1973

Window: a formally-specified graphics-based text editor

Douglas Gerhardt
Carnegie Mellon University

David Lorge. Parnas

Follow this and additional works at: http://repository.cmu.edu/compsci
NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.
WINDOW: A FORMALLY-SPECIFIED GRAPHICS-BASED TEXT EDITOR

Douglas Gerhardt and D. L. Parnas

Computer Science Department
Carnegie-Mellon University

June, 1973

This work was supported by the National Science Foundation under Grant GJ 30127 and Grant GJ 37728 to Carnegie-Mellon University and also by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107), monitored by the Air Force Office of Scientific Research.
PREFACE

WINDOW is a formally-specified text editing program which exploits the graphics capabilities of CRT terminals [1]. Terminal screens are divided into several "windows". Each window displays text from any of a number of simultaneously open files. Through the notion of linked "pointers", operations upon one file may invoke corresponding operations upon other files.

D. L. Parnas designed and produced the formal specification for its kernel editing system. Douglas Gerhardt implemented the kernel according to the specification and developed a command interpreter.

This paper has three main sections covering the interface and structure of the kernel, the relationship between the kernel and the command interpreter, and our experiences with this application of formal specification.

Drafts of this paper were edited using the implementation of WINDOW described in [1].

---

INTRODUCTION

INTERFACE SPECIFICATION

A technique [2] has been described for specifying the interface between parts of software systems. In this paper, we have applied the same technique to the formal specification of a man-machine interface [3].

At first glance, such a specification appears to be a program written in a high-order language. It is not. A program describes a process by listing a sequence of actions to be performed by a lower level machine. A specification does not admit the existence of such a machine. All of the functions mentioned in a formal specification are available to the user. They are described, not by giving their implementations, but by enumerating their effects upon each other. The result is a "black box" description.


The black box description in this paper suggests a simple implementation because we abstract files as arrays, an unworkable implementation for most real situations. The actual implementation is more complex but the details of it are hidden. They are not necessary for making good use of the system.

The formal specification is inherently complete. As with most formal structures, however, a human being needs a description of the intended interpretation of the structure in order to comprehend it. Such commentary is not a part of the specification, nor is it complete.
Text editing often involves creating or modifying one file on the basis of information contained in other files. As a great deal of time and effort is spent shifting physical focus among the files, we realized that our work would be assisted by an editing program which would allow us to look at several files on one screen. We wanted the ability to divide the screen into "windows", each capable of displaying a section from any of a number of files. The division into windows and the assignment of windows to file sections would not be fixed. An additional useful property would be to have the center line of each window move through a file as our attention moved so that the line of interest could always be found at a fixed point in the display. The ability to link pointers to several files would enable a change in one pointer to invoke a corresponding change in all pointers linked to it. The WINDOW kernel described in Section One encompasses the above features.
An understanding of WINDOW must begin with the notion of "pointers". Every pointer is a triple consisting of a file number, a line number relative to the beginning of the file, and a character position relative to the beginning of a line. All editing is performed on characters indicated by a pointer. The first action taken after opening a file is the definition of a pointer through which the file may be accessed. A particular pointer may be declared the "current pointer" so that it is not always necessary to name a specific pointer in the various editing functions of the kernel.

The text displayed through a window is determined by a pointer which is associated with that window when the window is defined; this pointer must be one which has previously been defined upon a file. Thus a window might be said to display a single character. However, surrounding text, always at least one line, is also displayed. The number of lines displayed (the width of a window) is specified when the window is defined.

Any change to a pointer is reflected through the text displayed in all windows associated with that pointer. Thus if the line-part of a pointer is repeatedly incremented so that it points to successive lines in its file, the text displayed will likewise appear to move in a scroll-like fashion as the center line of each window shows those successive lines.
SECTION ONE

INTERFACE AND STRUCTURE OF THE KERNEL

The structure of the kernel system is a three-level hierarchy consisting of a file subsystem, a graphics subsystem, and an editing subsystem, respectively. The interface presented by each level is composed of function references which either indicate a state of the kernel system or effect a change in that state.

The following is a commentary on the specification of the kernel. Functions are provided here with informal parameter lists. Implicit parameters are enclosed within square brackets. Explicit parameters are enclosed within parentheses. The formal specification is located in Appendix One.
FILE SYSTEM

Indicators

UPIEX (file f)
- true iff its parameter
  is the identifier of
  an open file

ULIEX (file f; line l)
- true iff its parameter
  is the identifier of
  an existing line in
  an open file

FLINES (file f)
- number of lines in
  an open file

UMAXCH (file f; line l)
- number of characters
  in an existing line
  in an open file

UFS (file f; line l; position c)
- character in an
  existing line in
  an open file

Effectors

FILIN (file f)
- makes UPIEX = true

FIOUT (file f)
- makes UPIEX = false

INSLIN ("current pointer")
- makes FLINES=FLINES+1

DELIN ["current pointer"]
- makes FLINES=FLINES-1

INSACHAR (pointer p; character z)
- makes UMAXCH=UMAXCH+1
- UFS at end of line ← z

DELACHAR (pointer p)
- makes UMAXCH=UMAXCH-1
- delete from end of line

ALTCHAR (pointer p; character z)
- alters UFS

Note: Functions INSLIN, DELIN, INSACHAR, DELACHAR, and ALTCHAR may affect SCREEN. Functions INSLIN and DELIN may also affect SRCLIN.
WINDOW
Commentary

GRAPHICS SYSTEM

Indicators

WINEXISTS (window w)
= true iff its parameter is the identifier of an existing window

TWIND (window w)
= identifier of bottom line of existing window

BWIND (window w)
= identifier of top line of existing window

SRCFIL (screenline x)
= identifier of file associated with line on screen

SRCLIN (screenline x)
= identifier of line associated with line on screen

WINPOINT (window w)
= identifier of pointer associated with an existing window

SCREEN (screenline x; position c)
= physical display

-----------------

Note: Functions DEFWIND and DELWIND affect SCREEN.
EDITING SYSTEM

Indicators

FILE (pointer p)

= identifier of open file associated with an existing pointer

LINE (pointer p)

= identifier of existing line associated with an existing pointer

Effectors

REDEFAPOINT ["current pointer"]

(file f; line l; position c)
moves pointer to a particular file, line, character

MOVPNT ["current pointer"]

(line l; position c)
moves pointer to a particular line and character

INCPOINT ["current pointer"]

(displacement i)
moves pointer to a line relative to current line

NEXTLINE ["current pointer"]
moves pointer ahead one line

PREVLINE ["current pointer"]
moves pointer back one line

CHAR (pointer p)

= character position associated with existing pointer

NEXTCHAR (pointer p)
makes CHAR=CHAR+1

BACKCHAR (pointer p)
makes CHAR=CHAR-1

Note: Functions REDEFAPOINT, MOVPNT, INCPOINT, NEXTLINE, and PREVLINE may affect SRCFIL, SRCLIN, and SCREEN.
**window**

Commentary

**ISAPPOINT (pointer p)**

= true iff its parameter
is the identifier of an
existing pointer

**LINKED (pointer p1, p2)**

= true iff its parameters
are linked existing
pointers

**OPENED ["current pointer"]**

= true iff a
"current pointer"
is declared

**CURPNT**

= identifier of
"current pointer" iff
OPENED = true

**ISLNKD ["current pointer"]**

= true iff the
"current pointer's"
file is linked through
pointers to itself or
any other file(s)

**DEPOINT (pointer p; file f; line l; position c)**

makes ISAPPOINT = true

**DELPOINT (pointer p)**

makes ISAPPOINT = false

**LINK (pointer p1, p2)**

makes LINKED = true

**UNLINK (pointer p1, p2)**

makes LINKED = false

**OPENIT (pointer p)**

makes CURPNT = p
makes OPENED = true

**UNOPEN**

makes OPENED = false

Note: Function DELPOINT may affect WINEXISTS.
SECTION TWO

RELATIONSHIP BETWEEN THE KERNEL AND THE COMMAND INTERPRETER

The relationship between the kernel and the command interpreter is that of a two-level hierarchy with the kernel on the lower level. Note that all of the kernel functions could be accessed directly by the user, as specified. They would comprise a comprehensive graphics-based text editor. However, this direct-access implementation would prove too tedious and too complex for any person to use effectively. The command interpreter instead plays the role of the user of the kernel and in turn provides its own set of editing functions to the next-higher-level user; in the current implementation, that user is assumed to be, but is not limited to being, a person. This new set of functions is transparent in the sense that no valuable editing capabilities have been lost by performing the abstraction. It is also more "useful" in the sense that the user is now provided with what appears to be a "conventional" text editing program -- with unconventional graphics capabilities.
Relationship Between Kernel and Command Interpreter

The following examples illustrate the abstraction:

* The kernel refers to files by integer numbers. The command interpreter can deal with file numbers; it can also deal with user-defined file names. It maps names to numbers and vice versa while communicating between the user and the kernel.

* Many functions of the command interpreter map directly onto single kernel functions. The command "DFP 2 1" causes a reference of the kernel function "DEFPONIT(pointer 2; file 1; line 1; position 1)". If the user defines the name "MYFILE" equivalent to file number 1, then the command "DFP 2 #MYFILE" results in exactly the same reference of the kernel function.

* Other commands result in a programmed sequence of kernel function references. The command "I" enables the user to insert line after line of text into a file without the need for any intervening commands. This is similar in observable effect to insertions using conventional text editors. For every line of text entered, the kernel function "INSLIN" is referenced, followed by as many "NEXTCHAR" and "INSACHAR" references as there are characters in the line of text.

The usefulness of the WINDOW kernel is not limited to this one application; it may be considered the basis for a family of text editors. The behavior of a member of the family would be dependent upon its abstraction of the kernel's facilities.
OUR EXPERIENCE WITH THIS APPLICATION OF FORMAL SPECIFICATION

The formal specification of the WINDOW kernel was produced long before its implementation. During the process of implementing the kernel according to its specification, many system design changes were made.

In particular, the original specification called for the "PREVLINE" and "NEXTLINE" functions to take a painter as an explicit parameter. We decided to eliminate that parameter and to implicitly use the "current pointer" instead. The role of the "current pointer" was similarly expanded throughout the kernel. Thus, textual revisions to the formal specification were made in parallel with program development when those changes involved the semantics of the specification.

The original specification was found to be incomplete with respect to the existence of some state-indicating functions without any complementary state-effecting functions. That was the case for the "CURPNT" and "OPENED" functions. "OPENIT" and "UNOPEN" were added during implementation.
Many other functions have come and gone in the process of fine-tuning the kernel. For example, the "ISLNKD" function was added after the kernel implementation was complete. It reduced the notational complexity of the specification, reduced the size of the kernel source and object programs, and reduced the execution-times of many kernel and command interpreter functions. When the kernel had formerly relied upon a sequence of function calls to determine file linkages through pointers, repeated tests were redundant because no memory existed between function interfaces of previous calls. In the current implementation, one test is performed in "LINK" and one in "UNLINK"; the result is simply recorded within "ISLNKD" for future reference.

The decomposition indicated by the formal specification proved an excellent basis for a statistical analysis of performance. The execution times and frequency counts of every function can be gathered, and in tabulated form point directly to areas of inefficiency. Statistics prompted, for example, the inclusion of the "FLINES" function after implementation was complete.

The formal specification served as the source of information in the production of the commentary of Section One. The specification was also used during implementation of the command interpreter as the definitive statement of the effects of referencing specific kernel functions.
Experience with Formal Specification

The formal specification can be the basis for proving assertions about properties of the kernel. On an informal basis alone, simple properties, such as the assertion that the UNLINK function cannot be driven to dissociate a pointer from itself, can be proven readily from the observation that the formal specification indicates an error trap within that function ("ERROR(25)") for that special case ("p=q").

In conclusion, had the formal interface specification not existed, the interface between the kernel and the command interpreter would have been less rigorously defined. Their individual responsibilities, intended to be disjoint, would have overlapped, resulting in a more difficult program development. Furthermore, the kernel would have lost much of its value as the basis for a family of text editing programs. The existence of shared responsibilities, leading to shared knowledge of their implementations, would have demanded of all other potential command interpreters that they too share the same responsibilities and knowledge. That would have greatly restricted the classes of programs which could belong to the family.
APPENDIX ONE

FORMAL SPECIFICATION

Let \( \text{DISP}(p,i) = \begin{cases} \text{'ULIEX'(\text{'FILE'}(p), \text{'LINE'}(p)+i)} & \text{if defined} \\ \text{undefined} & \text{else} \end{cases} \)

Function \text{UFIEX}

possible values: (boolean) true, false

initial value: false

parameters: integer \( f \)

effect:

\[
\text{call ERROR}(00) \text{ if } [ f < 0 \text{ or } f > p1 ]
\]

Function \text{ULIEX}

possible values: (boolean) if \( \text{'FLINES'}(f) \geq 1 \) then true else false

initial value: false

parameters: integer \( f, l \)

effect:

\[
\text{call ERROR}(00) \text{ if } [ f < 0 \text{ or } f > p1 ] \\
\text{call ERROR}(16) \text{ if } [ \text{'UFIEX'}(f) = \text{false} ] \\
\text{call ERROR}(01) \text{ if } [ l < 1 \text{ or } l > p2 ]
\]

Function \text{UMAXCH}

possible values: integer \( 0:p3 \)

initial value: undefined

parameters: integer \( f, l \)

effect:

\[
\text{call ERROR}(00) \text{ if } [ f < 0 \text{ or } f > p1 ] \\
\text{call ERROR}(16) \text{ if } [ \text{'UFIEX'}(f) = \text{false} ] \\
\text{call ERROR}(01) \text{ if } [ l < 1 \text{ or } l > p2 ] \\
\text{call ERROR}(17) \text{ if } [ \text{'ULIEX'}(f,l) = \text{false} ]
\]

Function \text{UFS}

possible values: integer \( 0:127 \)

initial value: undefined

parameters: integer \( f, l, c \)

effect:

\[
\text{call ERROR}(00) \text{ if } [ f < 0 \text{ or } f > p1 ] \\
\text{call ERROR}(16) \text{ if } [ \text{'UFIEX'}(f) = \text{false} ] \\
\text{call ERROR}(01) \text{ if } [ l < 1 \text{ or } l > p2 ] \\
\text{call ERROR}(17) \text{ if } [ \text{'ULIEX'}(f,l) = \text{false} ] \\
\text{call ERROR}(02) \text{ if } [ c < 1 \text{ or } c > p3 ] \\
\text{call ERROR}(05) \text{ if } [ c > \text{'UMAXCH'}(f,l) ]
\]
Function ISAPoint
possible values: (boolean) true, false
initial value: false
parameters: integer \( p \)
effect:

\[
\text{call ERROR(03) if } [ p < 1 \text{ or } p > 6 ]
\]

Function FILE
possible values: integer \( 0:p1 \)
initial value: undefined
parameters: integer \( p \)
effect:

\[
\text{call ERROR(03) if } [ p < 1 \text{ or } p > 6 ]
\]
\[
\text{call ERROR(15) if } [ \text{ISAPoint}(p) = \text{false} ]
\]

Function LINE
possible values: 1:p2
initial value: undefined
parameters: integer \( p \)
effect:

\[
\text{call ERROR(03) if } [ p < 1 \text{ or } p > 6 ]
\]
\[
\text{call ERROR(15) if } [ \text{ISAPoint}(p) = \text{false} ]
\]

Function CHAR
possible values: 0:p3
initial value: undefined
parameters: integer \( p \)
effect:

\[
\text{call ERROR(03) if } [ p < 1 \text{ or } p > 6 ]
\]
\[
\text{call ERROR(15) if } [ \text{ISAPoint}(p) = \text{false} ]
\]
Function LINKED
possible values: (boolean) true, false
initial value: false
parameters: integer p, q
effect:

    call ERROR(03) if [ p < 1 .or. p > p6 ]
    call ERROR(04) if [ q < 1 .or. q > p6 ]
    call ERROR(15) if [ 'ISAPoint'(p) = false ]
    call ERROR(15) if [ 'ISAPoint'(q) = false ]

Function DEFPPOINT
possible values: none
initial value: not applicable
parameters: integer p, f, l, c
effect:

    call ERROR(03) if [ p < 1 .or. p > p6 ]
    call ERROR(14) if [ 'ISAPoint'(p) = true ]
    call ERROR(00) if [ f < 0 .or. f > p1 ]
    call ERROR(01) if [ l < 1 .or. l > p2 ]
    call ERROR(02) if [ c < 1 .or. c > p3 ]
    call ERROR(16) if [ 'ULiEX'(f) = false ]
    call ERROR(17) if [ 'ULiEX'(f, l) = false ]
    call ERROR(05) if [ c > 'UMAXCH'(f, l) ]
    ISAPoint(p) = true
    FILE(p) = f
    LINE(p) = l
    CHAR(p) = c
    LINKED(p, p) = true
Function DELPOINT
possible values: none
initial value: not applicable
parameters: integer p
effect:

- call ERROR(03) if \[ p < 1 \text{ or } p > p6 \]
- call ERROR(15) if \[ 'ISAPPOINT'(p) = false \]
- ISAPPOINT(p) = false
- FILE(p) = undefined
- LINE(p) = undefined
- CHAR(p) = undefined
- for all q \[ LINKED(p, q) = LINKED(q, p) = false \]
- for all w \[ if 'WINEXISTS'(w) and 'WINPOINT'(w) = p then DELWIND(w) \]
- if \[ 'CURPNT' = p \text{ and 'OPENED' = true } \] then UNOPEN

Function LINK
possible values: none
initial value: not applicable
parameters: integer p, q
effect:

- call ERROR(03) if \[ p < 1 \text{ or } p > p6 \]
- call ERROR(04) if \[ q < 1 \text{ or } q > p6 \]
- call ERROR(15) if \[ 'ISAPPOINT'(p) = false \]
- call ERROR(15) if \[ 'ISAPPOINT'(q) = false \]
- call ERROR(41) if \[ 'LINKED'(p, q) = true \]
- call ERROR(87) if \[ 'LINKED'(q, p) = true \]
- LINKED(p, q) = true
- LINKED(q, p) = true
- for 'FILE'(p) and 'FILE'(q) \[ ISLNKD = true \]
FUNCTION UNLINK
possible values: none
initial value: not applicable
parameters: integer p, q
effect:

- call ERROR(03) if \[ p < 1 \text{ or } p > pB \]
- call ERROR(04) if \[ q < 1 \text{ or } q > pB \]
- call ERROR(15) if \[ 'ISAPPOINT'(p) = false \]
- call ERROR(15) if \[ 'ISAPPOINT'(q) = false \]
- call ERROR(12) if \[ 'LINKED'(p, q) = false \]
- call ERROR(13) if \[ 'LINKED'(q, p) = false \]
- call ERROR(25) if \[ p = q \]
- LINKED(q, p) = false
- LINKED(p, q) = false
for all f such that \('UFIEX'(f) = true) if there exists any qq such that
  \( 'ISAPPOINT'(qq) = true \text{ and } 'FILE'(qq) = f \)
  and
  there exists any pp such that
  \( 'ISAPPOINT'(pp) = true \text{ and } 'FILE'(pp) = f \)
  and
  \( (pp \neq p \text{ and } pp \neq q) \)
  and
  \( (qq \neq p \text{ and } qq \neq q) \)
  and
  \( (pp \neq qq \text{ and } 'LINKED'(pp, qq) = true) \)
then
  ISLNKD = true for f
else
  ISLNKD = false for f

FUNCTION WINEXISTS
possible values: (boolean) true, false
initial values: false
parameters: integer w
effect:

- call ERROR(89) if \[ w < 1 \text{ or } w > p4 \text{ div } 2 \]
WINDOW

Formal Specification

Function TWIND
possible values: l:p4
initial value: undefined
parameters: integer w
effect:

   call ERROR(89) if [ w < 1 .or. w > p4 div 2 ]
   call ERROR(11) if ['WINEXISTS'(w) = false ]

Function BWIND
possible values: l:p4
initial value: undefined
parameters: integer w
effect:

   call ERROR(89) if [ w < 1 .or. w > p4 div 2 ]
   call ERROR(11) if ['WINEXISTS'(w) = false ]

Function SRCFIL
possible values: B:p1
initial value: undefined
parameters: integer r
effect:

   call ERROR(88) if [ r < 1 .or. r > p4 ]

Function SRCLIN
possible values: l:p2
initial value: undefined
parameters: integer r
effect:

   call ERROR(88) if [ r < 1 .or. r > p4 ]
WINDOW

Formal Specification

Function WINPOINT
possible values: 1:p6
initial value: undefined
parameters: integer w
effect:

    call ERROR(89) if [ w < 1 .or. w > p4 div 2 ]
    call ERROR(11) if [ 'WINEXISTS'(w) = false ]

Function DELWIND
possible values: none
initial value: not applicable
parameters: integer w
effect:

    call ERROR(89) if [ w < 1 .or. w > p4 div 2 ]
    call ERROR(11) if [ 'WINEXISTS'(w) = false ]
    for all r such that ( 'BWIND'(w) <= r <= 'TWIND'(w) )
    [ SRCFIL(r) = undefined
        SRCLIN(r) = undefined
        for all c such that (1 <= c <= p5) SCREEN(r,c)=" " ]
    TWIND(w) = undefined
    BWIND(w) = undefined
    WINPOINT(w) = undefined
    WINEXISTS(w) = false
Window

Formal Specification

Function DEFWIND
possible values: none
initial value: not applicable
parameters: integer w, p, bw, tw
effect:

   call ERROR(89) if [ w < 1 or w > p4 div 2 ]
   call ERROR(19) if [ 'WINEXISTS'(w) = true ]
   call ERROR(21) if [ tw < 1 or tw > p4 ]
   call ERROR(22) if [ bw < 1 or bw > p4 ]
   call ERROR(23) if [ bw >= tw - 1 ]
   call ERROR(43, or 44) if [ exists j such 'WINEXISTS'(j)=true and.
      [ 'BWIND'(j) <= bw < 'TWIND'(j) or.
      'BWIND'(j) < tw <= 'TWIND'(j) ] ]
   call ERROR(83) if [ p < 1 or p > p6 ]
   call ERROR(15) if [ 'ISAPoint'(p) = false ]
   TWIND(w) = tw
   BWIND(w) = bw
   WINEXISTS(w) = true
   WINPOINT(w) = p
   SRCFIL(bw) = SRCFIL(tw) = 'FILE'(p)
   SRCLIN(bw) = SRCLIN(tw) = undefined
   let m = entier(( tw + bw ) / 2)
   for all r such that ( bw < r < tw )
      [ SRCFIL(r) = 'FILE'(p)
      SRCLIN(r) = 'DISP'(p, r-m)
      for all c such that ( 1 <= c <= p5)
      SCREEN(r,c) = 'UFS'(SRCFIL(r),SRCLIN(r),c)
      if defined else " " ]

Function NEXTCHAR
possible values: none
initial value: not applicable
parameters: p
effect:

   call ERROR(83) if [ p < 1 or p > p6 ]
   call ERROR(15) if [ 'ISAPoint'(p) = false ]
   call ERROR(24) if [ 'CHAR'(p) = p3 ]
   CHAR(p) = 'CHAR'(p) + 1
Window Formal Specification

Function BACKCHAR
possible values: none
initial value: not applicable
parameters: p
effect:

  call ERROR(83) if \[ p < 1 \text{ or } p > 6 \]
call ERROR(15) if \[ 'ISAPPOINT'(p) = \text{false} \]
call ERROR(38) if \[ 'CHAR'(p) = 1 \]
CHAR(p) = 'CHAR'(p) - 1

Function INCPOINT
possible values: none
initial value: not applicable
parameters: integer i
effect:

  call ERROR(10) if \[ 'OPENED' = \text{false} \]
let p = 'CURPNT'
call ERROR(27) if \[ 'DISP'(p,i) = \text{undefined} \]
call ERROR(28) if \[ 'ISLNKD' = \text{true}\text{and}\text{for any q, neq, p such that} \]
\[ 'LINKED'(p,q) = \text{true} \]
\[ 'DISP'(q,i) = \text{undefined} \]
for all q such that \[ 'LINKED'(p,q) = \text{true} \]
\[ LINE(q) = 'DISP'(q,i) \]
for all w such that \[ 'WINEXISTS'(w) = \text{true}\text{and}'WINPOINT'(w) = q \]
\[ \text{let m = enter}(( 'TWIND'(w) + 'BWIND'(w)) / 2) \]
for all r such that \[ 'BWIND'(w) < r < 'TWIND'(w) \]
\[ 'SRCLIN'(r) = 'DISP'(q, r - m) \]
for all c such that \( 1 \leq c \leq p5 \)
SCREEN(r,c) = 'UFS('SRCFIL'(r),
SRCLIN(r,c)
if defined else " ")
Formal Specification

Function OPENIT
possible values: none
initial value: not applicable
parameters: integer p
effect:

  call ERROR(83) if [ p < 1 .or. p > p6 ]
  call ERROR(15) if [ 'ISAPPOINT'(p) = false ]
  call ERROR(29) if [ 'OPENED' = true ]
  OPENED= true
  CURPNT= p

Function UNOPEN
possible values: none
initial value: not applicable
parameters: none
effect:

  call ERROR(18) if [ 'OPENED' = false ]
  OPENED = false
  CURPNT= undefined

Function OPENED
possible values: (boolean) true, false
initial value: false
parameters: none
effect:
Function FILIN
possible values: none
initial value: not applicable
parameters: integer f
effect:

- call ERROR(00) if \( f < 0 \) or \( f > p1 \)
- call ERROR(42) if \( \text{UFIEX}(f) = \text{true} \)
  \( \text{UFIEX}(f) = \text{true} \)

Function FILOUT
possible values: none
initial value: not applicable
parameters: integer f
effect:

- call ERROR(00) if \( f < 0 \) or \( f > p1 \)
- call ERROR(16) if \( \text{UFIEX}(f) = \text{false} \)
- call ERROR(20) if \( \text{for any } p \text{ such that } \text{FILE}(p) = f \)
  \( \text{UFIEX}(f) = \text{false} \)
  \( \text{for all } I \)
  \( \text{UFIEX}(I,f) = \text{false} \)
  \( \text{UMAXCH}(I,f) = \text{undefined} \)
  \( \text{for all } c \left[ \text{UFS}(I,f,c) = \text{undefined} \right] \)
Function DELINE
possible values: none
initial value: not applicable
parameters: none
effect:

call ERROR(18) if [ 'OPENED' = false ]
let p = 'CURPNT'
call ERROR(17) if [ 'ULIEK'('FILE'(p), 2) = false ]
call ERROR(17) if [ 'ISLNKD' = true and for any q .neq. p
such that ('LINKED'(p,q) = true)
[ 'ULIEK'('FILE'(q), 2) = false ] ]
for all q such that ('LINKED'(p,q) = true)
[ FLINES('FILE'(p)) = FLINES('FILE'(p)) - 1
for all w such that ('WINEXIST'S'(W)=true and.'WINPOIN'T'(w)=q)
[ let m = entier(('TWIND'(w) + 'BWIN'D'(w)) / 2)
for all r such that ('BWIN'D'(w) < r < 'TWIND'(w))
[ SRCLIN(r)= DISP(q, r-m)
for all c such that (1 <= c <= p5)
SCREEN(r,c)= 'UFS'('SRCFIL'(r),
SRCLIN(r,c)
if defined else " " ] ] ] ]

Function CURPNT
possible values: 1:p6
initial value: undefined
parameters: none
effect:
Function MOVPNT
possible values: none
initial value: not applicable
parameters: integer \( l, c \)
effect:

let \( p = 'CURPNT' \)
call ERROR(10) if \[ 'OPENED' = \text{false} \]
call ERROR(81) if \[ 1 < l \text{ or } l > p2 \]
call ERROR(17) if \[ ULIE('FILE'(p), l) = \text{false} \]
call ERROR(35) if \[ c < 1 \text{ or } c > \text{UMAXCH('FILE'(p),l)+1} \]
call ERROR(32) if \[ 'ISLNKD'=\text{true} \text{ and } \forall q \neq p \text{ such that } ('LINKED'(p,q) = \text{true}) \]
\[ ULIE('FILE'(q),l) = \text{false} \]
call ERROR(33) if \[ 'ISLNKD'=\text{true} \text{ and } \forall q \neq p \text{ such that } ('LINKED'(p,q) = \text{true}) \]
\[ c > \text{UMAXCH('FILE'(q),l)+1} \]
if \[ 'ISLNKD' = \text{true} \] then for all \( q \neq p \text{ such that } ('LINKED'(p,q) = \text{true}) \)
\[ \text{LINE}(q)=1 \]
\[ \text{CHAR}(q)=c \]
\[ \text{LINE}(p)=1 \]
\[ \text{CHAR}(p)=c \]
for all \( q \) such that \( ('LINKED'(p,q) = \text{true}) \)
\[ \forall w \text{ such that } ('\text{WINEXISTS}'(w)=\text{true} \text{ and } 'WINPOINT'(w)=q) \]
\[ \text{let } m = \text{entier}(('\text{TWIND}'(w) + '\text{BWIND}'(w)) / 2) \]
for all \( r \) such that \( ('\text{BWIND}'(w) < r < '\text{TWIND}'(w)) \)
\[ \text{SRCLIN}(r)=\text{DISP}(q, r-m) \]
for all \( c \) such that \( 1 \leq c \leq p5 \)
\[ \text{SCREEN}(r,c)=\text{'UFS('SRCFIL'(r),} \]
\[ \text{SRCLIN}(r),c) \]
if defined else " " ]}
Function REDEFAPoint
possible values: none
initial value: not applicable
parameters: integer f, l, c

effect:

call ERROR(10) if ['OPENED' = false]
call ERROR(36) if ['ISLNKD' = true .and. exists q such that
  ('LINKED'('CURPNT', q) = true)]
call ERROR(08) if [f < 0 .or. f > p1]
call ERROR(01) if [l < 1 .or. l > p2]
call ERROR(02) if [c < 1 .or. c > p3]
call ERROR(16) if ['UFIEX'(l) = false]
call ERROR(17) if ['ULIEX'(f, l) = false]
call ERROR(05) if [c > 'UMAXCH'(f, l)]
LINE('CURPNT') = 1
CHAR('CURPNT') = c
FILE('CURPNT') = f
for all w such that ('WINEXISTS'(w), .and., 'WINPOINT'(w) = 'CURPNT')
  [let m = int('TWIND'(w) + 'BWIND'(w)) / 2]
  for all r such that ('BWIND'(w) < r < 'TWIND'(w))
    [SRCFIL(r) = 'FILE'('CURPNT')
     SRCLIN(r) = DISP('CURPNT', r - m)
     for all c such that (1 <= c <= p5)
       SCREEN(r, c) = 'UFS'(SRCFIL(r),
                           SRCLIN(r), c)
     if defined else " "] }
Function NEXTLINE
possible values: none
initial value: not applicable
parameters: none
effect:

call ERROR(18) if ['OPENED' = false ]
call ERROR(45) if [ for all q such that
( 'LINKED'('CURPNT',q) = true )
[ there does not exist l such that
(l > 'LINE'(q) ,and,
'LILIEX'('FILE'(q),l) = true)]
for all q such that ( 'LINKED'('CURPNT',q) = true )
[ LINE(q) = 'LINE'(q) + 1
CHAR(q) = 1 ]
for all q such that ( 'LINKED'('CURPNT',q) = true )
[ for all w such that ( 'WINEXISTS'(w),and,'WINPOINT'(w)=q)
[ let m = entier(( 'TWIND'(w) + 'BWIND'(w)) / 2)
for all r such that ( 'BWIND'(w) < r < 'TWIND'(w))
[ SRCLIN(r)= DISP(q, r-m)
for all c such that (1 <= c <= p5)
SCREEN(r,c)= 'UFS'('SRCFIL'(r),
'SRCLIN'(r),c)
if defined else " " ])]}
Function PREVLINE
possible values: none
initial value: not applicable
parameters: none

effect:

call ERROR(18) if [ 'OPENED' = false ]
call ERROR(45) if [ for all q such that
     ( 'LINKED'('CURPNT',q) = true )
     [ there does not exist l such that
       ( 1 < 'LINE'(q) .and.
         'ULIEX'('FILE'(q),l) = true )]]
for all q such that ('LINKED'('CURPNT',q) = true )
[ LINE(q)= 'LINE'(q) - 1
  CHAR(q)= 1 ]
for all q such that ('LINKED'('CURPNT',q) = true)
 [ for all w such that ('WINEXISTS'(w) .and. 'WINPOINT'(w)=q)
   [ let m = entier(('TWIND'(w) + 'BWIND'(w)) / 2)
     for all r such that ('BWIND'(w) < r < 'TWIND'(w))
       [ SRCLIN(r)= DISP(q, r-m)
         for all c such that (1 <= c <= p5)
           SCREEN(r,c)= 'UFS'( 'SRCPIL'(r),
            SRCLIN(r),c)
           if defined else " " ]]]

Function ALTCHAR
possible values: none
initial value: not applicable
parameters: integer p, z

effect:

call ERROR(83) if [ p < 1 .or. p > p6 ]
call ERROR(15) if [ 'ISAPONT'(p) = false ]
call ERROR(34) if [ z < 0 .or. z > 127 ]
call ERROR(86) if [ 'CHAR'(p) = 8 ]
UFS('FILE'(p),'LINE'(p),'CHAR'(p))= z
for all q such that ( (q .neq. p) .and. 'LINKED'(p,q))
 [ for all w such that ('WINEXISTS'(w) .and. 'WINPOINT'(w)=q)
   [ if ('SRCLIN'('BWIND'(w)+1) if defined else 0) <=
     'LINE'(q) <=
     'SRCLIN'('TWIND'(w)-1))
     then SCREEN(entier(('TWIND'(w)+'BWIND'(w))/2),
       'CHAR'(q)= z)]
Function **DELACHAR**

possible values: none

initial value: not applicable

parameters: integer p

**effect:**

```plaintext
call ERROR(03) if [ p < 1 .or. p > 6 ]
call ERROR(15) if ['ISAPPOINT'(p) = false ]
call ERROR(19) if ['CHAR'(p).neq.'UMAXCH'('FILE'(p),'LINE'(p))]
call ERROR(19) if ['CHAR'(p) = 6 ]
UFS('FILE'(p),'LINE'(p),'CHAR'(p)) = undefined
UMAXCH('FILE'(p),'LINE'(p)) =
'UMAXCH'('FILE'(p),'LINE'(p)) - 1
for all q such that ([q .neq. p] .and. 'LINKED'(p,q))
[ for all w such that ('WINEXISTS'(w) .and. 'WINPOINT'(w)=q)
  [ if ('SRCLIN'('BWIND'(w)+l) if defined else 8) <=
      'LINE'(q) <=
      'SRCLIN'('TWIND'(w)-1))
    then SCREEN(entier(('TWIND'(w)+'BWIND'(w))/2),
      'CHAR'(q))= " "]
```
Function ISLNKD
possible values: (boolean) true, false
initial value: false
parameters: none
effect:

    call ERROR(18) if ['OPENED' = false ]

Function INSACHAR
possible values: none
initial value: not applicable
parameters: integer p, z
effect:

    call ERROR(83) if [ p < 1 .or. p > p6 ]
    call ERROR(15) if ['ISAPPOINT'(p) = false ]
    call ERROR(34) if [ z < 0 .or. z > 127 ]
    call ERROR(37) if ['CHAR'(p).neq.'UMAXCH('FILE'(p),'LINE'(p))+1]
    call ERROR(IS) if ['ISAPOINTXp) - false  
    call ERROR(34) if [ z < 0 .or. z > 127 ]
    call ERROR(37) if ['CHAR'(p).neq.'UMAXCH('FILE'(p),'LINE'(p))+1]

    UFS('FILE'(p),'LINE'(p),'CHAR'(p)) = z
for all q such that ((q .neq. p) .and. 'LINKED''(p,q))
    [ for all w such that ('WINEXISTS'(w) .and. 'WINPOINTER'"(w)=q)
      [ if (('SRCLIN'('BWIND'(w)))+1 if defined else 0) <=
        'LINE'(q) <=
        'SRCLIN'('TWIND'(w))-1)
      then SCREEN (integer ('TWIND'(w)+'BWIND'(w))/2),
        'CHAR'(q)= z]]

Function SCREEN
possible values: string
initial value: " "
parameters: integer r, c
effect:
Figure 1.

Figure 1 is a photograph of a CRT terminal screen. The screen is displaying text from this paper through two windows delimited by lines of asterisks. The upper window's text is from Appendix One; the lower window's text is from the specification commentary of Section One. The value of multiple windows on one screen is illustrated by this juxtaposition of related text concerning function FILIN.