NLS File System (Revised)

This version of the NLS File System document will appear as an appendix in the 1975 final report. The principal difference between it and earlier versions is its discussion of elements of the new property list based NLS file system in operation experimentally since January 1974. Programmers' attention is directed to the sections on primitives for dealing with the property entities (lower level procedures need not and probably should not be used) and to the sections which describe the ring and data block headers which are different from their earlier versions. Other changes are minor.

NLS File System (Revised)

Introduction 1

NLS operates on a heirarchical, random file system with several unique features evolved over the years that make possible the efficient online interaction used by the ARC community. Having information stored within separate structure and data blocks aids in rapid movement within and between NLS files; a "partial copy" locking mechanism provides security against attempted modification of a file by more than one user at the same time and provides a high degree of backup security against system failure or user error. This appendix includes a technical description of the file system as well as a discussion of motivating factors leading to its implementation. The design of the file system provides room for further extensions, some of which are also examined. 1a

Discussion of the heirarchical structure of NLS files at a user level, as well as a description of the user commands that permit movement through the files, may be found in [1]. 1b

This appendix is a revision of an earlier document which described the NLS file system as of July, 1974 and is current to January 1976. Discussed here are the most recent additions to the NLS file system, including property lists and inferior trees, which are currently used in the new graphics subsystem and offer great potential for the creation of new user entities. 1c

General Considerations Leading to the Current Design 2

The format and structure of NLS files were determined by certain design considerations: 2a

It is desirable to have virtually no limit on the size of a file. This means it is not practical to have an entire file in core when viewing or editing it. 2a1

The time required for most operations on a file should be independent of the file length. That is, small operations on a large file should take roughly the same time as the same operations on a small file. The user and the system should not be penalized for large files. 2a2

In executing a single editing function, there may be a large number of structural operations. 2a3

A random file structure satisfies these considerations. Each file is divided into logical blocks that may be accessed in random order. 2b

An early version of the file system was implemented on the XDS-940. Minor changes in the logical structure of the file system were made in the conversion of the system from the XDS-940 to the PDP-10 for two reasons: 2c

1) The current ARC programming language, L10, is more powerful than the several languages it replaces, MOL and the SPLs. L10 permits special purpose constructions anywhere in its code. It is a higher level language and provides greater compiler optimization. 2c1

2) An effort has been made to further modularize the functions within the system to ease development by a team of programmers. 2c2

In Winter 1975 extensions to the file system were made introducing property lists as data elements at each structural node. The first use of this capability was in the recently developed graphics subsystem. Further discussion of these changes may be found below. 2d

Reliability and the NLS File System 3

The reliability and security of file data both against system crashes and in face of the possibility of attempted simultaneous modification by more than one user were central goals in the design of the NLS file system. An attempt was made to minimize the amount of work which would be lost due to both hardware and operating system difficulties. 3a

Unlike the sequential file systems of some editors which require copying large sections of a file whenever an edit is made, NLS modifies copies of pages in which structural or data changes are made: all data in the original file is secure and a minimum of unaffected data is copied. Still other editors maintain recent changes in a dynamic buffer which may not be incorporated into the file in the event of a system crash; in NLS, barring a major hardware collapse, all changes other than those specified by the command being processed are present in the copied pages. Again, the original file is untouched. 3b

Other techniques to assure high reliability have been used such as organizing the code and sequence of operations in a way to minimize time windows of high vulnerability. 3c

An important problem in an online team environment such as that at ARC involves group collaboration on the same data files. The current file system permits multiple readers and a single writer to a file. The person obtaining write access to a file locks it in a manner described below; no other user is then permitted to write on the file, though they may read the original material. Readers without write access do not see the changes of the user currently editing the file until the file is explicitly "updated," causing the incorporation of edits and the unlocking of the file. Thus there can be no conflict between the edits of more than one writer. 3d

Details on the partial copy locking mechanism which implements these features of the NLS file system are discussed below in section (XXX). 3d1

Recent Extensions to the Nls File System 4

ARC recently extended the NLS file system to include a list of data blocks (a property list) rather than the single textual data block which existed before. These property lists are now associated with NLS structural nodes in the same manner that the single data block had been associated before. There is no restriction on the types of data nodes: for instance, graphic or numerical information may be possible as well combinations of data types within a single node. Additionally, data nodes may themselves have structure in the form of "inferior trees". The extended file system is upwardly compatable with the older file system: old files are still useable on the new file system without conversion. 4a

Short Technical Overview 5

This section gives a brief overview of the implementation of NLS files. For more detail see section (XXX). 5a

Block Header and Types of Blocks 5b

An NLS file is made up of a file header block, and up to a fixed number (currently 465) of 512-word (=equals one TENEX page) structure blocks (up to 95), and data blocks (up to 370). 5b1

There are several types of blocks, each with its own structure: 5b2

File header block--always page 0: contains general information about the file. 5b2a

Structure (ring) blocks--contain ring elements that implement the NLS structure: there currently may be a maximum of 95 of these blocks, each containing 102 five-word ring elements. They may appear in file pages 6 through 100. 5b2b

Data blocks--contain the data (in linked property lists associated with structural nodes) of NLS statements: each data block is composed of individual data elements made up of a five-word header followed by text strings or other data. There currently may be a maximum of 370 data blocks. They may appear in file pages 101 through 471. 5b2c

Miscellaneous blocks--not used in the current implementation. 5b2d

File Header Block 5c

In each file there is a header block that contains general information about that particular file. The header block remains in memory while the file is in use. 5c1

The file header is read into core by the procedure (nls, ioexec, rdhdr). This procedure checks for the validity of certain keywords. If the file is locked and has a partial copy, the header is read in from the partial copy. If the partial copy header block is invalid in the key spots, the file is unlocked and the header read in from the original file. If that is bad, the file may be initialized. 5c2

RDHDR sets the value of the FILENO-th element in the table FILEHEAD. FILENO is the NLS file number of the file. (It is an index into the file status table that provides, among other things, a correlation between JFNs for the original and partial copy and the single NLS file number). 5c3

Procedures in (nls, filmnp,) are responsible for reading, manipulating, creating, garbage collecting, and storing into ring blocks and ring elements within those blocks, and data blocks and statement data blocks within them. 5c4

Structure Blocks -- Ring Elements 5d

Conceptually an NLS file is a tree. Each node has a pointer to its first subnode and a pointer to its successor. If it has no subnode, the sub-pointer points to the node itself. If the node has no successor, the successor pointer points to the node's parent. (These conventions are used to aid in providing a set of primitives for rapidly moving around in NLS files.) Each node is currently represented by a ring element. These ring elements point in turn to the first data block in the node's property list. 5d1

Structure blocks contain five-word ring elements with a free list connecting those not in use. 5d2

Data Block -- Property Lists and (Textual) Statement Data Blocks 5e

Data blocks are composed of variable sized elements called (Textual) Statement Data Blocks (SDBs) that contain the text of NLS statements and other types of data elements. Other data element types are currently used in the NLS graphics system though the number of available types and uses may be easily extended. All data elements have a five word header followed by data appropriate to the element type. Each SDB has this five-word header with node related information followed by the text made up of 7-bit ASCII characters packed five to a word. New data elements are allocated in the free space at the end of a data block page. Data elements no longer in use (because of editing changes) are marked for garbage collection when the free space is exhausted. 5e1

Data elements associated with node are linked together in a property list. This property list may in turn have a structured inferior tree associated with it; the nodes on the inferior tree structure of a data element may also have associated property lists. This feature may prove to be useful in the creation of a comment entity in NLS for comments associated with a particular NLS statement. 5e2

Statement (or String) Identifiers (STIDS) and Text Pointers 5f

A statement identifier (STID) is a data structure used within NLS to identify NLS statements (structural nodes) or strings. 5f1

If the string is in an NLS statement, the STID contains a file identifier field (STFILE) and a ring element identifier (STPSID). 5f1a

The presence of a file identifier within the STID permit all editing functions to be carried out between files. 5f1b

Procedures in (nls, filmnp,) and (nls,strmnp,) permit traversal through the ring structure of a file given an STID. See, for example, (nls, filmnp, getsuc), which gets the STID of the successor of a statement; see also (nls, filmnp, getsdb), which returns the STDB for the statement whose STID is provided as an argument. (An STDB has, like an STID, a file number field and a pointer to the textual property block in the property list, a STPSDB). Additional primitives are available for other data properties. 5f2

Text pointers are two-word data structures used with the string analysis and construction features of L10. They consist of an STID and a character count. 5f3

Locking Mechanism -- Partial Copies 5g

The NLS file system under TENEX provides a locking mechanism that protects against inadvertent overwrite when several people are working on the same file. Once a user starts modifying a file, it is "locked" by him against changes by other users until he deems his changes consistent and complete and issues one of the commands: Update File, Update File Compact, or Delete Modifications, which unlock the file. A user can leave a file locked indefinitely--this protection is not limited to one console session. 5g1

When a file is locked (is being modified), the user who has modification rights sees all of the changes that he is making. However, others who read the file will see it in its original, unaltered state. If they try to modify it, they will be told that it is locked by a particular user. Thus the users can negotiate for modification rights to the file. 5g1a

This feature is implemented through the use of flags in the status table in the File Header and through the partial copy mechanism. 5g2

All modifications to a file are contained in a partial copy file. These include modified ring elements and data blocks. 5g2a

Any file page that is to be and that is not in the partial copy (discovered through a write pseudo-interrupt) is copied into the partial copy. All editing takes place there. The TENEX user-settable word in the FDB (TENEX file data block) for the original file contains locking information. 5g2b

The NLS Update file command merely replaces those structure and data pages in the original file that have been superseded by those in the partial copy, unlocks the file, and deletes the partial copy. For Update file old, this is done in the original file; for Update to new version, the pages are mapped to a new file from the original or partial copy where necessary. The Update file compact command garbage collects unused space; the update file command does not. 5g3

Core Management of File Space 5h

When space is needed for more data, the following steps are taken, in order, until enough is found to satisfy the request (See (nls, filmnp, nwrngb), (nls, filmnp, newsdb), and related routines): 5h1

1) Core-resident pages are checked for sufficient free space. 5h1a

2) Other pages are checked for free space. If one has sufficient space, it is brought in. 5h1b

3) If garbage collection on any page in the file will yield a page with sufficient free space, then the page

that will give the most free space is brought into core and garbage-collected; otherwise a new page is created. 5h1c

Detailed Technical Discussion 6

Note on Fields in NLS Records and Other L10 Language Features 6a

Several parts of this section are taken directly from record declarations in the code of the NLS system written in the L10 programming language. 6a1

Record declarations in the L10 language serve as templates on data structures declared in the system. Byte pointer instructions are dropped out by the compiler permitting access to specified parts of the array. Multiword records are filled from the lowest to the highest address of the array. Within words, bits are allocated from the first bit on the right. If several fields fail to fill a 36-bit word and the next field definition would go over the remaining bits in the word, the field is allocated in the next word available. 6a2

Example: 6a2a

Bit 0 is the leftmost bit in the word; bit 35 the rightmost. Suppose there is a record declaration of the form: 6a2a1

```
(newrecord) RECORD % A two word record %                6a2a1a
   field1[10]. %bits 26 through 35 (rightmost) of
   first word%                                          6a2a1a1
   field2[25], %bits 1 through 25 of first word %       6a2a1a2
   field3[15]; %bits 21 through 35 of second word
   (field would not fit in remainder of first
   word%                                                6a2a1a3
DECLARE array[2];                                       6a2a1b
```

There may be code within a program of the form: 6a2a2

```
variable _ array.field2;                                6a2a2a
array.field3 _ 20;                                      6a2a2b
```

In L10, false is zero and true is nonzero. 6a3

See the L10 manual for further information. 6a4

Block Header and Types of Blocks 6b

An NLS file is made up of a file header block page and up to a fixed number (currently 465) of 512-word (= one TENEX page) structure block pages (up to 95) and data block pages (up to 370). 6b1

Each page has a two-word header telling the type and giving the file page number and an index into a core status table. The record declaration from (nls, utility,) follows: 6b2

```
   (fileblockheader) RECORD %fbhdl = 2 is length%             6b2a
      fbnull[36],    %unused%                                  6b2a1
      fbind[9],     %status table index%                       6b2a2
      fbpnum[9],    %page number in file of this block%        6b2a3
      fbtype[5];    %type of this block (types declared in
      (nls, const,))                                           6b2a4
         hdtyp  = 0   = header                                 6b2a4a
         sdbtyp = 1   = data                                   6b2a4b
         rngtyp = 2   = ring                                   6b2a4c
         jnktyp = 3   = misc (such as keyword, viewchange,
         etc.  Not currently used.)%                           6b2a4d
```

```
PAGE HEADER BLOCK
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
X free
X
X 36
X
X----------------------------------------------------------
X
X free                              * Type * Page   * Status
X
X                                   *      * Number * Table
X
X 13                                * 5    * 9      * 9
X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
```
6b3

There are several types of block pages, each with its own structure. 6b4

File header block pages--always page 0: contains general information about the file. 6b4a

Structure (ring) block pages--contain ring elements that implement the NLS structure: there currently may be a maximum of 95 of these blocks, each containing 102 five-word ring elements. They may appear in file pages 6 through 100. 6b4b

Data block pages--contain the data properties of NLS statements: each data page has properties with five-word headers followed by data (text, graphics instructions, etc.). There currently may be a maximum of 370 data pages. They may appear in file pages 101 through 471. 6b4c

Miscellaneous blocks--not used in the current implementation. 6b4d

File Header Page 6c

In each file, there is a header block page that contains general information about that particular file. The header block remains in memory while the file is in use. 6c1

FILE HEADER CONTENTS (taken from (nls, data,)): 6c1a

```
DECLARE EXTERNAL                                          6c1a1
%...file header...%                                       6c1a2
   % DONT CHANGE THE ITEMS IN THE HEADER %                 6c1a2a
   filhed[5],                                             6c1a2b
      % these extra words may be taken for additions
      to header%                                          6c1a2b1
   fcredt,   % file creation date--TENEX gtad jsys
   internal format %                                      6c1a2c
   nlsvwd = 1,                                            6c1a2d
      % nls version word; changed when NLS file
      structure changes %                                 6c1a2d1
   sidcnt,   %count for generating SID's%                 6c1a2e
      % An SID (statement identifier) should not be
      confused with PSIDs (see below). The SID is
      uniquely generated for each statement in a file
      and is not reused if a statement is deleted;
      it is unchanged if a statement is moved. It
      may be used by a user for accessing particular
      statements in a file without worrying about
      changes because of additions or deletions (as
      is the case with statement numbers). The
      sidcnt field in the header is increased by one
      as statements are created. The value is stored
      in the RSID field of the ring element (see
      description below). %                               6c1a2e1
   finit,                                                 6c1a2f
      % initials of user who made the last write (by
      updating or outputting the file)--see DATA
```

BLOCK description below for explanation of initials % 6c1a2f1
funo, % user number (file owner) % 6c1a2g
lwtim, % last write time--TENEX internal JSYS gtad format % 6c1a2h
namdl1, % left name delimiter default character % 6c1a2i
namdl2, % right name delimiter default character % 6c1a2j
rngl, 6c1a2k
% upper bound on ring (structure) file blocks used % 6c1a2k1
dtbl, % upper bound on data file blocks used % 6c1a2l
rfbs[6], 6c1a2m
% start of random file block status tables (see description below) % 6c1a2m1
rngst[95], % ring block status table % 6c1a2n
dtbst[370], % data block status table % 6c1a2o
mkrtxn = 20, % marker table maximum length % 6c1a2p
mkrtbl, % marker table current length % 6c1a2q
mkrtb[20], % marker table % 6c1a2r
% Markers provide an alternative form of NLS addressing; see NLS Users Guide for description % 6c1a2r1
filhde; %end of the file header% 6c1a2s

Notes on File Header 6c1b

The file header is read into core by the procedure (nls, ioexec, rdhdr). This procedure checks for the validity of certain keywords. If the file is locked and has a partial copy (the file that includes current modifications--see below), the header is read in from the partial copy. If the partial copy header block is invalid in the key spots, the file is unlocked and the header read in from the original file. If that is bad, the file may be initialized. RDHDR sets the value of filehead[fileno] where fileno is the NLS file number of the file (an index into the file status table which provides, among other things, a correlation between JFNs for the original and partial copy and the single NLS file number; see description of the file status table below.) 6c1b1

(nls, ioexec, setfil) initializes a file header. 6c1b2

It should be noted that fields within a file header are accessed by full word indexing rather than by record pointers for speed. Thus we have the following typical code (from (nls, utilty, esc)) that reads the default name delimiters from an NLS file header: 6clb3

```
   .                                                   6clb3a
   .                                                   6clb3b
   .                                                   6clb3c
   ELSE IF rplsid.stpsid = origin THEN                 6clb3d
      BEGIN %use standard delimiters for that file%    6clb3dl
      fhdloc _ filehead[rplsid.stfile] - $filhed;      6clb3d2
      dlleft _ [fhdloc + $namdl1];                     6clb3d3
      dlrght _ [fhdloc + $namdl2];                     6clb3d4
      END                                              6clb3d5
   .                                                   6clb3e
   .                                                   6clb3f
   .                                                   6clb3g
```

Also, code from (nls, ioexec, rdhdr) that gets the address of the word in core that contains the nls version word for the file whose header has been read in order to check its validity: 6clb4

```
   .                                                   6clb4a
   .                                                   6clb4b
   &vwd _ (header _ filhdr(fileno)) - $filhed +
   $nlsvwd;                                            6clb4c
   filehead[fileno] _ header;                          6clb4d
   .                                                   6clb4e
   .                                                   6clb4f
```

The file header is initialized by (nls, ioexec, rdhdr) which fills up contiguous words declared in (nls, data,) and then moves the contents of those words to page zero of the file. 6clb5

```
FILE HEADER BLOCK  (FULL WORDS)
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X free[5]                    X X Max structure pages
X
X---------------------------X
X----------------------------X
X Creation data              X X Max data pages
X
X---------------------------X
X----------------------------X
X Version Number (=1)        X X Start of block tables[6]
X
X---------------------------X
X----------------------------X
X SID Count                  X X Ring block status
table[95]X
X---------------------------X
X----------------------------X
X Initials last write        X X Data blck status
table[370]X
X---------------------------X
X----------------------------X
X File Owner                 X X Marker table size (=20)
X
X---------------------------X
X----------------------------X
X Time last write            X X Marker table length
X
X---------------------------X
X----------------------------X
X Left name delimiter        X X Marker table[20]
X
X---------------------------X
X----------------------------X
X Right name delimiter       X X
X
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

6c1c

Procedures in (nls, filmnp,) are responsible for reading, manipulating, creating, garbage collecting, and storing into ring blocks and ring elements within those blocks, and data blocks and statement data blocks within them. 6c2

Random File Block Status Table Entries in File Header 6d

The random file block status tables appear in the file header. There is one word per ring block or data block page. Each entry contains the following: record declaration and comments from (nls, utilty,). 6d1

(rfstr) RECORD % Random file block status record. (The entry will be equal to 0 if the page (i.e., block) in the file is unallocated. Otherwise, the entry will be an instance of the following record.)% 6d1a

```
   rfexis[1],   %true  (i.e., nonzero) if the block
   exists in the file%                                  6d1a1
   rfpart[1],   %true if block comes from partial copy%  6d1a2
      %Whether page has been modified by a user.        6d1a2a
         (rfpart will be true in that case.)%          6d1a2a1
   rfnull[2],   %unused%                                6d1a3
   rfused[10],  %used word count for the block%         6d1a4
      %Current used word count (may be used to calculate
      post-garbage collection free space count.)%      6d1a4a
   rffree[10],  %free pointer for the block%            6d1a5
      %Free space count (for data block)               6d1a5a
         Pregarbage collection free space count.       6d1a5a1
      Free list pointer (for ring block)%              6d1a5b
   rfcore[9];   %0 then not in core, else page index%   6d1a6
```

```
BLOCK STATUS TABLE ENTRIES (STRUCTURE OR DATA)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX
X    * Page index else * Free    * Used    *    *
Part-*ExistX
Xfree* =0 if not in    * pointer * word    *free* ial   *
?  X
X    * core            *   or    * count *    * copy?*
X
X    *                 * count   *         *    *      *
X
X 3  * 9               * 10      * 10    * 2  * 1    * 1
X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX
```

6d1b

Notes on Random File Block Status Tables 6d2

The table RFBS in the file header is broken into two sections, each of which contains a collection of records of the above type. The first section includes RNGM entries from RFBS[RNGBAS] up to and including RFBS[RNGBAS+RNGM-1] and contains information about the ring block pages in the file. (RNGBAS is currently 6 and is the first page in a file that may be a ring page;

RNGM is currently 95 and is the maximum number of ring block pages permitted.) 6d2a

The second section includes DTBM entries from RFBS[DTBBAS] up to and including RFBS[DTBBAS+DTBM-1] and contains information about the data block pages in the file. (DTBBAS is currently 101 and is the first page in a file that may be a data block page; DTBM is currently 370 and is the maximum number of data block pages permitted.) The entry RFBS[RNGBAS+i] may also be referenced as RNGST[i]; likewise RFBS[DTBBAS+i] may be referenced as DTBST[i]. The index in RFBS is the actual page number of a data page in the file. 6d2b

A pointer to a data element or property (PSDB) consists of a nine-bit data page number in the range [0,DTBM) and a nine-bit displacement from the start of the page. The variable DTBL is maintained in each file header as the current upper bound on allocated data pages for that file. This is used to limit the search for a location for a new data element. The variable DBLST contains the index of the block from which a property was last allocated or freed. 6d2c

A pointer to ring element (PSID) consists of a nine-bit ring page number in the range [0,RNGM) and a nine-bit displacement from the start of the page. The variable RNGL is maintained in each file header as the current upper bound on allocated ring pages for that file. This is used to limit the search for a location for a new ring block. The variable RNGST contains the index of the page from which a ring was last allocated or freed. 6d2d

Structure Blocks -- Ring Elements 6e

These blocks contain five-word ring elements with a free list connecting those not in use. 6e1

```
(ring) RECORD %ringl is length% % from (nls, utility,) %      6e2
   rsub[18],     %psid of sub of this statment%               6e2a
      % A pointer to the first substatement of this
      statement %                                             6e2a1
   rsuc[18],     %psid of suc of this statement%              6e2b
      % A pointer to the successor of this statement (to
      the parent if no successor) %                           6e2b1
   rsdb[18],     %psdb of first property for this statement%  6e2c
      % Pointer to the first property in the property list
      of data blocks for this statement. %                    6e2c1
   rinstl[7],  %DEX interpolation string-- scratch space%     6e2d
```

```
      % Information in scratch fields may be reset and used
      by other subsystems such as DEX.  No other assumption
      concerning their contents shold be made. %              6e2d1
   rinst2[7],  %DEX interpolation string-- scratch space%      6e2e
   rdummy[1],   %DEX dummy flag-- scratch space%               6e2f
   repet[3],    %DEX repetition-- scratch space%               6e2g
   rhf[1],      %head flag. true (= 1) if this is head of
   plex%                                                       6e2h
   rtf[1],      %tail flag, true if tail of plex%              6e2i
   rnamef[1],   %name flag, true if statement has a name%      6e2j
   rtorgin[1],  %inferior tree origin flag, true if origin%    6e2k
   rnull[1],    %unused%                                       6e2l
   rnameh[30],  %name hash for this statement%                 6e2m
      % hash algorithm may be found in (nls, utilty, hash)
      %                                                       6e2m1
   rsid[30];    %statement identifier%                         6e2n
      % See SIDCNT description in file header above. %        6e2n1
   %although only need four words, use five so that have
   room to grow%                                               6e2o
```

```
RING ELEMENT
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
X PSID of Successor           * PSID of Substatement (Down)
X
X 18                          * 18
X
X-------------------------------------------------------
X
X Scratch space used by DEX * PSDB (pointer to data block)
X
X 18                          * 18
X
X-------------------------------------------------------
X
Xfree* Name Hash                    *free*org
*name*tail*headX
X 1  * 30                           * 1  * 1  * 1  * 1  * 1
X
X-------------------------------------------------------
X
X free * Statement Identifier
X
X 6    * 30
X
X-------------------------------------------------------
X
X free
X
```

```
X 36
X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
```
6e3

PSIDs and PSDBs are pointers to other ring or data blocks in a file. They have two nine-bit fields: one (stblk) is a page index; the other (stwc) is a word displacement within that page. Procedures in (nls, filmnp,) permit the traversal of a file's structure. 6e4

Given an STID (see below), one may use the primitive procedures in (nls, filmnp,)--e.g., (nls, filmnp, getsuc)--or the more elaborate procedures in that file--e.g., (nls, filmnp, getnxt)--to move around to related ring elements and retrieve or change (display or edit) relevant data. 6e5

There are two "fixed" values for PSIDs for special statements: 6e5a

The PSID of the origin statement is always 2. 6e5a1

The entire STID (and hence PSID) of the end of a file is endfil (=-1), which does not correspond to any real statement in the file, but which is returned by the "get" procedures in filmnp to indicate the end has been reached or an error has been found. 6e5a2

Some other conventions implemented in the file structure make possible special features in NLS: 6e5b

The successor of a statement with no real successor is its "parent." 6e5b1

The substatement of a statement with no sub is itself. 6e5b2

The origin is at a unique level; thus statement 1 is the sub of the origin. 6e5b3

Data Block -- Property Data and Statement Data Blocks 6f

Property lists are made up of linked lists of property data blocks. An example of a property is the Statement Data Block (SDB) which contains the text of an NLS statement. 6f1

Each property has a five-word general header with the following information. There then follows data appropriate to the particular property type. For example, (Textual) Statement Data Blocks (SDBs) contain the text in NLS statements; this text follows the property header and is composed of seven-bit ASCII characters packed five to a word. in a variable length block. New properties are allocated in the free space at the end of a data page. Properties no longer in use (because of editing changes) are marked for garbage collection when the free space is exhausted. 6f2

```
(sdbhead) RECORD %sdbhdl is length% % from (nls, utility,) %   6f3
   sgarb[1],     %true (non-zero) if this sdb is garbage,
   i.e., no onger used%                                         6f3a
   slength[9],  %number of words in this sdb%                   6f3b
   schars[11],  %number of characters in this statement%        6f3c
   slnmdl[7],   %left name delimiter for statement%             6f3d
   srnmdl[7],   %right name delimiter for statement%            6f3e
   spsid[18],   %psid of the statement for this sdb%            6f3f
      %Pointer to ring element.%                                6f3f1
   sname[11],   %position of character after name%              6f3g
      %  This is 1 for a statement with no name. Thus if
      the text of the statement were:                           6f3g1
         (author) The person who                                6f3g1a
      and the name delimiters were "(" and ")", the value
      of this field would be 9. %                               6f3g2
   stime[36],   %date and time when this SDB created%           6f3h
      % This is stored in the TENEX internal format; see
      the TENEX JSYS manual, gtad jsys %                        6f3h1
   sinit[21],   %initials of user who created this sdb%         6f3i
   sptype[15],  %property type of this data block%              6f3j
      %                                                         6f3j1
      txttyp    = text data block (SDB)                         6f3j2
      dhtyp     = graphics diagram header                       6f3j3
      chtyp     = graphics cell header                          6f3j4
      gtftyp    = graphics text format                          6f3j5
      lwtyp     = graphics line work                            6f3j6
      %                                                         6f3j7
   spsdb[18],   %PSDB of the next property data block;
   0=tail%                                                      6f3k
   sitpsid[18]; %PSID to head of inferior tree if any%          6f3l
   %sgarb and slength must be in the first word of the
   header for newsdb%                                           6f3m
```

```
DATA BLOCK HEADER
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
X          *Right name *Left name  *Character*Block
*Garb-X
```

```
X free      *delimiter  *delimiter  *count    *length*age?
X
X 1         * 7         * 7         * 11      * 9    * 1
X
X-----------------------------------------------------------
X
X free * Position of char   * PSID pointer to ring element
X
X 7    * 11   after name    * 18   for this statement
X
X-----------------------------------------------------------
X
X Creation time
X
X 36
X
X-----------------------------------------------------------
X
X Property    * Authors initials
X
X 15  type    * 21
X
X-----------------------------------------------------------
X
X PSID of inferior tree      * PSDB of the next property
X
X 18                         * 18
X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
```

6f4

```
STATEMENT DATA BLOCK (SDB'S) Text type block
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
X Data block header
X
X 5 full words
X
X-----------------------------------------------------------
X
X Text
X
X Block length - 5 words of 5 characters each
X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
```

6f5

Statement (or String) Identifiers (STIDS) and Text Pointers 6g

A statement identifier (STID) is a data structure used within NLS to identify NLS statements (structural nodes) or strings. 6g1

If the string is in an NLS statement, the STID contains a file identifier field (STFILE) and a ring element identifier (STPSID). (See PSID description above under ring elements.) 6g1a

The presence of a file identifier within the STID permit all editing functions to be carried out between files. 6g1b

Procedures in (nls, filmnp,) and (nls,strmnp,) permit traversal through the ring structure of a file given an STID. See, for example, (nls, filmnp, getsuc), which gets the STID of the successor of a statement; see also (nls, filmnp, getsdb), which returns the STDB for the statement whose STID is provided as an argument. (An STDB has, like an STID, a file number field and a pointer to the textual property block in the property list, a STPSDB). Additional primitives are available for other data properties. 6g2

Text pointers are used with the string analysis and construction features of L10. They consist of an STID and a character count. 6g3

Other Relevant Arrays 6h

The following arrays are used in system core and file management. They are described here to facilitate the study of the NLS file-handling code. 6h1

Filehead 6h2

An array of pointers (each contained in a single word) to the file headers of files currently in use is FILEHEAD. At present, up to 25 files (and their partial copies, if any) may be open simultaneously. 6h2a

CORPST (Core Page Status Table) and CRPGAD (Core Page Address Table) 6h3

The array CORPST provides the correspondence between the 100 (octal) pages in core reserved for file pages and user program buffer and the pages in files that are currently loaded into core. (This is really a maximum of 100 octal since the user program buffer may be enlarged into this area; the maximum is given by RFPMAX - RFPMIN +1.) 6h3a

(corpgr) RECORD %one word. core page status record. gives status for a given core page for random files.% 6h3a1

ctfull[1], %true if the page is in use% 6h3a1a

ctfile[4], %file to which the page belongs; an NLS file number% 6h3a1b

ctpnum[9], %page number within the file% 6h3a1c

ctfroz[3]; %number of reasons why frozen (locked into core because of some current NLS system need-- editing is in progress on a statement, a statement is being displayed, etc) % 6h3a1d

The array CORPST is the core page status table and is made up of instances of the above record. (RFPMAX (dependent on the current user program buffer size) gives the number of core pages that may contain file pages. The core pages are located at positions indicated by the array CRPGAD (core page address). CORPST is indexed by numbers in the range (RFPMIN, RFPMAX). The elements in this array are actual addresses. The starting location of page k is given by crpgad[k]. RFPMIN is initialized to be 7; six pages are initially allocated for a user program buffer. See (nls, usrpgm, gpbsz) for the procedure that changes these limits. 6h3b

FILST (File Status Table) 6h4

An NLS file number provides an index into the FILST, the file status table. This 100-word array is made up of 25 four-word entries and contains the following information for files of interest that have NLS file numbers at any time (these may or may not at that time be open; they do, however, have JFNs.) The information comes from the record declaration in (nls, utilty,): 6h4a

(filstr) RECORD %File status table record. entry length = filstl = 4, max no. entries = filmax = 25% 6h4a1

flexis[1], %true: entry represents an existing file% 6h4a1a

flhead[9], %crgpad index of the file header% 6h4a1b

flbrws[1], %this file in browse mode% 6h4a1c

fllock[1], %This file was locked by another user when loaded% 6h4a1d

flpcread[1], %PC read only--write open failed (openpc)--see discussion of partial copies below% 6h4a1e

flaccm[8], %file access mask% 6h4a1f

% Used to tell whether or not the file may be written on by the current user. Used primarily for files such as those in the Journal that are read-only to most users. % 6h4alfl

```
fldirno[12],  %directory number for the original
file%                                                6h4alg
flpart[18],   %JFN for the partial copy%             6h4alh
flbpart[18],  %JFN for the browse partial copy%      6h4ali
florig[18],   %JFN for the original file%            6h4alj
flastr[18],   %address of the file name string%      6h4alk
flpcst[18],   %address of partial copy name string%  6h4all
flbpcst[18];  %address of browse partial copy name
string%                                              6h4alm
```

Primitives for Use with Basic NLS File Entities 6i

Introduction 6i1

The following primitives will be available for manipulation of basic file entities. While they make use of even more basic procedures, most programmers should have no reason for accessing lower level routines. These primitives and lower level procedures live in the file FILMNP. 6i1a

Property types must be assigned numbers by ARC. Code for management and portrayal of properties not generally available or useful for all NLS users will be managed and written by the prime users. The procedures listed below will provide access to property blocks and nodes in the files. 6i1b

The code which manages graphics file entities lives, currently, in the graphics subsystem. 6i1b1

Entity types 6i2

Primitives will be available to operate on the following file entity types: 6i2a

NODE -- a ring element and its associated data contained in a property list. 6i2a1

PROPERTY -- a data block and any associated inferior tree within the property list associated with a node. 6i2a2

INFERIOR TREE -- structure and data associated with a property block. 6i2a3

An example of the use of an inferior tree may be found in the graphics subsystem in which diagrams have structure reflected by the existence of this inferior tree. Another possible use could be for imposing the structure (NLS Plex-like in nature) of comments associated with a statement's text. Normal NLS structural procedures for examining structure and modifying it at the file level may be used at the inferior tree level as well. 6i2a3a

Note that while no direct primitives are provided for operating on property lists or portions of them, such primitives exist at lower levels. It is not felt that higher level primitives for such entities are necessary. The operations listed below follow the currently existing examples for text nodes in NLS files. 6i2b

Operands and procedures 6i3

READ -- Most read functions are dependent on the property type and are to be managed by formatters and other specific application code. Thus a set of "get" and "set" routines are available for examining and setting fields in the statement text nodes and similar procedures exist in the graphics subsystem. A general primitive to load particular property types into core is provided. Also, the usual procedures for moving around in structure will be available. 6i3a

lodprop (stid, proptype) -- 6i3a1

loads the indicated property block into core. Returns three items: first is FALSE if error, page number in core if success; second is address of block in core (which must be frozen if you want to do anything with it!), third is stdb of property block 6i3a1a

Note change: as originally written, lodprop also took "occurence" of property in list. we now will not permit more than one property of a particular type in a particular list. Multiple occurences may be handled by a structural inferior tree hanging off the property block. 6i3a2

CREATE -- Allocate space for entity and link the blocks into existing structure and/or data. 6i3b

crenod (stid, rlevcnt) -- 6i3b1

Gets a new ring element with no associated data blocks and links it into the structure at the location specified by stid and rlevcnt (a relative level count: < 0 is down, =0 is successor, > 0 is up by rlevcnt levels). Returns stid of new ring or 0 if error. 6i3b1a

creprop (stid, proptype, length, data) -- 6i3b2

Builds a data block of property type proptype which must be a valid type assigned (and declared) by ARC and links it into the property list associated with the stid in the proper order (determined in the procedure linkprop). If such a property already exists in the node, we have an error: it must first be deleted. Returns stdb of new block or 0 if error. length is the length of the data and data is a pointer to an array of length words in which the data is stored. 6i3b2a

creit (stid, proptype) -- 6i3b3

Creates the origin of an inferior tree and links it to the data block property specified by stid and proptype. Returns 0 if error or stid of origin of inferior tree. 6i3b3a

DELETE -- Unlink entity from other structure and data. Release space. 6i3c

delprop (stid, proptyp) -- 6i3c1

deletes the property block and any associated inferior tree structure for the block proptype block of the indicated node. Returns TRUE if successful, 0 if not. 6i3c1a

delit (stid, proptype) -- 6i3c2

deletes the inferior tree of the indicated property block. Unlinks it and releases space. Returns True if successful, 0 if not. 6i3c2a

Currently no primitive exists to directly delete a node, though the primitives remgrp and delgrp perform this function together. The x-routines which implement structural deletes call these file system primitives. 6i3c3

MOVE -- Unlink entity and relink it at new location in file. 6i3d

movprop (stid, proptyp, destid) -- 6i3d1

Moves the property indicated from node specifed by stid to node specified by destid. Accomplishes this by unlinking and relinking the block. If a property type of the type being moved exists at the destination, we have an error. Returns true if OK, 0 if error. 6i3d1a

movit (stid, proptype, destid) -- 6i3d2

moves the inferior tree associated with property block indicated by stid and proptype to the property block proptype associated with node destid. Returns true if OK, 0 if error 6i3d2a

X-routines currently exist to move and move filtered other strucural entities. 6i3d3

COPY -- Allocate space for new entity, copy old entity, and link the new entity into the file. 6i3e

copprop (stid, proptype, destid) -- 6i3e1

copies property block (and associated inferior tree if any) from block indicated by stid and proptype to a new block to be created on destid. Returns TRUE if OK. 0 if error 6i3e1a

citree (stid, proptyp, destid) -- 6i3e2

copies inferior tree of property block at node indicated by stid and proptype to the proptype block of destid. Returns TRUE if successful, 0 if error 6i3e2a

Primitives exist to do structural copies both in filtered and unfiltered modes. 6i3e3

REPLACE -- In keeping with the mode which exists for text statements, a replace primitive will not be provided for the inferior tree entity or the node entity. These functions may be accomplished using existing x-routines or primitives which delete nodes followed by a copy or create. 6i3f reprop (stid, proptype, length, data) -- 6i3f1

replaces the property block indicated by stid and proptyp 6i3f1a

with a block with data as indicated. If length is the same as 6i3f1b

the length of data in the existing property block, a short cut may 6i3f1c

be taken and the data overwrites the old data. If, however, 6i3f1d

the length is different, a new block is built and linked in. 6i3f1e

The inferior tree is not replaced in any case: it remains the 6i3f1f

same. The inferior tree's pointer to the "owning" property 6i3f1g

block is changed to point to the new block. Uses filesc if this is a text block. 6i3f1h

References 7

(17b2) Douglas C. Engelbart and Staff of ARC. Computer-Augmented Management-System Research and Development of Augmentation Facility [Final Report]. Augmentation Research Center, Stanford Research Institute, Menlo Park, California 94025. APR-70. (5139,) 7a