in the top level, and when the two values 'xhead' and 'xtail', which point ot the top and bottom respectively of the xbuf queue, are not equal, signifying that there is something in the queue to transmit. Next, several condition handlers are passed over, setting them up, so that the daemon can gracefully handle mail or other messages sent to it, and can properly clean up on logout.

At this point the daemon goes to sleep. In general, at this point one would signal "nio_make" on the daemon which would cause a Nexis stream to be initialized (using $make_nio) with its own wakeup handler watching it for

changes in the input buffer. To kill a line, such as at the end of a session, $free_nio is used.

The two last routines, $proc_recv and $proc_xmit are called when there is something in the input or output buffers respectively. $proc_recv strips off MDC characters SOM and ETX along with the two final LRC characters. [note: all transmission in either direction with MDC end with the LRC, or Longitudinal Redundancy Check, characters. What they amount to is a runnung exclusive or of all characters in the transmission, and they give a fairly good check on the accuracy of a transmission.] On receipt of the ETX, the condition "bufr_rdy" is signalled on the querying user, unless the user has not displayed interest in the page (see below), and has previously called nex$dont_bother.

$proc_xmit gets a "best choice" free line and transmit_id number and sends the bottom request to MDC. If the user displays interest in the request, either by including a page number in the request or through calling nex$stp, the daemon will also allocate a page in nexinput.

Appendix C

Notes on nex - the low level user interface to NEXIS.

=> overview:

nex is a rather largish binding of all sorts of useful entries to a user of the Nexis / MagicSix information / communication NewsPeek interface. It contains two basic types of entries -- those which involve direct communication to MDC (generally through a certain amount of buffering provided by the Nexis daemon), and those which simply make inquiries (or requests) concerning the status of the daemon's state, such as asking that a buffered input page from MDC be written into the users page_buffer area.

nex routines touch three of the four major data bases which are used in the newspeek system. By touching nexsuper, nex requests can login or logout a user, can allocate and free MDC id numbers, and can place information to be transmitted to MDC in xbuf. nex$save_page copies an input page from nexinput to nex_page, the user's page buffer area.

The entries $login, $logout, $get_xid (that's get_transmit_id), $free_xid, $what_rid, $save_page, and $kill_page communicate information between the user and the NEXIS daemon and do not involve MDC (i. e. the NEXIS system). Most of the other entries correspond to commands that can be given at a NEXIS terminal. (For an explanation of MDC NEXIS and its commands, see the NEXIS manual.)

nex routines cause the appropriate command to be sent to MDC and may also cause certain other things to happen in order to maintain a consistency within the NEXIS daemon (e.g. nex$abort). All logging in and out with MDC will be handled by the NEXIS daemon, although the user must log in and out from the daemon. The user must get at least one NEXIS_id (although one per active contour is reasonable) from nex$get_xid. A valid NEXIS_id must be passed with each request to MDC. User's should relinquish a NEXIS_id (use nex$free_xid) when they no longer require it (e.g. when deactivating the current contour or preparing to logout from the NEXIS daemon). In any case nex$logout will free any of the user's NEXIS_ids not already freed by the user.

```
/* ---------------- */
```

=> notes on implementation:

nex has a fairly smart command parser, in that it can accept integer arguments in character form (as from a command line) and it will convert them to integer form and check for bounds errors before executing the command. String inputs (for transmission to MDC) can be char(anything)varying, but nex will signal an error and not process the command if it is longer than 128 characters, or contains non-printing characters.

=> argument conventions:

All nex commands take a user_name as the first argument, and most (transmitting) commands require a valid xid (transmit_id) as the second. All nex commands return a fixed(15) number, which, if negative, indicates an error and signals the user that the last command was not processed. If an error is indicated, any side-effects caused (perpetrated?) by the command previous to the error's detection will be undone. In general, however, the returned number will be an rid. Exceptions are $login, $logout, etc, which simple return a fix(15) code = 0 if all goes well, and $get_xid which returns (what else?) a xid.

```
/* declarations */

dcl   uname    char(8) vary,   /* user name */
      xid      fix,            /* transmit id (0 <= xid <= 255)    */
      rid      fix,            /* request id  (1 <= rid <= 100)    */
      code     fix;            /* NOT fix(31). Generally the routines
                                  return the xid, which basically IS the
                                  code. Or the rid. If the returned value
                                  is negative, an error has occured.  */

/* ---------------- */
```

=> simple usage:

A generic user session with MagicSix/Nexis would begin with a call to nex$login, which takes the user_name and returns a negative code only if the user is already logged in, or the system is full. Then, a call to nex$get_xid will

get a free (if any) id_number to use in calls to the various transmit functions.

note: The user should store the xid (transmit_id) in a safe place (such as in nex_user.contour[ccontour].id_number_used) for all requests made in this line of inquiry (which I call a contour) should be accompanied by this number. The system reallocates old id numbers by time since last used, so there is some real possibility that, if the system has not been too heavily used, your id number may still be yours the next day, which will significantly reduce search resumption time.

note two: The xid should not be confused with the rid (request_id) which is returned from a call to nex$what_rid and other queueing transmit functions. This call is made after the condition bufr_rdy is signalled upon the user's process, signifying that a complete input_page is ready to be read. The request_id is a temporally unique number which describes (physically) the xbuf location used for some pending transmission to MDC, and should be, again, carefully saved by the user if she wishes to read the returned output from the command. It is returned by all nex$foo routines which queue on nexsuper.xbuf, such as nex$transmit and nex$chg_lib.

Before the user makes her first request to MDC's NEXIS, she should set up a bufr_rdy condition handler which is guaranteed to call either nex$save_page or

nex$kill_page, after, of course, calling nex$what_rid. This is required, for as input pages are buffered by the daemon in a single segment, there is only room for about 32 of them, and the daemon will cease to transmit requests until there is room for the returned output. Both nex$save_page and $kill_page free the input buffer for re_use.

When the user is finished with her session, a call to nex$logout is executed. This cleans up the data base, and removes the user from the Nexis user-ring.

```
/* ---------------- */
```

=> terminology:

For the purposes of these notes the following definitions apply:

A 'contour' is a state of NewsPeek and, by implication, of the MDC NEXIS system. To NEXIS, each contour is equivalent to a single user at a NEXIS terminal. As a coherent area of activity, a contour can be said to be a 'personality' of the user.

An 'active' contour is one for which the user has obtained a NEXIS_id and, hence, can issue requests to MDC NEXIS.

The 'current' contour is the one on which the user is currently working (e.g. issuing a request).

A 'dormant' contour is one which is not 'active'.

A 'request' is a request to the MDC NEXIS system (i.e. NOT call to nex$login, $what_rid, etc.).

'Current requests' are those of which the NEXIS daemon is cuurently aware. A 'current request' (uniquely identified by it's rid) is 'current' between the time nex$foo_request is called and the time the user calls $save_page or $kill_page. Being 'current' in NO WAY implies that either the NEXIS daemon or MDC is currently *working* on said request.

Bibliography

Milton's Areopagitica

Smith, Antony,
The Geopolitics of Information,
-- How Western Culture Dominates the World,
c1980 by Oxford University Press, New York

Wicklein, John,
Electronic Nightmare,
-- The New Communications and Freedom,
c1979 by The Viking Press, New York