INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume VIII - User Interface Subsystem

S. Barker
Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209

September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 71

FOR THE COMMANDER:

BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 71

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright-Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.
**REPORT DOCUMENTATION PAGE**

<table>
<thead>
<tr>
<th>1a. REPORT SECURITY CLASSIFICATION</th>
<th>1b. RESTRICTIVE MARKINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclassified</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2a. SECURITY CLASSIFICATION AUTHORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Data Corporation;</td>
</tr>
<tr>
<td>Integration Technology Services</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2b. DECLASSIFICATION/DOWNGRADING SCHEDULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for Public Release;</td>
</tr>
<tr>
<td>Distribution is Unlimited.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. DISTRIBUTION/AVAILABILITY OF REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for Public Release;</td>
</tr>
<tr>
<td>Distribution is Unlimited.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. PERFORMING ORGANIZATION REPORT NUMBER(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UM 620344300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. MONITORING ORGANIZATION REPORT NUMBER(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRDC-TR-90-8007 Vol. VIII, Part 13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6a. NAME OF PERFORMING ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Data Corporation;</td>
</tr>
<tr>
<td>Integration Technology Services</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6b. OFFICE SYMBOL (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRDC/MTI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7a. NAME OF MONITORING ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRDC/MTI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7b. ADDRESS (City, State, and ZIP Code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WPAFB, OH 45433-6533</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8a. NAME OF FUNDING/SPONSORING ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wright Research and Development Center,</td>
</tr>
<tr>
<td>Air Force Systems Command, USAF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8b. OFFICE SYMBOL (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRDC/MTI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM.</th>
</tr>
</thead>
<tbody>
<tr>
<td>F33600-87-C-0464</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10. SOURCE OF FUNDING NOS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRDC/MTI Project Priority 6203</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11. TITLE (In Se, block 19)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Terminal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12. PERSONAL AUTHOR(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Dynamics Research Corporation: Barker, S. et al.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13a. TYPE OF REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Report</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13b. TIME COVERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/1/87-12/31/90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>14. DATE OF REPORT (Yr.,Mo.,Day)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990 September 30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15. PAGE COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16. SUPPLEMENTARY NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRDC/MTI Project Priority 6203</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17. COSATI CODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD GROUP SUB GR.</td>
</tr>
<tr>
<td>1308 0905</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>18. SUBJECT TERMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Continue on reverse if necessary and identify block no.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19. ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Continue on reverse if necessary and identify block number)</td>
</tr>
</tbody>
</table>

This manual describes the program callable interface to the Integrated Information Support System Virtual Terminal, the Virtual Terminal Commands, and provides terminal implementation information for programmers.

**LOCK 11:**

**INTEGRATED INFORMATION SUPPORT SYSTEM**

Vol VIII - User Interface Subsystem


<table>
<thead>
<tr>
<th>20. DISTRIBUTION/AVAILABILITY OF ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclassified/Unlimited x SAME AS RPT. DTIC USERS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>21. ABSTRACT SECURITY CLASSIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclassified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>22a. NAME OF RESPONSIBLE INDIVIDUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>David L. Judson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>22b. TELEPHONE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Include Area Code) (513) 255-7371</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>22c. OFFICE SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRDC/MTI</td>
</tr>
</tbody>
</table>

**DD FORM 1473, 83 APR**

EDITION OF 1 JAN 73 IS OBSOLETE
This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

<table>
<thead>
<tr>
<th>SUBCONTRACTOR</th>
<th>ROLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Data Corporation</td>
<td>Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.</td>
</tr>
<tr>
<td>D. Appleton Company</td>
<td>Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.</td>
</tr>
<tr>
<td>ONTEK</td>
<td>Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.</td>
</tr>
<tr>
<td>Simpact Corporation</td>
<td>Responsible for Communication development.</td>
</tr>
<tr>
<td>Structural Dynamics</td>
<td>Responsible for User Interfaces, Virtual Terminal Interface, and Network Transaction Manager design, development, implementation, and support.</td>
</tr>
<tr>
<td>Arizona State University</td>
<td>Responsible for test bed operations and support.</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 INTRODUCTION</td>
<td>1-1</td>
</tr>
<tr>
<td>2.0 DOCUMENTS</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1 Reference Documents</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2 Terms and Abbreviations</td>
<td>2-1</td>
</tr>
<tr>
<td>3.0 VIRTUAL TERMINAL COMMANDS</td>
<td>3-1</td>
</tr>
<tr>
<td>3.1 General</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2 Command Descriptions</td>
<td>3-2</td>
</tr>
<tr>
<td>3.3 Input-Output Routines</td>
<td>3-7</td>
</tr>
<tr>
<td>4.0 TERMINAL IMPLEMENTATION</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1 Adding New Terminals</td>
<td>4-1</td>
</tr>
<tr>
<td>4.2 General Purpose Device Driver</td>
<td>4-1</td>
</tr>
<tr>
<td>4.3 Special Case Device Driver</td>
<td>4-2</td>
</tr>
</tbody>
</table>

## APPENDICES

<table>
<thead>
<tr>
<th>APPENDIX</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A VIRTUAL TERMINAL CHARACTER SET</td>
<td>A-1</td>
</tr>
<tr>
<td>B COMMAND REFERENCE</td>
<td>B-1</td>
</tr>
<tr>
<td>C DEVICE DRIVER SUPPORT ROUTINES</td>
<td>C-1</td>
</tr>
<tr>
<td>D DEVICE DRIVER INCLUDE FILES</td>
<td>D-1</td>
</tr>
<tr>
<td>E SAMPLE DEVICE DRIVER (DEC VT-100 - MONOCHROME)</td>
<td>E-1</td>
</tr>
<tr>
<td>F SAMPLE DEVICE DRIVER (IBM-3270 - COLOR)</td>
<td>F-1</td>
</tr>
</tbody>
</table>
SECTION 1
INTRODUCTION

This manual describes the program callable interface to the Integrated Information Support System Virtual Terminal, the Virtual Terminal commands, and provides terminal implementation information for programmers who wish to add new terminal types to the system. Although the program callable interface is NOT supported in IISS Release 3.0, it will be supported in later releases.

This manual is intended for application and system programmers working in the IISS environment.
SECTION 2
DOCUMENTS

2.1 Reference Documents


2.2 Terms and Abbreviations

American Standard Code for Information Interchange: (ASCII), the character set defined by ANSI X3.4 and used by most computer vendors.
Application Interface: (AI), subset of the IISS User Interface that consists of the callable routines that are linked with applications that use the Form Processor or Virtual Terminal. The AI enables applications to be hosted on computers other than the host of the User Interface.

Application Process: (AP), a cohesive unit of software that can be initiated as a unit to perform some function or functions.

Attribute: field characteristic such as blinking, highlighted, black, etc. and various other combinations. Background attributes are defined for forms or windows only. Foreground attributes are defined for items. Attributes may be permanent, i.e., they remain the same unless changed by the application program, or they may be temporary, i.e., they remain in effect until the window is redisplayed.

Communication Services: allows on host interprocess communication and inter-host communication between the various Test Bed subsystems.

Computer Program Configuration Item: (CPCI), an aggregation of computer programs or any of their discrete portions, which satisfies an end-use function and is designed by the ICAM Program Office for ICAM Configuration Management.

Data stream: device I/O commands and their accompanying data.

Device Drivers: (DD), software modules written to handle I/O for a specific kind of terminal. The modules map terminal specific commands and data to a neutral format. Device drivers are part of the UI Virtual Terminal.

Extended Binary Coded Decimal Interchange Code: (EBCDIC), the character set used by a few computer vendors (notable IBM) instead of ASCII.

Field: two dimensional space on a terminal screen.

Integrated Information Support System: (IISS), a computing environment used to investigate, demonstrate, test the concepts and produce application for information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous data bases supported by heterogeneous computers interconnected via a Local Area Network.

Logical Device: a conceptual device that identifies a top level window of an application. It is used to distinguish between multiple applications running simultaneously on a physical device. NOTE that a single application can have more than one logical device. To the end user this also appears as multiple applications running simultaneously.
Network Transaction Manager: (NTM), IISS subsystem that performs the coordination, communication and housekeeping functions required to integrate the Application Processes and System Services resident on the various hosts into a cohesive system.

Operating System: (OS), software supplied with a computer which allows it to supervise its own operations and manage access to hardware facilities such as memory and peripherals.

Physical Device: a hardware terminal.

Reverse Virtual Terminal: a subset of IISS User Interface that translates IBM 3270 data stream to Virtual Terminal data stream and vice versa.

User Interface: (UI), IISS subsystem that controls the user's terminal and interfaces with the rest of the system. The UI consists of two major subsystems: The User Interface Development System (UIDS) and the User Interface Management System (UIMS).

User Interface Management System: (UIMS), the run time UI. It consists of the Form Processor, Virtual Terminal, Application Interface and the User Interface Services.

User Interface Monitor: (UIM), part of the Form Processor that handles messaging between the NTM and the UI. It also provides authorization checks and initiates applications.

User Interface/Virtual Terminal Interface: (UI/VTI), another name for the User Interface.

Virtual Terminal: (VT), subset of the IISS User Interface that performs the interfacing between different terminals and the UI. This is done by defining a specific set of terminal features and protocols which must be supported by the UI software which constitutes the virtual terminal definition. Specific terminals are then mapped against the virtual terminal software by specific software modules written for each type of real terminal supported.

Virtual Terminal Interface: (VTI), the callable interface to the VT.

Window: dynamic area of a form in which predefined forms may be placed at runtime.

Window Manager: a facility which allows the following to be manipulated: size and location of windows, the device on which an application is running, the position of a form within a window. It is part of the Form Processor.
SECTION 3
VIRTUAL TERMINAL COMMANDS

3.1 General

The Virtual Terminal accepts two kinds of data: Graphic (or printable) Characters which are displayed on the screen, and Commands which affect the way in which Graphic Characters are displayed.

The format of the following command descriptions is: the command name and short description, the command syntax, and a detailed description of the command. In the command syntax, characters within angle brackets (e.g. <ESC>) indicate Control Characters (codings depend on your system character set - see Appendix A), Pn indicates a Numeric Parameter, Ps indicates a Selective Parameter, an ellipsis (...) indicates additional unspecified characters, and all other characters stand for themselves.

Parameters are represented in ordinary human-readable decimal form, with Numeric Parameters representing numbers (such as a row number or the number of times to repeat a function), and Selective Parameters standing for selections from a list of options with multiple selections separated by semicolons. Unless specified otherwise, Numeric Parameters indicate the number of times to repeat the specified function, omitted Numeric Parameters are taken to be 1, and omitted Selective Parameters are taken to be 0.

The Virtual Terminal screen consists of an arbitrary number of rows numbered from 1 to n, and an arbitrary number of columns numbered from 1 to m; the actual size is specified by the Define Window command. The standard ordering of objects is from top to bottom and left to right, with wrap-around from the last object to the first. In the command descriptions, "next" refers to this order, "previous" to its reverse. For example, from row 6 column 80 on an 80 character wide screen, the next character position is row 7 column 1, and the previous character position is row 6 column 79.

Any command whose effect is limited to a single field (including Graphic Characters) will cause the cursor to move to the next unprotected field before the command takes effect if the cursor is in a protected field when the command is received. If there are no unprotected fields defined, the command is ignored.

An application program is only permitted to use the following commands: Bell, Define Field, Erase Field, Record Separator, Set Transmit State. The following commands may also be used, subject to constraints: Define Window (window id not specified), Erase Window (window id not specified), all cursor positioning commands (position within logical device bounds). The following commands are for internal use only and may not be
used under any circumstances: Define Window (window id specified), Remove Window, Erase Window (window id specified), Set Window, Window Precedence. All other commands may be used, but there is no guarantee that the application will correctly be constrained to the limits of its logical device.

3.2 Command Descriptions

Graphic Character

Causes the character to be displayed according to the graphic rendition in effect at the cursor location and advances the cursor to the next character position. This advancing may possibly cause scrolling.

BEL - Sound Bell
   <BEL>
   Sounds an audible alarm at the terminal.

BS - Backspace
   <BS>
   Moves the cursor to the previous character position; if the cursor is at the left margin, no action occurs.

HT - Horizontal Tab
   <HT>
   Moves the cursor to the next field.

LF - Line Feed
   <LF>
   Moves the cursor down to the next line in the current column and may possibly cause scrolling the screen.

FF - Form Feed
   <FF>
   Clears the screen and moves the cursor to the first unprotected character position. In Forms Mode, only unprotected areas of the screen are erased.

CR - Carriage Return
   <CR>
   Moves the cursor to the left margin in the current line.

RS - Record Separator
   <RS>
   Used to indicate end of a formatted buffer.

US - Unit Separator
   <US>
   Used to indicate end of subset of formatted buffer.

IND - Index
   <ESC> D
   Same as LF.
NEL - Next Line
  <ESC> E  
  Same as CR followed by LF.

RI - Reverse Index
  <ESC> M  
  Moves the cursor up to the previous line in the current 
  column, possibly scrolling the screen.

STS - Set Transmit State
  <ESC> S  
  Indicates that the currently selected window is to be 
  enabled for input. All unguarded fields are made enterable 
  and a data message will be sent when a function key is 
  pressed.

APC - Application Program Command
  <ESC> Pn <ESC> \  
  Generated when a function key is pressed. The parameter is 
  the function key number (0 - n) which must not be omitted. 
  Function key zero is the "ENTER" key.

RIS - Reset to Initial State
  <ESC> c  
  Resets the terminal to its initial state. The screen is 
  cleared, the cursor is positioned in the upper left corner.

REF - Refresh Screen
  <ESC> ?  
  Retransmits the current screen contents to the terminal. 
  Its main uses are to recover from unsolicited messages or 
  line noise which have corrupted the screen contents, or to 
  update the terminal when in Deferred Display Mode.

ICH - Insert Character
  <ESC> [ Pn @  
  Makes room for a character by shifting the rest of the field 
  one character position to the right (and down). Characters 
  shifted past the end of the field are lost. The cursor is 
  left at the first inserted character position (i.e. not 
  moved).

CUU - Cursor Up
  <ESC> [ Pn A  
  Moves the cursor to the previous line in the current 
  column, but not past the top margin.

CUD - Cursor Down
  <ESC> [ Pn B  
  Moves the cursor to the next line in the current column, but 
  not past the bottom margin.

CUF - Cursor Forward
  <ESC> [ Pn C  
  Moves the cursor to the next character position, but not 
  past the right margin.
CUB - Cursor Backward
<ESC> [ Pn D
Moves the cursor to the previous character position, but not past the left margin.

CNL - Cursor Next Line
<ESC> [ Pn E
Moves the cursor to the left margin of the next line, but not past the bottom margin.

CPL - Cursor Previous Line
<ESC> [ Pn F
Moves the cursor to the left margin of the previous line, but not past the top margin.

CUP - Cursor Position
<ESC> [ Pn ; Pn H
Moves the cursor to the specified position. The first parameter is the row number, the second parameter is the column number. If both parameters are omitted, the semi-colon may be omitted as well.

CHT - Cursor Horizontal Tab
<ESC> [ Pn I
Moves the cursor to the next field.

ED - Erase Display
<ESC> [ Ps J
Erases the screen according to the parameter:
0 - Erase from the cursor to the end of the screen
1 - Erase from the beginning of the screen to the cursor
2 - Erase the entire screen
The cursor is not moved. Only unprotected fields of the screen are erased.

EL - Erase Line
<ESC> [ Ps K
Erases the current line according to the parameter:
0 - Erase from the cursor to the end of the line
1 - Erase from the beginning of the line to the cursor
2 - Erase the entire line
The cursor is not moved. Only unprotected fields of the screen are erased.

IL - Insert Line
<ESC> [ Pn L
Makes room for Pn line(s) in each unprotected field on the line by shifting down the rest of the field Pn line(s) ([width * Pn] number of characters). Characters shifted past the end of the field are lost. The cursor is not moved.

DL - Delete Line
<ESC> [ Pn M
Deletes Pn line(s) in each unprotected field by moving the rest of the field up one line ([Pn * width] number of characters).
EF - Erase Field
<ESC> [ Ps N
Erases the current field according to the parameter:
0 - Erase from the cursor to the end of the field
1 - Erase from the beginning of the field to the cursor
2 - Erase the entire field
The cursor is not moved.

DCH - Delete Character
<ESC> [ Pn P
Deletes the current character by shifting the rest of the field one character position to the left.

CPR - Cursor Position Report
<ESC> [ Pn ; Pn R
Along with the APC command this is returned at the beginning of each buffer obtained through reading data from the VT. The first parameter is the current row, the second parameter is the current column.

NP - Next Page
<ESC> [ Pn U
Same as FF.

PP - Previous Page
<ESC> [ Pn V
Same as FF.

ECH - Erase Character
<ESC> [ Pn X
Erases the current character (the character is NOT deleted). The cursor is not moved. Only a single field is affected.

CBT - Cursor Backward Tab
<ESC> [ Pn Z
Moves the cursor to the previous field.

HPA - Horizontal Position Absolute
<ESC> [ Pn
Moves the cursor to the specified column in the current line.

HFR - Horizontal Position Relative
<ESC> [ Pn a
Same as CUF.

VPA - Vertical Position Absolute
<ESC> [ Pn d
Moves the cursor to the specified line in the current column.

VPR - Vertical Position Relative
<ESC> [ Pn e
Same as CUD.
HVP - Horizontal and Vertical Position
<ESC> [ Pn ; Pn f
Same as CUP.

SM - Set Mode
<ESC> [ Ps h (standard modes)
Sets the indicated modes. Only Insert mode (IRM) is currently supported.

MC - Media Copy
<ESC> [Ps i
Controls the transfer of data between the device and an auxiliary input/output device:
0 - Print Screen

RM - Reset Mode
<ESC> [ Ps l (standard modes)
Resets the indicated mode.

WP - Window Precedence
<ESC> [ Pn ... p
Sets the precedence of the specified windows. Each window is in turn placed on top of all other existing windows. Thus, the last window specified will ultimately be the top-most and all specified windows will be on top of any unspecified windows.

RW - Remove Window
<ESC> [ Pn r
Removes the specified window. If the window id is omitted, the currently selected window is used.

SW - Select Window
<ESC> [ Pn s
Selects the specified window.

EW - Erase Window
<ESC> [ Pn u
Removes all windows and fields from the specified window. If the window id is omitted, the currently selected window is used. If window 0 is specified by Pn or if no window is specified, then the currently selected window is erased.

DW - Define Window
<ESC> [ Pn ; Pn ; Pn ; Pn ; Pn ; Pn ; Pn ; Pn ; Pn ; Ps w
Defines a window within the currently selected window. The first parameter is the window id, the second and third parameters are the row and column within the selected window for this window to be displayed, the fourth and fifth parameters are the display width and depth, the sixth and seventh parameters are the offsets of the first displayed row and column from the actual first row and column, the eighth and ninth parameters are the actual width and depth, and the tenth parameter is the window attributes. The attributes are:
0 - Normal (reset existing attributes)
1 - Bright or Bold
2 - Dim
The specified attributes are in effect from the cursor position to the end of the current line, whichever comes first. Note that the specified attributes are IN ADDITION to the currently existing attributes unless Normal is specified.

`DF - Define Field`  
`<ESC> [ Pn ; Pn ; Pn ; Pn ; Ps ; Ps ; x`  
Defines a field within the currently selected window. The first and second parameters are the row and column within the selected window for the field to be displayed, the third and fourth parameters are the field width and depth, the fifth parameter is the "guarded" flag which is 1 if the field is guarded and 0 or omitted if the field is enterable, and the sixth parameter is the field attributes as per Define Window. The data to be displayed in the field must immediately follow the Define Field command in the same buffer (see PUTVTI, below).

### 3.3 Input-Output Routines

Four routines are provided for direct Virtual Terminal input and output. The calling sequences and parameter definitions follow.

**INITVT**

```call "INITVT" USING RCODE.```
Outputs

RCODE - character - Return code.
Possible return codes:
OK - good return
INVTIMD - already in VTI mode
probable cause: application has already called INITVT or application did not call TERMVT.
solution: none needed.

This routine performs all necessary initialization in preparation for using the Virtual Terminal. Specifically, it initiates Form Processor Bypass mode wherein the Form Processor no longer interprets Virtual Terminal messages but simply passes them back to the application.

GETVTI

CALL "GETVTI" USING BUFFER, MAX-LEN, ACT-LEN, RCODE.

Inputs

MAX-LEN - PIC S9(5) COMP - maximum length to read.

Outputs

BUFFER - PIC X(N) - data read from terminal.
ACT-LEN - PIC S9(5) COMP - length of data read.
RCODE - character - return code.
Possible return codes:
OK - good return.
NINVTIMD - not in VTI mode.
probable cause: application did not call INITVT or application already called TERMVT.
solution: application must call INITVT before GETVTI.
OVRFLW - length of data buffer larger than maximum length given in GETVTI call.
probable cause: application did not provide large enough buffer for data.
solution: application provide a larger buffer and recall GETVTI.

This routine performs a read from the Virtual Terminal.

In Forms Mode, the returned buffer consists of a Set Window command followed by Define Field commands for each field in the window which has been modified since the last read. This is followed by additional Set Window and Define Field commands for nested windows. Finally, a Cursor Position Report command giving the cursor position when the terminating function key was pressed and an Application Program Command command specifying which function key was pressed terminate the buffer.
If not in Forms Mode, the returned buffer consists of all the printable characters entered followed (if in Control Transfer Mode) by the control sequence which terminated the input.

If an inquiry (e.g. DSR) was performed prior to reading, the returned buffer contains only the reply regardless of Forms Mode and Control Transfer Mode.

**PUTVTI**

CALL "PUTVTI" USING BUFFER, ACT-LEN, RCODE.

**Inputs**

- BUFFER - PIC X(n) - Data to be written.
- ACT-LEN - PIC S9(5) COMP - Length of data to write.

**Outputs**

- RCODE - character - Return code
  - Possible return codes:
    - OK - good return.
    - NINVTIMD - not in VTI mode.
      - probable cause: application did not call VTI or application has already called TERMVT.
      - solution: application must call INITVT before PUTVTI.

This routine performs a write to the Virtual Terminal. This routine may be called multiple times to send multiple buffers of commands to the Virtual Terminal. In any case, the final buffer must end with a Record Separator command in order to process the preceding commands. See above for restrictions on the commands which may be contained in BUFFER.

**TERMVT**

CALL "TERMVT" USING RCODE.

**Outputs**

- RCODE - character - Return code
  - Possible return codes
    - OK - good return.
    - NINVTIMD - not in VTI mode.
      - probable cause: application did not call VTI or application has already called TERMVT.
      - solution: none needed.

This routine terminates the Virtual Terminal. It terminates Form Processor Bypass mode, causing the Form Processor to once again interpret Virtual Terminal messages and refreshes the screen to eliminate any disruption caused by the Virtual Terminal output.
SECTION 4
TERMINAL IMPLEMENTATION

4.1 Adding New Terminals

The translation from Virtual Terminal commands to commands for a specific terminal (and vice versa) is performed by a program known as a device driver. Adding a new terminal is accomplished simply by writing a device driver for the terminal and making it known to the system. Since all device drivers perform the same basic functions, most of the necessary routines are already written, and only a few will need to be written for a particular terminal. (Since the currently existing device drivers are written in the C programming language, a large number of utility and support functions exist for device drivers written in C. For this reason, this discussion will focus on device drivers which are being written in C; this should not be interpreted as meaning that device drivers could not be written in another language, only that doing so would be significantly more work.)

Two different types of device drivers will be discussed. First, we will consider a general purpose device driver which can support any type of terminal. Second, we will consider the special case of a terminal which does not support forms and does not perform local echoing (or allows local echoing to be disabled). It should be noted that all of the currently supported terminals fall into this category.

4.2 General Purpose Device Driver

A general purpose device driver must contain four routines: INTVT, GETVT, PUTVT, and TRMVT.

GETVT and PUTVT (which have already been discussed) accept Virtual Terminal commands and translate them into commands for a particular device and vice versa. All Virtual Terminal commands must be supported, even if this requires simulation in software. (It should be noted, however, that it is not necessary to allow all Virtual Terminal commands to be entered by the user. It is up to the implementor to determine a reasonable subset to be supported, but the subset should at least include the cursor movements, forward and backward tab, 20 function keys including the enter key, screen refresh, and delete character.)

The only allowable exceptions to this are the Bell and Media Copy, and Define Window and Define Field. The Bell and Media Copy commands must be recognized correctly, but need not produce any effect if the terminal does not have an audible alarm or printer. Visual attributes should be simulated as well as possible; some guidelines follow.
If the terminal only has two brightness levels, BOLD should be supported with DIM being the same as NORMAL; if only a single brightness level exists, BOLD, DIM, and NORMAL should all be the same. If the terminal has only a single blink speed, it should be used for both FAST BLINK and SLOW BLINK; if blink is not supported, FAST BLINK and SLOW BLINK may be ignored. If only a single highlight is supported (e.g. reverse video, underline, etc.), it should be used for both REVERSE and UNDERSCORE; if no highlights are supported, both REVERSE and UNDERSCORE should be simulated by a software underscore (blanks in the field are replaced by underscores). CONCEALED may be simulated by blanking the field on the screen as necessary.

The Window Manager portion of the Device Driver processes the Set Transmit State, Window Precedence, Define Window, Remove Window, Select Window, Erase Window, and Define Field commands. It is intended to be portable and used in all Device Drivers without change. Thus, these commands do not need to be supported by new Device Drivers. (If, however, the terminal in question supports windowing, it may be desirable to implement these commands as part of the device-specific part of the driver.)

INTVT and TRMVT (which have also been discussed previously) are called once at startup and termination respectively to initialize the device driver and perform cleanup. The initialization usually consists of opening a communication channel to the terminal and calling PUTVTI with a Reset to Initial State command to reset the terminal. The cleanup usually consists of sending commands to the terminal to return it to the normal state of terminals on the system (such as setting normal modes or tab stops) and clear the screen, and closing the communication channel to the terminal.

4.3 Special Case Device Driver

If a terminal supports forms, writing a general purpose device driver for it should not be very difficult. However, a terminal which does not support forms requires most functions to be simulated in software, requiring a very complex device driver. Since all of the terminals which are currently supported fall into this category, routines exist which make writing a device driver for this type of terminal much easier. (However, it should be noted that supporting this type of terminal requires being able to perform character at a time I/O without echo. This is not possible on some computer systems, making support impossible.) These support routines are documented in Appendix C; many unsubstantiated references to them will be made in the following text.

Supporting a new terminal of this type requires writing six routines: TRMINI, TRMCHK, TRMGET, TRMPUT, TRMFLS, and TRMEND. TRMINI is called once to establish communication with the
terminal. This is usually done with a call to TBOPEN. The calling sequence for TRMINI is:

```
trmini(tname)
```

TNAME is the terminal name passed in to INITVT converted to a C string.

TRMCHK is called to check for terminal input that must be processed. The calling sequence for TRMCHK is:

```
trmchk()
```

It returns TRUE or FALSE depending on whether there are keyboard characters to be processed or not.

TRMGET and TRMPUT are called to get commands from and put commands to the terminal. TRMGET usually calls TRMPUT as well in order to echo the user input. The calling sequences for TRMGET and TRMPUT are:

```
trmget(cmd)
trmput(cmd)
```

CMD is a command in internal form.

TRMFLS is called to insure that all output has been displayed (any buffers should be flushed). The calling sequence for TRMFLS is:

```
trmfls()
```

TRMEND is called once to terminate communications with the terminal. The calling sequence for TRNEND is:

```
trmend()
```
APPENDIX A

VIRTUAL TERMINAL CHARACTER SET

<table>
<thead>
<tr>
<th>Char</th>
<th>ASCII Hex</th>
<th>ASCII Oct</th>
<th>ASCII Dec</th>
<th>EBCDIC Hex</th>
<th>EBCDIC Oct</th>
<th>EBCDIC Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;NUL&gt;</td>
<td>00 000</td>
<td>0 00 000</td>
<td>0 00 000</td>
<td>0 00 000</td>
<td>0 00 000</td>
<td>&lt;SP&gt; 20 040 32 40 100 64</td>
</tr>
<tr>
<td>&lt;SOH&gt;</td>
<td>01 001</td>
<td>01 001 1</td>
<td>01 001 1</td>
<td>01 001 1</td>
<td>01 001 1</td>
<td>&lt;ETX&gt; 03 003 3</td>
</tr>
<tr>
<td>&lt;STX&gt;</td>
<td>02 002</td>
<td>02 002 2</td>
<td>02 002 2</td>
<td>02 002 2</td>
<td>02 002 2</td>
<td>&lt;EOT&gt; 04 004 4</td>
</tr>
<tr>
<td>&lt;ETX&gt;</td>
<td>03 003</td>
<td>03 003 3</td>
<td>03 003 3</td>
<td>03 003 3</td>
<td>03 003 3</td>
<td></td>
</tr>
<tr>
<td>&lt;SOH&gt;</td>
<td>05 005</td>
<td>05 005 5</td>
<td>05 005 5</td>
<td>05 005 5</td>
<td>05 005 5</td>
<td>&lt;ACK&gt; 06 006 6</td>
</tr>
<tr>
<td>&lt;BEL&gt;</td>
<td>07 007</td>
<td>07 007 7</td>
<td>07 007 7</td>
<td>07 007 7</td>
<td>07 007 7</td>
<td></td>
</tr>
<tr>
<td>&lt;BS&gt;</td>
<td>08 010</td>
<td>08 010 8</td>
<td>08 010 8</td>
<td>08 010 8</td>
<td>08 010 8</td>
<td></td>
</tr>
<tr>
<td>&lt;HT&gt;</td>
<td>09 011</td>
<td>09 011 9</td>
<td>09 011 9</td>
<td>09 011 9</td>
<td>09 011 9</td>
<td></td>
</tr>
<tr>
<td>&lt;LF&gt;</td>
<td>0A 012</td>
<td>0A 012 10</td>
<td>0A 012 10</td>
<td>0A 012 10</td>
<td>0A 012 10</td>
<td></td>
</tr>
<tr>
<td>&lt;VT&gt;</td>
<td>0B 013</td>
<td>0B 013 11</td>
<td>0B 013 11</td>
<td>0B 013 11</td>
<td>0B 013 11</td>
<td></td>
</tr>
<tr>
<td>&lt;FF&gt;</td>
<td>0C 014</td>
<td>0C 014 12</td>
<td>0C 014 12</td>
<td>0C 014 12</td>
<td>0C 014 12</td>
<td></td>
</tr>
<tr>
<td>&lt;CR&gt;</td>
<td>0D 015</td>
<td>0D 015 13</td>
<td>0D 015 13</td>
<td>0D 015 13</td>
<td>0D 015 13</td>
<td></td>
</tr>
<tr>
<td>&lt;SO&gt;</td>
<td>0E 016</td>
<td>0E 016 14</td>
<td>0E 016 14</td>
<td>0E 016 14</td>
<td>0E 016 14</td>
<td></td>
</tr>
<tr>
<td>&lt;SI&gt;</td>
<td>0F 017</td>
<td>0F 017 15</td>
<td>0F 017 15</td>
<td>0F 017 15</td>
<td>0F 017 15</td>
<td></td>
</tr>
<tr>
<td>&lt;DLE&gt;</td>
<td>10 020</td>
<td>10 020 16</td>
<td>10 020 16</td>
<td>10 020 16</td>
<td>10 020 16</td>
<td></td>
</tr>
<tr>
<td>&lt;DC1&gt;</td>
<td>11 021</td>
<td>11 021 17</td>
<td>11 021 17</td>
<td>11 021 17</td>
<td>11 021 17</td>
<td></td>
</tr>
<tr>
<td>&lt;DC2&gt;</td>
<td>12 022</td>
<td>12 022 18</td>
<td>12 022 18</td>
<td>12 022 18</td>
<td>12 022 18</td>
<td></td>
</tr>
<tr>
<td>&lt;DC3&gt;</td>
<td>13 023</td>
<td>13 023 19</td>
<td>13 023 19</td>
<td>13 023 19</td>
<td>13 023 19</td>
<td></td>
</tr>
<tr>
<td>&lt;DC4&gt;</td>
<td>14 024</td>
<td>14 024 20</td>
<td>14 024 20</td>
<td>14 024 20</td>
<td>14 024 20</td>
<td></td>
</tr>
<tr>
<td>&lt;NAK&gt;</td>
<td>15 025</td>
<td>15 025 21</td>
<td>15 025 21</td>
<td>15 025 21</td>
<td>15 025 21</td>
<td></td>
</tr>
<tr>
<td>&lt;SYN&gt;</td>
<td>16 026</td>
<td>16 026 22</td>
<td>16 026 22</td>
<td>16 026 22</td>
<td>16 026 22</td>
<td></td>
</tr>
<tr>
<td>&lt;ETB&gt;</td>
<td>17 027</td>
<td>17 027 23</td>
<td>17 027 23</td>
<td>17 027 23</td>
<td>17 027 23</td>
<td></td>
</tr>
<tr>
<td>&lt;CAN&gt;</td>
<td>18 030</td>
<td>18 030 24</td>
<td>18 030 24</td>
<td>18 030 24</td>
<td>18 030 24</td>
<td></td>
</tr>
<tr>
<td>&lt;SUB&gt;</td>
<td>1A 032</td>
<td>1A 032 26</td>
<td>1A 032 26</td>
<td>1A 032 26</td>
<td>1A 032 26</td>
<td></td>
</tr>
<tr>
<td>&lt;ESC&gt;</td>
<td>1B 033</td>
<td>1B 033 27</td>
<td>1B 033 27</td>
<td>1B 033 27</td>
<td>1B 033 27</td>
<td></td>
</tr>
<tr>
<td>&lt;FS&gt;</td>
<td>1C 034</td>
<td>1C 034 28</td>
<td>1C 034 28</td>
<td>1C 034 28</td>
<td>1C 034 28</td>
<td></td>
</tr>
<tr>
<td>&lt;GS&gt;</td>
<td>1D 035</td>
<td>1D 035 29</td>
<td>1D 035 29</td>
<td>1D 035 29</td>
<td>1D 035 29</td>
<td></td>
</tr>
<tr>
<td>&lt;RS&gt;</td>
<td>1E 036</td>
<td>1E 036 30</td>
<td>1E 036 30</td>
<td>1E 036 30</td>
<td>1E 036 30</td>
<td></td>
</tr>
<tr>
<td>&lt;US&gt;</td>
<td>1F 037</td>
<td>1F 037 31</td>
<td>1F 037 31</td>
<td>1F 037 31</td>
<td>1F 037 31</td>
<td></td>
</tr>
</tbody>
</table>

A-1
<table>
<thead>
<tr>
<th>Char</th>
<th>ASCII</th>
<th>EBCDIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Char</td>
<td>Hex</td>
</tr>
<tr>
<td>@</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>A</td>
<td>41</td>
<td>101</td>
</tr>
<tr>
<td>B</td>
<td>42</td>
<td>102</td>
</tr>
<tr>
<td>C</td>
<td>43</td>
<td>103</td>
</tr>
<tr>
<td>D</td>
<td>44</td>
<td>104</td>
</tr>
<tr>
<td>E</td>
<td>45</td>
<td>105</td>
</tr>
<tr>
<td>F</td>
<td>46</td>
<td>106</td>
</tr>
<tr>
<td>G</td>
<td>47</td>
<td>107</td>
</tr>
<tr>
<td>H</td>
<td>48</td>
<td>110</td>
</tr>
<tr>
<td>I</td>
<td>49</td>
<td>111</td>
</tr>
<tr>
<td>J</td>
<td>4A</td>
<td>112</td>
</tr>
<tr>
<td>K</td>
<td>4B</td>
<td>113</td>
</tr>
<tr>
<td>L</td>
<td>4C</td>
<td>114</td>
</tr>
<tr>
<td>M</td>
<td>4D</td>
<td>115</td>
</tr>
<tr>
<td>N</td>
<td>4E</td>
<td>116</td>
</tr>
<tr>
<td>O</td>
<td>4F</td>
<td>117</td>
</tr>
<tr>
<td>P</td>
<td>50</td>
<td>120</td>
</tr>
<tr>
<td>Q</td>
<td>51</td>
<td>121</td>
</tr>
<tr>
<td>R</td>
<td>52</td>
<td>122</td>
</tr>
<tr>
<td>S</td>
<td>53</td>
<td>123</td>
</tr>
<tr>
<td>T</td>
<td>54</td>
<td>124</td>
</tr>
<tr>
<td>U</td>
<td>55</td>
<td>125</td>
</tr>
<tr>
<td>V</td>
<td>56</td>
<td>126</td>
</tr>
<tr>
<td>W</td>
<td>57</td>
<td>127</td>
</tr>
<tr>
<td>X</td>
<td>58</td>
<td>130</td>
</tr>
<tr>
<td>Y</td>
<td>59</td>
<td>131</td>
</tr>
<tr>
<td>Z</td>
<td>5A</td>
<td>132</td>
</tr>
<tr>
<td>[</td>
<td>5B</td>
<td>133</td>
</tr>
<tr>
<td>\</td>
<td>5C</td>
<td>134</td>
</tr>
<tr>
<td>]</td>
<td>5D</td>
<td>135</td>
</tr>
<tr>
<td>_</td>
<td>5E</td>
<td>136</td>
</tr>
</tbody>
</table>

DEL 7F 177 127 07 007 7
APPENDIX B

COMMAND REFERENCE

For each function the key sequence, internal function identifier, command abbreviation, and command description are given. Tables of selective parameters follow the function definitions.

Function Definitions

Control Characters

Ctrl-G 0007 BEL Sound Bell
Ctrl-H 0008 BS Backspace
Ctrl-I 0009 HT Forward Tab
Ctrl-J 0010 LF Line Feed / New Line
Ctrl-L 0012 FF Form Feed
Ctrl-M 0013 CR Carriage Return
Ctrl-[ ESC Character Set Extension (see following)
Ctrl-\ 0030 RS Record Separator
<ESC>D 1004 IND Index
<ESC>E 1005 NEL Next Line
<ESC>M 1013 RI Reverse Index
<ESC>S 1019 STS Set Transmit State
<ESC>[ CSI Control Sequence Introducer (see following)
<ESC>\ 1028 ST String Terminator
<ESC>– 1031 APC Application Program Command (function keys)
<ESC>? 4000 REF Refresh Screen (private)

Control Sequences (<CSI> ...)

Pn @ 3000 ICH Insert Character
Pn A 3001 CUU Cursor Up
Pn B 3002 CUD Cursor Down
Pn C 3003 CUF Cursor Forward
Pn D 3004 CUB Cursor Backward
Pn E 3005 CNL Cursor Next Line
Pn F 3006 CPL Cursor Preceding Line
Pn;Pn H 3008 CUP Cursor Position
Pn I 3009 CHT Cursor Horizontal Tab
Ps J 3010 ED Erase Display
Ps K 3011 EL Erase Line
Ps L 3012 IL Insert Line
Ps M 3013 DL Delete Line
Ps N 3014 EF Erase Field
Pn P 3016 DCH Delete Character
Pn;Pn R 3018 CPR Cursor Position Report
Pn U 3021 NP Next Page
Pn V 3022 PP Preceding Page
Pn X 3024 ECH Erase Character
Pn Z 3026 CBT Cursor Backward Tab
Pn . 3032 HPA Horizontal Position Absolute
Pn a 3033 HPR Horizontal Position Relative
Characters to be sent to terminal without interpretation.

Application Program Commands (<APC> ... <ST>)

Decimal representation of function key number.

Selective Parameter Tables

Erase Parameters

0 - Current Position to End of Area (inclusive)
1 - Beginning of Area to Current Position (inclusive)
2 - Entire Area

Media Copy Parameters

0 - Print Screen

Attribute Parameters

0 - Default
1 - Bright
2 - Dim
4 - Underscore
5 - Slow Blink
6 - Fast Blink
7 - Reverse
8 - Concealed
30 - Black Display
31 - Red Display
32 - Green Display
33 - Yellow Display
34 - Blue Display
35 - Magenta Display
36 - Cyan Display
37 - White Display
40 - Black Background
41 - Red Background
42 - Green Background
43 - Yellow Background
44 - Blue Background
45 - Magenta Background
46 - Cyan Background
47 - White Background
Device Status Request Parameters

6 - Report Current Position (via CPR)
APPENDIX C

DEVICE DRIVER SUPPORT ROUTINES

BITS.H
/* NAME
bits.h - include file for bit manipulation routines
 Written: 25-MAY-1983 09:47:12 - SCJONES
 Revised: 7-AUG-1985 11:19:09 - JONES
*/

/* NAME
 tbit - Test BIT (macro)
 */
/* SYNOPSIS
 #include <bits.h>

 bool tbit(bits, nbits, bit)
 BITTYP bits[];
 int nbits, bit;

 /* DESCRIPTION
 tbit returns TRUE or FALSE depending on whether bit bit is
 set or cleared in bits. nbits is the number of bits in
 bits.
 */

/* NAME
 sbit - Set BIT (macro)
 */
/* SYNOPSIS
 #include <bits.h>

 void sbit(bits, nbits, bit)
 BITTYP bits[];
 int nbits, bit;

 /* DESCRIPTION
 sets bit <bit> in the bit string <bits>. <nbits> is the
 length of <bits>.
 */

/* NAME
 cbit - Clear BIT (macro)
 */
/* SYNOPSIS
 #include <bits.h>

 void cbit(bits, nbits, bit)
 BITTYP bits[];
 int nbits, bit;
* DESCRIPTION
* Clears bit <bit> in the bit string <bits>. <nbits> is the
* length of <bits>.
*/

/* NAME
* sabit - Set All BITs (macro)
*
* SYNOPSIS
* #include <bits.h>
* void sabit(bits, nbits)
* BITYP bits[];
* int nbits;
*
* DESCRIPTION
* sets the bit string <bits>. <nbits> is the length of
* <bits>.
*/

/* NAME
* cabit - Clear All BITs (macro)
*
* SYNOPSIS
* #include <bits.h>
* void cabit(bits, nbits)
* BITYP bits[];
* int nbits;
*
* DESCRIPTION
* clears the bit string <bits>. <nbits> is the length of
* <bits>.
*/

DOSCRN.C
/* NAME
* DOSCRN - DO command to internal SCReeN
* Written: 20-DEC-1984 12:45:14 - SCWEHRMAN
* Revised: 2-APR-1986 09:13:25 - WEHRMAN
*
* SYNOPSIS
* int doscrn(cmd)
* struct command *cmd;
*
* DESCRIPTION
* Executes cmd on the internal screen and fixes up its
* parameters. Returns -1 for errors, 0 for no action, 1 for
* normal command, and 2 for move the cursor and retry.
*/

/* NAME
* hscr - Horizontal SCRoll
*/
/* NAME
*  vscr - Vertical SCRoll
*/
/* NAME
*  erase - ERASE part of screen
*/

FFBCA.C
/* NAME
*  ffbca - Find First Bit Clear After
*  Written: 11-AUG-1983 11:56:12
*  Revised: 12-NOV-1984 14:15:25 - SCJONES
*  SYNOPSIS
*  #include <bits.h>
*  int ffbca(bitstr, nbits, after)
*    BITTYP bitstr[];
*    int nbits, after;
*  DESCRIPTION
*  ffbca returns the number of the first clear bit following
*  bit after in the bit string bitstr. If after is < 0, the
*  number of the first clear bit is returned. nbits is the
*  number of bits in bitstr. If all remaining bits are set,
*  -1 is returned (A VAX RTL routine is called if the symbol
*  vms is defined).
*/

FFBCB.C
/* NAME
*  ffbcb - Find First Bit Clear Before
*  Written: 11-AUG-1983 11:57:42
*  Revised: 12-NOV-1984 14:17:42 - SCJONES
*  SYNOPSIS
*  #include <bits.h>
*  int ffbcb(bitstr, nbits, before)
*    BITTYP bitstr[];
*    int nbits, before;
*  DESCRIPTION
*  ffbcb returns the number of the first clear bit before bit
*  before going backwards in the bit string bitstr. If
*  before is < 0, the number of the first clear bit from the
*  end of the string is returned. nbits is the number of
*  bits in bitstr. If all remaining bits are clear, -1 is
*  returned.
*/
FFBD.C
/* NAME
 *   ffbd - Find First Bit Different
 *   Written: 11-AUG-1983 12:02:55
 *   Revised: 12-NOV-1984 14:20:37 - SCJONES
 */

SYNOPSIS
*   #include <bits.h>
*   int ffbd(bitstr1, bitstr2, nbits)
*     BITTYP bitstr1[], bitstr2[];
*     int nbits;

DESCRIPTION
*   ffbd returns the number of the first bit which is
different in the bit strings bitstr1 and bitstr2. nbits
is the number of bits in the strings. If all remaining
bits are the same, -1 is returned.
*/

FFBDA.C
/* NAME
 *   ffbda - Find First Bit Different After
 *   Written: 11-AUG-1983 12:05:58
 */

SYNOPSIS
*   #include <bits.h>
*   int ffbda(bitstr1, bitstr2, nbits, after)
*     BITTYP bitstr1[], bitstr2[];
*     int nbits, after;

DESCRIPTION
*   ffbda returns the number of the first bit following bit
after which is different in the bit strings bitstr1 and
bitstr2. If after is < 0, the number of the first
difference bit is returned. nbits is the number of bits
in bitstr1 and bitstr2. If all remaining bits are the
same, -1 is returned.

FFBSA.C
/* NAME
 *   ffbsa - Find First Bit Set After
 *   Written: 11-AUG-1983 12:36:42
 */

SYNOPSIS
*   #include <bits.h>
*   int ffbsa(bitstr, nbits, after)
*     BITTYP bitstr[];
*     int nbits, after;
UM 620344300
30 September 1990

* DESCRIPTION
* ffbsa returns the number of the first set bit following
* bit after in the bit string bitstr. If after is < 0, the
* number of the first set bit is returned. nbits is the
* number of bits in bitstr. If all remaining bits are set,
* -1 is returned (A VAX RTL routine is called if the symbol
* vms is defined.).
*/

** FFBSB.C **

/* NAME
* ffbsb - Find First Bit Set Before
* Revised: 12-NOV-1984 14:25:44 - SCJONES
*
* SYNOPSIS
* #include <bits.h>
* int ffbsb(bitstr, nbits, before)
* BITTYP bitstr[];
* int nbits, before;
*
* DESCRIPTION
* ffbsb returns the number of the first set bit before bit
* before going backwards in the bit string bitstr. If
* before is < 0, the number of the first set bit from the
* end of the string is returned. nbits is the number of
* bits in bitstr. If all remaining bits are set, -1 is
* returned.
*/

** TERMIO.C **

/* NAME
* termio - TERMINal I/O package
* Written: 11-MAY-1983 15:54:05 - JONES
* Revised: 7-FEB-1986 11:40:04 - SDRCTEST
*
* DESCRIPTION
* This package provides immediate, character at a time i/o
* from a terminal (i.e. does not collect an edited line like
* stdio).
*
* For details on the supported functions, see the individual
* function descriptions.
*/

/* NAME
* topen - Terminal OPEN channel
*
* SYNOPSIS
* #include <termio.h>
* TERM *topen(device)
* char *device;
*
/* DESCRIPTION
   *  tgetc returns the next character typed at the specified
   *  terminal OR EOF.
   */

/* NAME
   *  tgetct - Terminal GET Character (Transparent)
   */
**DESCRIPTION**

tgetct returns the next character typed at the specified
terminal without processing special control characters.
Note that characters already in the type-ahead buffer may
have been subject to special processing.

```c
#include <termio.h>

void tgetct(c, term)
    char c;
    TERM *term;
```

**DESCRIPTION**
tgetct returns the specified character to the specified
terminal so that the next tgetc or tgetct call will return
it. Only a single push-back is allowed.

```c
#include <termio.h>

int tputc(c, term)
    char c;
    TERM *term;
```

**DESCRIPTION**
tputc outputs the specified character to the specified
terminal.

```c
#include <termio.h>

int tputc(c, term)
    char c;
    TERM *term;
```

**DESCRIPTION**
tputc outputs the specified character to the specified
terminal without processing special control characters.
/* NAME
* tflush - Terminal FLUSH buffer
* SYNOPSIS
* #include <termio.h>
* int tflush(term)
*   TERM *term;
* DESCRIPTION
* tflush empties the specified terminal's output buffer.
*/

/* NAME
* tflusht - Terminal FLUSH buffer (Transparent)
* SYNOPSIS
* #include <termio.h>
* int tflusht(term)
*   TERM *term;
* DESCRIPTION
* tflusht empties the specified terminal's output buffer
* without interpreting special control characters.
*/

/* NAME
* tclose - Terminal CLOSE
* SYNOPSIS
* #include <termio.h>
* int tclose(term)
*   TERM *term;
* DESCRIPTION
* tclose closes the specified terminal.
*/

/* NAME
* ttrans - Terminal set TRANSParent mode
* SYNOPSIS
* #include <termio.h>
* int ttrans(term)
*   TERM *term;
* DESCRIPTION
* ttrans places the terminal in transparent mode. In this
* mode, all special characters (ctrl-y, ctrl-c, ctrl-s,
* ct.l-q, ctrl-o, ctrl-r, and ctrl-t) are treated as data
* and returned by tgetc.
*/
/* NAME */
* tntrans - Terminal set Non-TRANSParent mode
* 
* SYNOPSIS
* #include <termio.h>
* int tntrans(term)
* TERM *term;
* 
* DESCRIPTION
* tntrans cancels transparent mode set by ttrans.
*/

/* NAME */
* tckeck - Terminal CHECK for input
* 
* SYNOPSIS
* #include <termio.h>
* int tcheck(term)
* TERM *term;
* 
* DESCRIPTION
* tcheck returns the number of characters in the type-ahead buffer.
*/

/* NAME */
* tpurge - Terminal PURGE typeahead
* 
* SYNOPSIS
* #include <termio.h>
* int tpurge(term)
* TERM *term;
* 
* DESCRIPTION
* tpurge removes all characters from the typeahead buffer.
*/

/* NAME */
* tgetnm - Terminal GET device NaMe
* 
* SYNOPSIS
* #include <termio.h>
* char *tgetnm(dev)
* char *dev;
* 
* DESCRIPTION
* returns the physical device name associated with the specified logical device name
*/
/* NAME
 * iopnd - internal - IO PeNDing
 * SYNOPSIS
 * static int iopnd(term)
 *   TERM *term;
 * DESCRIPTION
 * Checks for outstanding io and waits for it to clear
 * returns 1 io pending and cleared else returns 0
 */

/* NAME
 * intclose - INTernal CLOSE - exit handler
 * SYNOPSIS
 * static void intclose(reason, term)
 *   unsigned long reason;
 *   TERM *term;
 * DESCRIPTION
 * called at image exit to close open terminals
 */

TPUTNUM.C
/* NAME
 * tputnum - Terminal PUT NUMBER
 * Written: 3-JUN-1983 10:12:49
 * Revised: 9-NOV-1984 15:38:30 - SCJONES
 * SYNOPSIS
 * void tputnum(i, chan)
 *   int i;
 *   TERM *chan;
 * DESCRIPTION
 * Converts i to character form and writes it to the
 * specified terminal.
 */

TPUTS.C
/* NAME
 * tputs - Terminal PUT String
 * Written: 3-JUN-1983 10:14:03
 * Revised: 24-AUG-1983 09:43:27
 * SYNOPSIS
 * void tputs(s, chan)
 *   char *s;
 *   TERM *chan;
 * DESCRIPTION
 * Writes the specified string to the specified terminal.
 */
APPENDIX D

DEVICE DRIVER INCLUDE FILES

VTDEF.H

/* NAME */
* vtdfl.h - Virtual Terminal DEFINitions and data structures
* Written: 28-AUG-1986 08:58:52 - SCWEHRMAN
*
* DESCRIPTION
* Defines symbols, externals, etc. for the internal Virtual Terminal
*/

#ifndef POS

/* Standard (Public) mode definitions */

#define GATM 1 /* Guarded Area Transfer Mode */
#define KAM 2 /* Keyboard Action Mode */
#define CRM 3 /* Control Representation Mode */
#define IRM 4 /* Insertion-Replacement Mode:
(cleared if reformatting the screen set/cleared by some device drivers -
to insert/replace characters).
This is the only mode currently supported */

#define SRTM 5 /* Status Reporting Transfer Mode */
#define ERM 6 /* Erasure Mode */
#define VEM 7 /* Vertical Editing Mode */
#define HEM 10 /* Horizontal Editing Mode */
#define PUM 11 /* Position Unit Mode */
#define SRM 12 /* Send-Receive Mode */
#define FEAM 13 /* Format Effector Action Mode */
#define FETM 14 /* Format Effector Transfer Mode */
#define MATM 15 /* Multiple Area Transfer Mode */
#define TTM 16 /* Transfer Termination Mode */
#define SATM 17 /* Selected Area Transfer Mode */
#define TSM 18 /* Tabulation Stop Mode */
#define EBM 19 /* Editing Boundary Mode */
#define LNM 20 /* Line feed New line Mode */

#define GOODRET 0
#define FAILED 1

#define POS(r, c) ((c) + (r) * vt.dvce.dspsiz.width)
/* gives position in screen array */
#define ROW(pos) ((pos) / vt.dvce.dspsiz.width)
/* gives row pos = position in screen array */
#define COL(pos) ((pos) % vt.dvce.dspsiz.width)
/* gives col pos = position in screen array */

D-1
typedef struct {
    unsigned nfldmrk:1; /* Set if the beginning of new field */
    unsigned bright:1; /* Set if foreground color to be displayed bright */
    unsigned dim:1; /* Set if foreground color to be displayed dim */
    unsigned undrscor:1; /* Set if is to be underlined */
    unsigned slowblnk:1; /* Set if is to blink slow */
    unsigned fastblnk:1; /* Set if is to blink fast */
    unsigned reverse:1; /* Set if background & foreground colors switched */
    unsigned conceald:1; /* Set if contents not to be displayed on dev. */
    unsigned guard:1; /* Set if is to be protected from input from dev. */
    unsigned tabstp:1; /* Set if is to be tabbed to. */
    unsigned fgcolor:3; /* color for foreground */
    unsigned bgcolor:3; /* color for background */
} ATTR; /* color definitions
BLACK = 0
RED = 1
GREEN = 2
YELLOW = 3
BLUE = 4
MAGENTA = 5
CYAN = 6
WHITE = 7 */

typedef struct {
    unsigned rdpnd:1; /* Read pending */
    unsigned chginp:1; /* Changed input */
    unsigned chgfmt:1; /* Changed format or changed by output */
} FLAGS;

typedef struct {
    short row, col;
} POSITION;

typedef struct {
    short width, depth;
} SIZE;

typedef struct fldtyp {
    struct fldtyp *nxtfld, *prvfld; /* Next & previous fields */
    struct wndtyp *wndptr; /* Containing window */
    char *dap; /* Data buffer */
    int lngth; /* Length of data buffer */
}
typedef struct wndtyp
{
    struct wndtyp *nxtwnd, *prvwnd; /* Next & previous windows */
    struct wndtyp *fstwnd, *lstwnd; /* First & last child */
    struct wndtyp *parptr; /* Parent window */
    FLD *fstfld, *lstfld; /* First & last fields */
    int wndid; /* Name of this window */
    POSITION curcrs; /* current cursor position */
    POSITION offset; /* Scrolling offset */
    SIZE logsiz; /* Logical size */
    POSITION dsppos; /* Display position */
    SIZE dspsiz; /* Display size */
    ATTR attrib; /* Display attributes */
    FLAGS flags; /* Status flags */
} WND;

defined struct
{
    char *chrptr; /* Pointes to position in field's dap */
    /* if NULL then not field but window */
    WND *wndpnt; /* Pointes to window occupying this position */
    FLD *fldpnt; /* Pointes to field occupying this position if NULL then no field at his position */
    FLAGS *flags; /* Status flags for this position on screen */
    ATTR attr; /* Attribute for this position on screen */
    unsigned chg:1; /* Need to Changed on terminal flag */
} SCREEN;

defined struct devtyp
{
    unsigned bldflg:1; /* In process of building screen flag */
    unsigned inpflg:1; /* Input from terminal flag */
    unsigned insrt:1; /* In insert character (instead of overstrike) mode */
}
int maxpos; /* Maximum terminal screen position */
int termpos; /* Current physical terminal screen position */
int curpos; /* Current virtual terminal screen position */
int savpos; /* Save virtual terminal screen position */
WND *curwnd; /* Current window */
short funct; /* Last function key */
WND dvce; /* Device window - the top window of the device */
SCREEN *screen; /* One to one mapping of screen positions to internal data structure */

) DEVICE;

extern DEVICE vt; /* External definition of device structure */

FUNCTS.H
#endif
/* NAME
 * functs.h - FUNCTION definitions
 * Written: 24-AUG-1983 09:49:37
 * Revised: 17-JUL-1986 14:19:43 SCWEHRMAN
 *
 * DESCRIPTION
 * Defines the mnemonic virtual terminal command functions.
 * And defines structure for parsing vti message buffer.
 */
#ifndef FUNCFLAG
#define FUNCFLAG 1
#define BEL 7
#define BS 8
#define HT 9
#define LF 10
#define FF 12
#define CR 13
#define RS 30
#define US 31
#define IND 1004
#define NEL 1005
#define HTS 1008
#define RI 1013
#define DCS 1016
#define STS 1019
#define ST 1028
#define APC 1031
#define RIS 1035
#define ICH 3000
#define CUU 3001

D-4
#define CUD 3002
#define CUF 3003
#define CUB 3004
#define CNL 3005
#define CPL 3006
#define CUP 3008
#define CHT 3009
#define ED 3010
#define EL 3011
#define IL 3012
#define DL 3013
#define EF 3014
#define DCH 3016
#define CPR 3018
#define NP 3021
#define PP 3022
#define ECH 3024
#define CBT 3026
#define HPA 3032
#define HPR 3033
#define VPA 3036
#define VPR 3037
#define HVP 3038
#define TBC 3039
#define SM 3040
#define MC 3041
#define RM 3044
#define SGR 3045
#define DSR 3046
#define DAQ 3047
#define WP 3048
#define RW 3050
#define SW 3051
#define EW 3053
#define DW 3055
#define DF 3056
#define REF 4000
#define SPM 4040
#define RPM 4044

typedef struct command
{
    int funct, maxparm, nparm, parm[1];
} CMD;

#define BLDCMD(n) struct(int funct, maxparm, nparm, parm[n];}

/* Iternally used virtual term. commands */
extern struct command errcmd; /* error command */
extern struct command rstcmd; /* restore command */

#endif
APPENDIX E

SAMPLE DEVICE DRIVER (DEC VT-100 - MONOCROME)

/* NAME
* vt100 - vt100 terminal driver routines
* DESCRIPTION
* Device dependent modules for the DEC VT100 device driver.
*/

#include <stdtyp.h>
#include <string.h>
#include <ctype.h>
#include <termio.h>
#include <vtdef.h>
#include <functs.h>
#include "prntdef.h"

#define BUFSIZ 512
#define BUFNUM 2

static TERM *chan;

static void refresh();
static void movcur();
static void sndchr();
static void setatr();
void trmput();

/* NAME
* trmini - Terminal INITIALize
* SYNOPSIS
* bool trmini(tname)
* char *tname;
* DESCRIPTION
* Opens the terminal specified by tname and initializes it.
*/
bool trmini(tname)
char *tname;
{
    if ((chan = tbopen(tname, BUFSIZ, BUFNUM)) == NULL) return FAILED;
#ifdef WIDE
    vt.dvce.logsz.width = vt.dvce.dspsz.width = 132;
#else
    vt.dvce.logsz.width = vt.dvce.dspsz.width = 80;
#endif
#ifdef PRINTER
    prnini(tname);
#else
    return GOODRET;
#endif
)
/* NAME  
*     trmend - Terminal END  
* SYNOPSIS  
*     void trmend()  
*  
* DESCRIPTION  
*     Resets the currently open terminal and closes it.  
*/

void trmend()  
{    register int i;
    #ifdef PRINTER
        prnend();
    #endif
    for (i = 9; i < 80; i += 8) tputs("\33[8C\33H", chan);
    tputs("\33[m\33>\r", chan);
    tclose(chan);
}

/* NAME  
*     trmfls - Terminal FLush  
* SYNOPSIS  
*     void trmfls()  
*  
* DESCRIPTION  
*     Flush any terminal buffers.  
*/

void trmfls()  
{    movcur();
    tflush(chan);
}

/* NAME  
*     trmchk - Terminal CHeck  
* SYNOPSIS  
*     int trmchk()  
*  
* DESCRIPTION  
*     This module returns the number of characters in the  
*     type-ahead buffer.  
*/

int trmchk()  
{    return tcheck(chan);
}
/* NAME
*   trmget - Terminal GET
*
* SYNOPSIS
*   void trmget(cmd)
*       struct command *cmd;
*
* DESCRIPTION
*   Gets the next command from the terminal and converts it to
*   internal form.
*/

void trmget(cmd)
struct command *cmd;
{
    register char c;
    register int num, i;
    static BLDCMD(2) curcmd = { CUP, 2, 2, 0, 0 };

    if (isprint(c = tgetc(chan))) /* printable */
    {
        cmd->funct = 0;
        cmd->nparm = 1;
        cmd->parm[0] = c;
    }
    else if (c != '\33') /* control char */
    {
        if (c == '\22' || c == '\27') cmd->funct = REF;
        else if (c == '\177') cmd->funct = DCH;
        else cmd->funct = c;
        cmd->nparm = 0;
    }
    else switch (c = tgetc(chan))
    {
        case 'O': /* APC */
            cmd->funct = APC;
            cmd->nparm = 1;
            cmd->parm[0] = 1;
            switch (c = tgetc(chan))
            {
                case 'M': cmd->parm[0] = 0; break;
                case 'P': cmd->parm[0] = 1; break;
                case 'Q': cmd->parm[0] = 2; break;
                case 'R': cmd->parm[0] = 3; break;
                case 'S': cmd->parm[0] = 4; break;
                case 'W': cmd->parm[0] = 5; break;
                case 'X': cmd->parm[0] = 6; break;
                case 'Y': cmd->parm[0] = 7; break;
                case 'M': cmd->parm[0] = 8; break;
                case 'T': cmd->parm[0] = 9; break;
                case 'U': cmd->parm[0] = 10; break;
                case 'V': cmd->parm[0] = 11; break;
                case 'L': cmd->parm[0] = 12; break;
                case 'G': cmd->parm[0] = 13; break;
                case 'F': cmd->parm[0] = 14; break;
            }
```c

case 's': cmd->parm[0] = 15; break;
case 'p': cmd->parm[0] = 16; break;
case 'n': cmd->parm[0] = 17; break;
case 'A': cmd->funct = CUU; break;
case 'B': cmd->funct = CUD; break;
case 'C': cmd->funct = CUF; break;
case 'D': cmd->funct = CUB; break;
} break;

break;            /* back tab */
case 't':
    cmd->funct = CBT;
    cmd->nparm = 1;
    cmd->parm[0] = 1;
    break;

break;            /* erase end of field */
case 'l':
    cmd->funct = EF;
    cmd->nparm = 0;
    break;

break;            /* insert/overstrike mode */
case 'l':
    cmd->funct = vt.insrt ? RM : SM;
    cmd->nparm = 1;
    cmd->parm[0] = IRM;
    break;

/* function keys sequence */
num = 0;
while(isdigit(c = tgetc(chan))) num = 10 * num + c - '0';
if (c == 't')
    {
        cmd->funct = APC;
        cmd->nparm = 1;
        cmd->parm[0] = num;
    }
else
    {
        cmd->funct = BEL;
        cmd->nparm = 0;
    }
break;            /* function keys */

/* function keys */

case '1': case '2': case '3': case '4': case '5':
    case '6': case '7': case '8': case '9':
    cmd->parm[0] = c - '0';
goto pfcom;

```

goto pfcom;
  case 'r':
    cmd->parm[0] = 14;
    goto pfcom;
  case 't':
    cmd->parm[0] = 15;
    goto pfcom;
  case 'y':
    cmd->parm[0] = 16;
    goto pfcom;
  case 'u':
    cmd->parm[0] = 17;
    goto pfcom;
  case 'i':
    cmd->parm[0] = 18;
    goto pfcom;
  case 'o':
    cmd->parm[0] = 19;
    goto pfcom;
  case 'p':
    cmd->parm[0] = 20;
    goto pfcom;
  case '\r':
    cmd->parm[0] = 0;
    pfcom:
    cmd->funct = APC;
    cmd->nparm = 1;
    break;
  case '[':          /* control sequence */
    i = 0;
    do
    {
      num = 0;
      while(isdigit(c = tgetc(chan))) num = 10 * num + c - '0';
      cmd->parm[i++] = num;
    } while (c == ';');
    cmd->funct = 3000 + c - '0';
    cmd->nparm = i;
    break;
  default:
    cmd->funct = 1000 + c - '0';
    cmd->nparm = 0;
  }

  switch (dovt(cmd))
  {
    case -1:
      cmd->funct = BEL;
      cmd->nparm = 0;
    case 1:
      trmput(cmd);
      break;
  }
  trmfls();
/ * NAME 
 * trmput - Terminal PUT 
 * 
 * SYNOPSIS 
 * void trmput(cmd) 
 * struct command *cmd; 
 * 
 * DESCRIPTION 
 * Puts an internal format command to the terminal. 
 */

void trmput(cmd) 
    struct command *cmd; 
{
    int i, j, k, savepos; 
    char c;

    switch (cmd->funct) 
    { 
    case 0: 
        if (vt.insrt) refresh(); 
        else sndchr(cmd->parm[0]); 
        vt.curpos++; 
        if (vt.curpos < vt.maxpos) 
            { 
            vt.dvce.curcurs.row = ROW(vt.curpos) + 1; 
            vt.dvce.curcurs.col = COL(vt.curpos) + 1; 
            } 
        else 
            { 
            vt.curpos = 0; 
            vt.dvce.curcurs.row = vt.dvce.curcurs.col = 1; 
            } 
        break; 
    case BEL: 
        tputc('\7', chan); 
        break; 
    case BS: 
    case US: 
    case NEL: 
    case LF: 
    case IND: 
    case CR: 
    case HTS: 
    case RI: 
    case DCS: 
    case CPL: 
    case CUU: 
    case CNL: 
    case CUD: 
    case VPR: 
    case CUF: 
    case HPR: 
    case CUB: 
    case CUP: 

E-6
case HVP:
case CPR:
case HT:
case CHT:
case CBT:
case HPA:
case VPA:
    movcuro();
    break;
case RIS:
case REF:
#endif WIDE
    tputs("\33<\33[H\33[J\33[?3h\33[4;201\33[3g\33[m\33=\33[q", chan);
    /* removed \33[12h for Tek 410x firmware bug */
#else
    tputs("\33<\33[H\33[J\33[?31\33(4;201\33[3g\33[m\33=q",
    chan); /* removed \33[12h for Tek 410x firmware bug */
#endif
if (cmd->funct == RIS) break;

/****************************
\33< - Enter ANSI mode
\33[H - Home cursor
\33[J - Erase screen from cursor to end
\33[?3h - Set terminal to 132 columns
\33[?3l - Set terminal to 80 columns
\33[4;201 - Set terminal to: Insertion-replacement Mode
            Linefeed / new line Mode
\33[3g - Clear all tabs
\33[3m - Select graphic rendition (no attributes)
\33[= - Enter Alternate Keypad mode
\33[3q - Turn off Load LEDs (L1 off)
/*****************************/

case RS:
case FF:
case NP:
case PP:
case EW:
case DW:
case DF:
case ICH:
case IL:
case DL:
case EF:
case DCH:
case ECH:
    refresh();
    break;
case MC:
#endif PRINTER
    cmd->funct = REF;

E-7
prnput(cmd);
prnfls();
break;
#endif

case DSR:
case TBC:
case APC:
case SM:
case RM:
case SPM:
case RPM:
break;
}

/ * NAME
* refresh - REFRESH terminal
* SYNOPSIS
* static void refresh()
* DESCRIPTION
* Clears the terminal screen and rewrites it from the internal screen.
*/

static void refresh()
{
    register int k, j, i, maxpos, savpos;
savpos = vt.curpos;
vt.termpos = -1;
for (maxpos = vt.maxpos, i = 0; i < maxpos; i = j)
    {j = i + vt.dvce.dspsz.width;
for (k = i; k < j; k++)
    {
if (((vt.screen[k].fdpnt
     || (vt.screen[k].attr.undrscor
     || (vt.screen[k].attr.reverse ?
     vt.screen[k].attr.fgcolor :
     vt.screen[k].attr.bgcolor)
     >
     (vt.screen[k].attr.reverse ?
     vt.screen[k].attr.fgcolor :
     vt.screen[k].attr.bgcolor)
     && (vt.screen[k].flags &&
     vt.screen[k].flags->chgfmt))
     || vt.screen[k].chg)
    {
    vt.curpos = k;
    if (vt.screen[k].chrptr)
    sndchr(*vt.screen[k].chrptr);
    else sndchr(' ');
    }
    }
vt.termpos = -1;
vt.curpos = savpos;
trnfls();
}

E-8
/* NAME
*  sndchr - Send Character to terminal
*
* SYNOPSIS
*  static void sndchr(c)
*  char c;
*
* DESCRIPTION
*  sends character to terminal
*/
static void sndchr(c)
char c;
{
    static ATTR lstatt,tstatt;
    static FLD *lstfld;
    static WND *lstwnd;
    movcur();
    if (vt.screen[vt.termpos].wndpnt != lstwnd ||
        vt.screen[vt.termpos].fldpnt != lstfld)
    {
        lstwnd = vt.screen[vt.termpos].wndpnt;
        lstfld = vt.screen[vt.termpos].fldpnt;
        STRASN(tstatt, vt.screen[vt.termpos].attr);
        tstatt.nfldmrk = FALSE;
        if (memcmp(&tstatt, &lstatt, sizeof lstatt) != 0)
        {
            STRASN(lstatt, tstatt);
            setatr();
        }
    }
    if (vt.screen[vt.termpos].attr.conceald || c < ' ')
        c = ' ';
    tputc(c, chan);
    if(COL(vt.termpos++) == 0)
        vt.termpos = -1;
}

/* NAME
*  setatr - SET Attributes (internal)
*
* SYNOPSIS
*  void setatr()
*
* DESCRIPTION
*  Sets the specified terminal attributes.
*/
static void setatr()
{
    register char *ptr;
    ATTR tnew;
    char buff[25];
    static char cmap[8] = ('0', '2', '4', '6', '1', '3', '5', '7');
    tnew.bright = vt.screen[vt.termpos].attr.bright;
tnew.undrsccor = vt.screen[vt.termpos].attr.undrsccor;
tnew.slowblnk = vt.screen[vt.termpos].attr.slowblnk |
                         vt.screen[vt.termpos].attr.fastblnk;
if (vt.screen[vt.termpos].attr.reverse)
{
    tnew.fgcolor = vt.screen[vt.termpos].attr.bgcolor;
    tnew.bgcolor = vt.screen[vt.termpos].attr.fgcolor;
}
else
{
    tnew.fgcolor = vt.screen[vt.termpos].attr.fgcolor;
    tnew.bgcolor = vt.screen[vt.termpos].attr.bgcolor;
}
ptr = buff;
*ptr++ = '\33';
*ptr++ = '[';
if (tnew.bright)
{
    *ptr++ = ';';
    *ptr++ = '1';
}
if (tnew.undrsccor)
{
    *ptr++ = ';';
    *ptr++ = '4';
}
if (tnew.slowblnk)
{
    *ptr++ = ';';
    *ptr++ = '5';
}
if (cmap[tnew.bgcolor] > cmap[tnew.fgcolor])
{
    *ptr++ = ';';
    *ptr++ = '7';
}
*ptr++ = 'm';
*ptr = '\0';
tputs(buff, chan);
}

*/
* NAME
* movcur - MOVe CURsor (internal)
* *
* SYNOPSIS
* static void movcur()
* *
* DESCRIPTION
* Moves the terminal cursor to the current cursor position
*/

static void movcur()
{
    register int dr, dc, nr, nc;

    if (vt.curpos != vt.termpos)
dr = (nr = ROW(vt.curpos)) - ROW(vt.termpos);
dc = (nc = COL(vt.curpos)) - COL(vt.termpos);
if (vt.termpos >= 0 && dr == 0)
{
    if (nc == 0) tputc(\r, chan);
    else if (dc > 0)
    {
        tputs("\33[", chan);
        tputnum(dc, chan);
        tputc('C', chan);
    }
    else if (dc >= -4)
    { 
        while (dc++ < 0) tputc('\10', chan);
    }
    else if (vt.termpos == 0 && (dc == 0 || nc == 0))
    {
        if (dc != 0) tputc(\r, chan);
        if (dr > 4)
        {
            tputs("\33[", chan);
            tputnum(dr, chan);
            tputc('B', chan);
        }
        else if (dr > 0)
        { 
            while (di-- > 0) tputc('\12', chan);
        }
    }
    else if (dr == -1)
    { 
        tputs("\33M", chan);
    }
    else
    { 
        tputs("\33[", chan);
        tputnum(-dr, chan);
        tputc('A', chan);
    }
}
else
{
    tputs("\33[", chan);
    if (nr > 0) tputnum(nr + 1, chan);
    if (nc > 0)
    {
        tputc(';', chan);
        tputnum(nc + 1, chan);
    }
    tputc('H', chan);
    vt.termpos = vt.curpos;
}
#include "prntdev.h"
APPENDIX F

SAMPLE DEVICE DRIVER (IBM-3270 - COLOR)

/* NAME *
*   ibm3270 - IBM 3270 terminal driver routines
*   Revised: 26-SEP-1986 03:42:57 - WEHRMAN -
NEW-VT-RELEASE
*
* DESCRIPTION
*   device dependent modules for the IBM 3270 device driver.
*/

#include <stdtyp.h>
#include <string.h>
#include <functs.h>
#include <vtdef.h>
#include <trmrtn.h>
#include <ibm3270.h>

#define TBSIZE 4096
static char tbuff[TBSIZE];
static char *tbptr;
static char *tbend = tbuff + TBSIZE;
static char *chan;
static int hilite;

int dovt();
static void intfls();
static void setatr();
static void movcur();
static int getaddr();
static void putaddr();
static void addchr();
static void addstr();
static void refresh();
static void sndchr();

/* NAME *
*   trmini - TeRMinal INItialize
*
* SYNOPSIS
*   bool trmini(tname)
*   char *tname;
*
* DESCRIPTION
*   Opens the terminal specified by tname and initializes it.
*/

F-1
bool trmini(tname)
    char *tname;
{
    int wdth, dpth;
tbuff[0] = W_CMD;
tbuff[1] = 0;
tbptr = &tbuff[2];
if (itopen(&chan, &dpth, &wdth, &hilite))
    return FAILED;
vt.dvce.dspsiz.width = wdth;
vt.dvce.dspsiz.depth = dpth;
return GOODRET;
}

/* NAME
*   trmend - TeRMinal END
* SYNOPSIS
*   void trmend()
* DESCRIPTION
*   Resets the currently open terminal and closes it.
*/

void trmend()
{
    itclos(chan);
}

/* NAME
*   trmfls - TeRMinal FLuSh
* SYNOPSIS
*   void trmfls()
* DESCRIPTION
*   Routine to flush any terminal buffers.
*/

void trmfls()
{
    movcur();
    addchr(IC);
    intfls();
}

/* NAME
*   intfls - INTernal FLuSh
* SYNOPSIS
*   static void intfls()
* DESCRIPTION
*   Internal routine to flush any terminal buffers.
*/
static void intfls()
{
    int len;

    if ((len = tpbr - tbuff) > 2 || tbuff[0] != W_CMD ||
        tbuff[1] != 0)
    {
        tbuff[1] = convrtb[tbuff[1]];
        itsend(chan, tbuff, len);
    }
    tbuff[0] = W_CMD;
    tbuff[1] = 0;
    tpbr = &tbuff[2];
}

/* NAME
 *   trmchk - Terminal CHECK
 * SYNOPSIS
 *   int trmchk()
 * DESCRIPTION
 *   This module returns TRUE if there are input characters to
 *   be processed.
 */

int trmchk()
{
    return (itchek(chan) == 0);
}

/* NAME
 *   trmget - Terminal GET
 * SYNOPSIS
 *   void trmget(cmd)
 *   struct command *cmd;
 * DESCRIPTION
 *   Gets the and processes entire buffer passes back a pc
 *   command or
 *   cursor command depending on mode.
 */

void trmget(cmd)
    struct command *cmd;
{
    int aid, len, pendpos;
    char *bufend;
    bool shift = FALSE, flag = FALSE;
do {
    aid = 0;
    if (!itrecv(chan, tbuff, sizeof tbuff, &len)) {
        tbptr = tbuff;
        bufend = tbptr + len;
        aid = *tbptr++ & 0x3f;
        if (len > 2) vt.termpos = getaddr(tbptr), tbptr += 2;
        while (tbptr < bufend) {
            if (*((tbptr) == SBA) /* has to be SBA */
                if (flag)
                    {flag = FALSE;
                     cmd->funct = EF;
                     cmd->nparm = 1;
                     cmd->parm[0] = 0;
                     tbptr += 3;
                    }
                else
                    {cmd->funct = CUP;
                     cmd->nparm = 2;
                     pendpos = getaddr(tbptr + 1);
                     cmd->parm[0] = ROW(pendpos) + 1;
                     cmd->parm[1] = COL(pendpos) + 1;
                     flag = TRUE;
                    }
            } else
                {cmd->funct = 0;
                 cmd->nparm = 1;
                 cmd->parm[0] = *tbptr++;
                }
            if (dovt(cmd) == -1)
                {aid = 0;
                 break;
                } else if (cmd->funct == 0)
                {vt.curpos++;
                 if (vt.curpos < vt.maxpos)
                     {vt.dvce.cursrs.row = ROW(vt.curpos) + 1;
                      vt.dvce.cursrs.col = COL(vt.curpos) + 1;
                     }
                 else
                     {vt.curpos = 0;
                      vt.dvce.cursrs.row = vt.dvce.cursrs.col = 1;
                     }
                }
        }
    }
}

F-4
cmd->funct = APC;
cmd->nparm = 1;
if (aid == ENTER) cmd->parm[0] = ENTER_KEY;
else if (aid >= PF1 && aid <= PF12)
    cmd->parm[0] = aid - PF1 + (shift ? 11 : 1);
else if (aid >= PF13 && aid <= PF24) cmd->parm[0] =
    aid - PF13 + 13;
else if (aid == PA2)
{
    cmd->funct = 0;
    shift = TRUE;
}
else /* error/unknown aid - refresh screen and try
again
    CLEAR aid - refresh screen before leaving
loop */
{
    cmd->funct = REF;
cmd->nparm = 1;
dovt(cmd);
tbuff[0] = EW_CMD; /* change command */
tbuff[1] |= UNLCK | RSMDT | ALARM; /* add to wcc */
tbptr = &tbuff[2];
refresh();
if (aid == CLEAR)
{
    cmd->funct = APC;
cmd->nparm = 1;
cmd->parm[0] = CLEAR_KEY;
}
else
{
    cmd->funct = 0;
    shift = FALSE;
}
}
} while (cmd->funct != APC);
vterm->curpos = vt->termpos;
tbuff[0] = W_CMD;
tbuff[1] = 0;
tbptr = &tbuff[2];

/* NAME
 * trmput - TeRMinal PUT
 *
 * SYNOPSIS
 * void trmput(cmd)
 * struct command *cmd;
 *
 * DESCRIPTION
 * Puts an internal format command to the terminal.
 */
void trmput(cmd)
    struct command *cmd;
{
    int i, j, savepos;
    char c;
    char intcmd[6];

    switch (cmd->funct)
    {
    case BS:
    case NEL:
    case LF:
    case IND:
    case CR:
    case HTS:
    case RI:
    case DCS:
    case CPL:
    case CUU:
    case CNL:
    case CUD:
    case VPR:
    case CUF:
    case HPR:
    case CUB:
    case CUP:
    case HVP:
    case CPR:
    case HT:
    case CHT:
    case CBT:
    case HPA:
    case VPA:
        movcur();
        break;
    case RIS:
    case REF:
        tbuff[0] = EW_CMD;    /* change command */
        tbuff[1] = 0;
        tbptr = &tbuff[2];
        if (cmd->funct != REF)
            break;
    case RS:
        tbuff[1] |= UNLCK | RSMDT;
        refresh();
        break;
    case BEL:
        tbuff[1] |= ALARM;    /* add to wcc */
        break;
    case MC:
        #ifdef PRINTER
            cmd->funct = REF;
            prnput(cmd);
            prnfls();
        #endif
        break;
    }
}
/* NAME
 *   refresh - REFRESH terminal
 * SYNOPSIS
 *   static void refresh()
 * DESCRIPTION
 *   Clears the terminal screen and rewrites it from the
 *   internal screen.
 */

static void refresh()
{
    register int k, i, mp, savpos;

    savpos = vt.curpos;
    vt.termpos = -1;
    for (i = vt.minprf.row; i < vt.maxprf.row; i++)
    {
        for (k = POS(i, vt.minprf.col), mp = POS(i, vt.maxprf.col); k < mp; k++)
            if (((vt.screen[k].chrptr && (*vt.screenllk].chrptr != vt.screen[k].c))
                || (vt.screen[k].attr.reverse ?
                    vt.screen[k].attrbgcolor :
                    (vt.screen[k].oattr.reverse ?
                        vt.screen[k].oattrbgcolor :
                        vt.screen[k].oattrbgcolor)
                        || vt.screen[k].attrguard !=
                        vt.screen[k].oattrguard
                        || vt.screen[k].attrundrscor !=
                        vt.screen[k].oattrundrscor
                        || vt.screen[k].attrslowblnk !=
                        vt.screen[k].oattrslowblnk
                        || vt.screen[k].attrfastblnk !=
                        vt.screen[k].oattrfastblnk)
                || (vt.screen[k].attr.reverse ?
                    vt.screen[k].attrbgcolor :
                    (vt.screen[k].oattr.reverse ?
                        vt.screen[k].oattrbgcolor :
                        vt.screen[k].oattrbgcolor)
                        || vt.screen[k].oattrbgcolor
                        || vt.screen[k].attrnfldmrk !=
                        vt.screen[k].oattrnfldmrk
                        || (vt.screen[k].c != ' ' && !vt.screen[k].chrptr))
            {
                vt.curpos = k;
                if (vt.screen[k].chrptr)
                    sndchr(*vt.screen[k].chrptr);
                else sndchr(' ');
            }
}
/* NAME
   * sndchr - Send Character to terminal
   *
   * SYNOPSIS
   * static void sndchr(c, )
   *     char c;
   *
   * DESCRIPTION
   * sends character to terminal
   */
static void sndchr(c)
{
    static FLD *lstfld = NULL;
    static WND *lstwnd = NULL;
    int pos = vt.curpos;

    if (vt.screen[vt.curpos].wndpnt != lstwnd ||
        vt.screen[vt.curpos].fldpnt != lstfld)
    {
        lstwnd = vt.screen[pos].wndpnt;
        lstfld = vt.screen[pos].fldpnt;
        if (lstfld && COL(pos) != 0) vt.curpos--;
        else if (lstfld != vt.screen[pos + 1].fldpnt) return;
        movcur();
        setatr(pos);
        vt.screen[vt.termpos].attr.nfldmrk = TRUE;
        STRASN(vt.screen[vt.termpos].oattr,
        vt.screen[vt.termpos].attr);
        if(COL(++vt.termpos) == 0)
            vt.termpos = -1;
        if (!lstfld || COL(pos) == 0) return;
        vt.curpos++;
    }
    movcur();
    if (c < ' ') c = ' ';
    vt.screen[vt.termpos].c = c;
    STRASN(vt.screen[vt.termpos].oattr,
    vt.screen[vt.termpos].attr);
    addchr(c);
    if(COL(++vt.termpos) == 0)
        vt.termpos = -1;
}

/* NAME
   * setatr - SET Attributes
   *
   * SYNOPSIS
   * void setatr(pos)
   *     int pos;
   *
   * DESCRIPTION
   * Sets the specified terminal attributes.
   */
static void setatr(pos)
int pos;
{
    register char *ptr;
    VTATTR tnew;
    int temp;
    char intcmd[25];
    static char coltab[8] = {
        IBLACK,
        IRED,
        IGREEN,
        IYELLOW,
        IBLUE,
        IMAGENTA,
        ICYAN,
        IWHITE,
    };
    static char cmap[8] = {'0', '2', '4', '6', '1', '3', '5', '7'};

    STRASN(tnew, vt.screen[pos].attr);
    tnew.slowblnk = vt.screen[pos].attr.slowblnk |
        vt.screen[pos].attr.fastblnk;
    if (vt.screen[pos].attr.reverse)
    {
        tnew.fgcolor = vt.screen[pos].attr.bgcolor;
        tnew.bgcolor = vt.screen[pos].attr.fgcolor;
    }
    else
    {
        tnew.fgcolor = vt.screen[pos].attr.fgcolor;
        tnew.bgcolor = vt.screen[pos].attr.bgcolor;
    }
    if (hilite == 0)
    {
        intcmd[0] = SF;
        temp = 0;
        if (tnew.guard) temp |= NENT | NUM;
        if (tnew.slowblnk || tnew.fastblnk) temp |= ISEL;
        if (tnew.concealed) temp |= CONC;
        intcmd[1] = convrtb[temp];
        addstr(intcmd, 2);
    }
    else
    {
        intcmd[0] = SFE;
        intcmd[2] = FLDATT;
        temp = 0;
        if (tnew.guard) temp |= NENT | NUM;
        if (tnew.bright) temp |= ISEL;
        if (tnew.concealed) temp |= CONC;
        intcmd[3] = convrtb[temp];
        intcmd[4] = HILITE;
            (!(hilite & 2) && cmap[tnew.bgcolor] >
            
    F-9
cmap[tnew.fgcolor]
? INVID:
  tnew.undrsorc ? ULINE:
  DEFAULT);
if (hilite & 2)
{
  intcmd[1] = 3;
  intcmd[6] = COLOR;
  intcmd[7] = coltab[tnewbgcolor ? tnewbgcolor :
  tnew.fgcolor];
  addstr(intcmd, 8);
}
else
{
  intcmd[1] = 2;
  addstr(intcmd, 6);
}

/* NAME
 * movcur - MOVE CURsor (internal)
 * SYNOPSIS
 * static void movcur()
 * DESCRIPTION
 * Moves the terminal cursor to the current cursor position
 */
static void movcur()
{
  char intcmd[4];

  if (vt.curpos != vt.termpos)
  {
    intcmd[0] = SBA;
    putaddr(&intcmd[1], vt.curpos);
    addstr(intcmd, 3);
  }
  vt.termpos = vt.curpos;
}

/* NAME
 * getaddr - GET ADDRess
 * SYNOPSIS
 * static int getaddr(ptr)
 * char *ptr;
 * DESCRIPTION
 * Returns the 12 or 14 bit address represented by the two
 * bytes pointed to
 */
static int getaddr(ptr)
  char *ptr;
{
if (((*ptr & 0xc0) == 0) return (*ptr << 8) | (*ptr + 1);
else return (((*ptr & 0x3f) << 6) | (*(ptr + 1) & 0x3f));

/ * NAME
* putaddr - PUT ADDRESS

* SYNOPSIS
* static void putaddr(ptr, val)
* char *ptr;
* int val;

* DESCRIPTION
* Encodes the value into a 12 or 14 bit address (depending on vt.maxpos) in
* the two bytes pointed to by ptr.
*/

static void putaddr(ptr, val)
char *ptr;
int val;
{

if (vt.maxpos < 4096)
{
  *ptr++ = convrtb[val >> 6];
  *ptr  = convrtb[val & 0x3f];
}
else
{
  *ptr++ = val >> 8;
  *ptr  = val * Oxff;
}

/ * NAME
* addchr - ADD CHaRacter to buffer

* SYNOPSIS
* static void addchr(c)
* char c;

* DESCRIPTION
* Adds the specified character to the buffer, flushing as required.
*/

static void addchr(c)
char c;
{

if (tbptr >= tbend) intfls();
*tbptr++ = c;
}
/* NAME
 *    addstr - ADD STRing to buffer
 *
 * SYNOPSIS
 *    static void addstr(s, l)
 *    char *s;
 *    int  l;
 *
 * DESCRIPTION
 *    Adds the specified string to the buffer, flushing as required.
 */

static void addstr(s, l)
    char *s;
    int  l;
    {
        if (tbptr + l > tbend) intfls();
        memcpy(tbptr, s, l);
        tbptr += l;
    }