EL-SM070-00

DEC STD 070 Video Systems Reference Manual

For Internal Use Only

digital™

DEC STD 070 Video Systems Reference Manual

DOCUMENT IDENTIFIER: A-MN-ELSM070-00-0000 Rev H, 03-Dec-1991

ABSTRACT: This manual contains The Video Systems Reference Manual (VSRM), a collection of Digital standards relating to the development of video display (interactive) terminals and terminal related products, including printers, personal computers, workstations, and terminal software. Each document in this manual is individually controlled and will be updated as required.

APPLICABILITY: The standards included within this manual are mandatory for engineers designing hardware for terminal products and software engineers designing programs using terminal interfaces. Additional requirements are defined in DEC STD 070-1 Concepts and Conformance Criteria.

STATUS:APPROVED 03-Dec-1991; use VTX SMC for current status.

NOTE

The material referenced by this document is assumed to define mandatory standards unless it is clearly marked as: (a) not mandatory; or (b) guidelines. Material that is marked as not mandatory is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. Guidelines define approaches and techniques that are considered to be good practice, but should not be considered as requirements.

DEC STD 070 Video Systems Reference Manual

DOCUMENT IDENTIFIER: A-MN-ELSM070-00-0000 Rev H, 03-Dec-1991

Rev C,	14-Apr-1989	ECO Number CTS02
Rev D,	06-Sep-1990	ECO Number CTS03
Rev E,	17-Oct-1990	ECO Number CTS04
Rev F,	20-Mar-1991	ECO Number CTS05
Rev H,	03-Dec-1991	ECO Number NR006

Document Management Category:	Terminal Interface Architecture (STI)
Responsible Department:	VIPS Terminals Architecture
Responsible Person:	Peter Sichel
SMC Writer:	Georgia Ireland

APPROVAL: Peter Sichel- VIPS Terminals Architecture

Peter A. Sichel

Peter Sichel - VIPS Terminal Interface Architecture Control

Direct requests for further information to the responsible person shown on the management page of each document in this manual.

Use VTX SMC to order copies of this document from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 234-4423

Table of Contents/Revision Status

DEC STD 070-0 VIDEO SYSTEMS STANDARD -

INTRODUCTION

Document Identifier: A-DS-EL00070-00-0 Rev B, 14-Apr-1989

ABSTRACT: This standard contains an introduction to DEC STD 070 Video Systems Reference Manual (VSRM), which is a collection of Digital standards relating to the development of video display (interactive) terminals and terminal related products, including printers, personal computers, workstations, and terminal software. It describes the audience for DEC STD 070 and those responsible for its development and maintenance. It also defines key terminology, and provides a Table of Contents for all sections of the manual.

Each document in this manual is controlled individually. The material in this handbook will be updated as required.

APPLICABILITY: This standard is mandatory for engineers designing hardware for terminal products and software engineers designing programs using terminal interfaces. Additional requirements are defined in DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria.

STATUS: APPROVED 14-Apr-1989; type $ VTX SMC for expiration date.

digital™

TITLE: DEC STD 070-0 VIDEO SYSTEMS STANDARD - INTRODUCTION

DOCUMENT IDENTIFIER: A-DS-EL00070-00-0 Rev B, 14-Apr-1989

REVISION HISTORY: Rev A, 29-Apr-1988
Rev B, 14-Apr-1989 ECO #CTS01

Document Management Category:	Terminal Interface Architecture (STI)
Responsible Department:	DSG Terminals Architecture
Responsible Person:	Peter Sichel
SMC Writer:	Patricia Winner

APPROVAL: This document, prepared by the Desktop Systems Group, has been reviewed and recommended for approval by the General Review Group for its category for use throughout Digital.

Peter Conklin, Technical Director,
Desktop Systems Group

Peter A. Sichel
Peter Sichel, Desktop Systems Group

Eric A Williams
Eric Williams, Standards Process Manager

Direct requests for further information to Peter Sichel, DSG1-2/C7, DTN 235-8374, HANNAH::TERMARCH

Copies of this document can be ordered from Standards and Methods Control, $ VTX SMC, CTS1-2/D4, DTN 287-3724, or JOKUR::SMC.

Please provide your name, badge number, cost center, mailstop, and ENET node when ordering.

digital™

CONTENTS

1 INTRODUCTION

Digital Equipment Corporation has traditionally been a pioneer in developing and implementing standards for terminal service interfaces. The VT100 was Digital's first terminal based on the ANSI standard for control sequence encoding (ANSI X3.64). Many of Digital's printer and display terminals have since implemented ANSI and DEC-private controls, and Digital has been an active participant in the development of new controls. The development of ReGIS (Remote Graphics Instruction Set) has enabled the corporation to implement a variety of graphics terminal products, and establish Digital as a force in the graphics terminal market.

Digital has built a number of terminals based on these interfaces, and has committed that future terminal products will continue to support these interfaces to ensure software compatibility for operating systems as well as application programs. It is expected that computing terminals and workstations will also emulate these interfaces in order to take advantage of the large body of software that already exists for these devices.

The DEC STD 070 Video Systems Reference Manual (or Video SRM) documents the interfaces to Digital video terminal products. It is intended to serve as both an implementation specification for product developers, and as a guide to programmers attempting to produce software that is transportable across a wide range of devices. It defines the interfaces in a manner that is independent of the physical characteristics of a particular device implementation. Enough flexibility is allowed in the interface description to permit a wide variety of terminal products to be produced, while ensuring that software can be designed that can intelligently make use of the full range of features in a given product.

As new functions are developed for new terminal products, they will be added to the Video SRM before final implementation. Furthermore, an architectural certification process provides a means of measuring product performance against the Video SRM, and ensures software compatibility across products.

Products claiming conformance to the architectural specifications contained in the Video SRM must be submitted for certification testing and evaluated against the requirements of these standards.

2 AUDIENCE

There are two intended audiences for the Video SRM. One is the engineers who design the hardware and firmware for future terminal products. It is important that these designers have a thorough knowledge of the functionality required at the device interface in order to build devices at a reasonable cost that will be interface compatible with existing software products. The Video SRM provides a detailed description of the logic required to implement these interfaces, and thus guarantees that gratuitous changes will not be introduced.

The second audience for the Video SRM is the software engineers who design programs that use terminals to implement their user interfaces. The Video SRM provides these programmers with a convenient single source of information regarding Digital's family of terminals and workstation products. This document enables those engineers to produce device-independent software without the need to consult a proliferation of user's manuals and understand the discrepancies between implementations.

The greatest benefit of this document is to the Digital customer. By providing a single point of reference between hardware and software components designed by the corporation, Digital significantly reduces consternation and confusion among field sales and service personnel and satisfies the marketplace that Digital provides system components that are designed to work together.

3 RESPONSIBILITY

Development and maintenance of the DEC STD 070 Video Systems Reference Manual is the responsibility of the Terminals Architecture Group, reporting to the Technical Director of the Desktop Systems Group (DSG). Distribution and control of the document is handled through the Standards and Methods Control group of International Standards, Information, and Services (ISIS).

Section 1 delineates general architectural concepts and conformance certification criteria applicable to these requirements.

4 GENERAL TERMINOLOGY

The following terminology is used throughout this document and applies across each section.

coding interface - A software or hardware interface through which bytes of character-coded information are passed between terminal equipment and an application process across a host port, or between terminal equipment and a printer across a printer port.

external interface - Product interfaces between a terminal, personal computer, or workstation, and a remote system.

Host Port - The coding interface between a terminal or terminal process and an application process, whether the application process is running inside the terminal or in a host computer.

internal interface - Product interfaces between a terminal sub-system and software processes running within a terminal, personal computer, or workstation.

Printer Port - The coding interface between a terminal or terminal process and a printer.

receive - To accept coded character information across a coding interface.

service class - A set of functionally related terminal control operations, associated by a common object type.

transmit - To transfer coded character information across a coding interface.

user; terminal user - A person operating a terminal device for the purpose of interacting with some component of a computer system or network.

The following Table of Contents presents an outline of this manual and serves to point the reader to major areas of information. A more detailed Table of Contents is available with each section.

CONTENTS

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

CHAPTER 7 DEC STD 070-7 VIDEO SYSTEMS REFERENCE MANUAL - PRINTER PORT EXTENSION

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

CHAPTER 9 DEC STD 070-9 VIDEO AND PRINTER SYSTEMS REFERENCE MANUAL - SIXEL GRAPHICS EXTENSION

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

CONTENTS

A more detailed Table of Contents is available with each section.

5 REFERENCED DOCUMENTS

The following corporate, national, and international standards apply to the interfaces defined within DEC STD 070 and are referenced herein.

Digital Standards

EL-00052-00	DEC STD 052-0 Operational Requirements for Serial Terminals and Computer Interfaces Operating as DTEs Connected to EIA RS-232-C or CCITT V.28 Modems
EL-00052-01	DEC STD 052-1 Operational Requirements for Serial Terminals and Serial System Interfaces Operating as DTEs Connected to EIA RS-232-C or CCITT V.28
EL-00070-01	DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria
EL-00070-05	DEC STD 070-5 Video Systems Reference Manual - Character Cell Display
EL-00070-06	DEC STD 070-6 Video Systems Reference Manual - Keyboard Processing
EL-00070-07	DEC STD 070-7 Video Systems Reference Manual - Printer Port Extension
EL-00107-00	DEC STD 107-0 Ditigal Standard for Terminal Keyboards - Standard Keyboard Layouts
EL-00138-00	DEC STD 138-0 Registry of Control Functions for Character-Imaging Devices
EL-00169-00	DEC STD 169-0 DEC Standard Coded Graphic Character Sets for Hardware and Software

Copies of Digital Standards Can be obtained from Standards and Methods Control, $ VTX SMC, JOKUR::SMC, DTN 287-3724, or CTS1-2/D4.

Please provide your name, badge number, cost center, mailstop, and ENET node when ordering.

Other Digital Documents

A-SP-LK200-A-0	LK200 Functional Specification
A-SP-LK201-A-2	LK201 Keyboard Design Specification

Specifications are available from any Microfilm Reference File (35mm aperture "tub").

ANSI And ISO Standards

ANSI X3.4 - 1986	American National Standard Code for Information Interchange (ASCII character set)
ANSI X3.41 - 1974	American National Standard Code Extension Techniques for use with the 7-Bit Coded Character Set of the American National Standard Code for Information Interchange
ANSI X3.64 - 1979	Additional Controls for use with American National Standard Code for Information Interchange
ANSI X4.23 - 1982	Keyboard Arrangement for Alphanumeric Machines
ANSI X3.122-1986 (IS8632-1986)	Computer Graphics Metafile
ANSI X3.124-1985 (IS7942-1985)	Graphical Kernel System (GKS)
dpANS X3.134.1-1985	8-bit ASCII Structure and Rules
dpANS X3.134.2-1985	7-bit and 8-bit ASCII Supplemental Multinational Graphic Character Set
ISO 646	Information Processing / 7-Bit Coded Character Set for Information Interchange
ISO 2022:1986	Information Processing / ISO 7-Bit and 8-Bit Coded Character Sets - Code Extension Techniques

ANSI AND ISO STANDARDS (Continued)

ISO 6429:1988	Information Processing / Control Functions for Coded Character Sets
(unnumbered)	ISO Register of Coded Character Sets for Use with Escape Sequences
ISO 8859-1:1987	Information Processing / 8-Bit Single-Byte Coded Character Sets - Part 1 : ISO Latin Alphabet Nr 1;

Copies of ANSI and ISO Standards can be obtained from local Digital Libraries and the American National Standards Institute, 1430 Broadway, New York, NY 10018, telephone (212) 354-3300 X479.

READER COMMENTS

Your comments and suggestions will help Standards and Methods Control improve their services and documents.

Did you request this document? _____ If so, did it arrive within a satisfactory period of time? _____ Please comment.

What are your impressions of this document? Consider format, organization, completeness, readability, and illustrations.

- -- -- -- -- -- -- -- -- -- -- FOLD ON THIS LINE -- -- -- -- -- -- --

Did you find technical or clerical errors in this document? If so, please specify the page number(s) and the error(s).

Are the instructions for the update package clear? _____
Do you have other suggestions for improving this document?

The following information is optional:

Name _____ Mailstop _____

Department _____ Node _____

Send your comments to JOKUR::PROJECTS, or fold, staple, and send this page through interoffice mail to:

READERS' COMMENTS
STANDARDS AND METHODS CONTROL
CTS1-2/D4

digital™

digital™

VIDEO SYSTEMS REFERENCE MANUAL

Concepts and Conformance Criteria

Document Identifier: A-DS-EL00070-01-0 Rev. AX10, 15-May-1983

ABSTRACT: This section describes the conformance requirements for devices implementing the Video Systems and intending to be certified as conforming implementations.

APPLICABILITY: SARA Formal Cross-Component Standard. Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces.

STATUS: FOR REVIEW ONLY

+----------+
| This document has been placed in the SARA "Formal Cross-Component Standard" category. The material contained within this document is assumed to define mandatory standards unless it is clearly marked as (a) not mandatory or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements. The procedures for modifying or evolving this standard are contained within the contents of this document. |
+----------+

+----------+
| FOR INTERNAL USE ONLY |
+----------+

TITLE: VIDEO SYSTEMS - CONCEPTS AND CONFORMANCE CRITERIA

DOCUMENT IDENTIFIER: A-DS-EL00070-01-0 Rev. AX10, 15-May-1983

REVISION HISTORY: Original Draft 25-Dec-1982
Revision AX01 28-Feb-1983

FILES: User Documentation EL070S1.mem

Internal Documentation EL070S1.rno
EL070S1.rnt
EL070S1.rnx

Document Management Group: Terminal Interface Architecture
Responsible Department: Video Architecture Group
Responsible Individual: Peter Sichel

ACCEPTANCE: This document has been approved by the Manager of the Video Architecture Group based on a comprehensive review of its individual sections by the members of the SARA component groups working Terminal Interface Architecture issues. The list of individuals on the review and approval list are on file in Standards and Methods Control.

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, RANI::VIDARCH

Copies of this document can be ordered from:

Standards and Methods Control
APO1/F7, DTN 289-1414, JOKUR::SIMONETTI

CONTENTS

1.1 INTRODUCTION

The Video Systems Reference Manual (or Video SRM) documents the interfaces to DEC video terminal products. It is intended to serve as both an implementation specification for product developers, as well as a guide to programmers attempting to produce software which will be transportable across a wide range of devices. It defines the interfaces in a manner which is independent of the physical characteristics of a particular device implementation. Enough flexibility is allowed in the interface description to permit a wide variety of terminal products to be produced, while insuring that software can be designed that can intelligently make use of the full range of features in a given product.

As new functionality is developed for new terminal products, it will be added to the Video SRM before final implementation. Furthermore, an architectural certification process provides a means of measuring product performance against the Video SRM, and ensures software compatibility across products.

Products claiming conformance must be submitted for certification testing and evaluated against these standards.

1.2 STRUCTURE OF THE MANUAL

Terminal technology has become increasingly complex as the emphasis in computer system design has shifted from performing mathematical and scientific calculations to making information services readily accessible to the public. Even so-called "dumb terminals" of today incorporate a wide range of capabilities that bridge all aspects of computer architecture and design. In order to reduce this complexity to a state which is comprehensible to those who are not intimately familiar with these technologies, this manual has been broken up into sections which describe different functional components of a typical terminal system. Although there is some interaction between these components, they are described in such a way that they can more or less stand alone.

The sections of this manual include descriptions of the following components:

- Code Extension Layer - defines 7 and 8 bit character processing; defines escape and control sequence parsing; defines compatibility with DEC STD 169 for hardware, including graphic code extension techniques.

- Terminal Management - defines basic device independent terminal management functions.

- Character Cell Display - defines Level 1 and Level 2 operation for character cell display devices.

- Keyboard Processing - defines use of the LK201 keyboard in terminals.

- Printer Port - defines the display terminal to printer interface.

- ReGIS Graphics - defines descriptor graphics protocol for Base ReGIS and Raster Extensions.

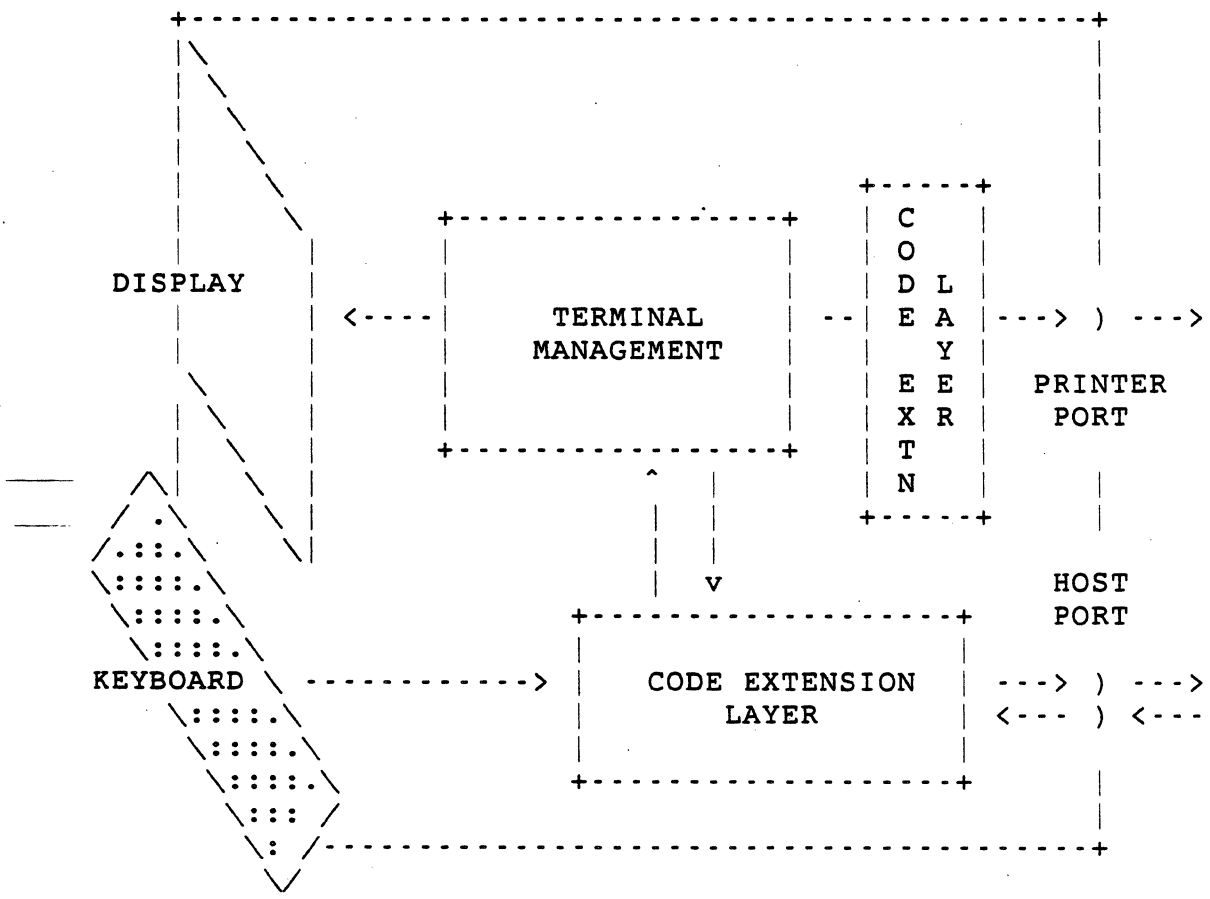
- Sixel Graphics - defines serial protocol for image transfer.

- Dynamically Redefinable Character Sets - defines loading and designation of Dynamically Redefinable Character Sets ("soft" character sets).

- User Definable Keys - defines loading and invocation of User Definable Keys (programmable function keys).

These components have the following functional relationship in the design of an actual terminal product:

Components of a Terminal System



1.3 ARCHITECTURAL STRUCTURE

Terminals are an important part of any communications scheme. The earliest computer communications was performed using coded character sets and control functions. This document describes in detail some portions of an architectural model which provides a consistent and uniform basis for the development of terminal service interfaces.

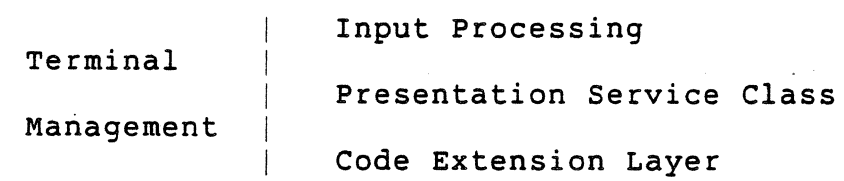
1.3.1 Virtual Terminals

The architectural model is built around the concept of 'Virtual Terminals', which virtualize the interface to real physical terminals. "A virtual terminal is a model which defines an abstract terminal in terms of logical functions that can be implemented differently on different real terminals; therefore a virtual terminal consists of a real terminal and a protocol adaptor." (Electrical Communication, Vol. 56, No. 1, 1981). The virtual terminal interface consists of a number of disparate service classes, or sets of functions.

1.3.2 Layers

The architecture is layered in an attempt to group those sets of functions together which perform similar services within the communications structure.

The architecture is broken down broadly into three functional layers. These layers are referred to as the Code Extension layer, the Presentation Service Class layer, and the Input Processing layer. In addition, there is a set of management functions which operate across layers.



The purpose of layering the architecture is to separate components which are logically distinct, allowing components in one layer to be modified or replaced without impacting components in the other layers of the architecture. For example, the incorporation of a new keyboard interface would not necessarily affect the presentation protocol, or the syntax of the communications interchange.

1.3.2.1 Code Extension Layer -

The lowest layer of the architecture is referred to as the Code Extension Layer. This layer includes all services which form a part of the interface into the communications environment, as well as the syntax, or coding, of data and control functions.

Since all of the existing terminal services use character coded data and controls (referred to as "ASCII stream"), this layer currently includes the following functions:

- Communications Controls (such as XON/XOFF synchronization, padding, etc.)
- ANSI Parsing (separation of controls and data)
- Graphic Character Coding (7-bit and 8-bit environments)

In the future this layer may be extended or replaced with other interfaces into DECnet, ISDN, X.25, etc., without affecting the overall structure of the architecture.

1.3.2.2 Presentation Service Class -

The next layer is referred to as the Presentation Service Class Layer. This layer includes all services which directly provide imaging information. Output to the terminal is presented in the form of data types, whose structure and operations are unique to each service class definition. The only service class currently defined for this layer is the Character Cell Display. Future architectural development may include the definition of the following service classes:

- Character Cell Printer
- Proportionally Spaced Printer
- Videotex
- Image Transfer
- Voice

1.3.2.3 Input Processing -

The upper layer of the architecture is referred to as the Input Processing Layer. This layer includes all services which relate to the processing of data entered into the terminal by the terminal user. Currently the only such device is the corporate standard keyboard, but this may be extended in the future to include other entry devices (such as voice input, mice, digitizers, etc.).

This layer also may be extended to include local processing functions such as editing of text and graphics. It should be noted that the services provided at this layer may utilize the various services of the Presentation Service Class Layer in order to execute the functions they provide. For example, a field editor might use ANSI text, ReGIS text, Videotex text, Teletex text (or even voice I/O) to perform the same requirements of processing field input.

1.3.2.4 Terminal Management -

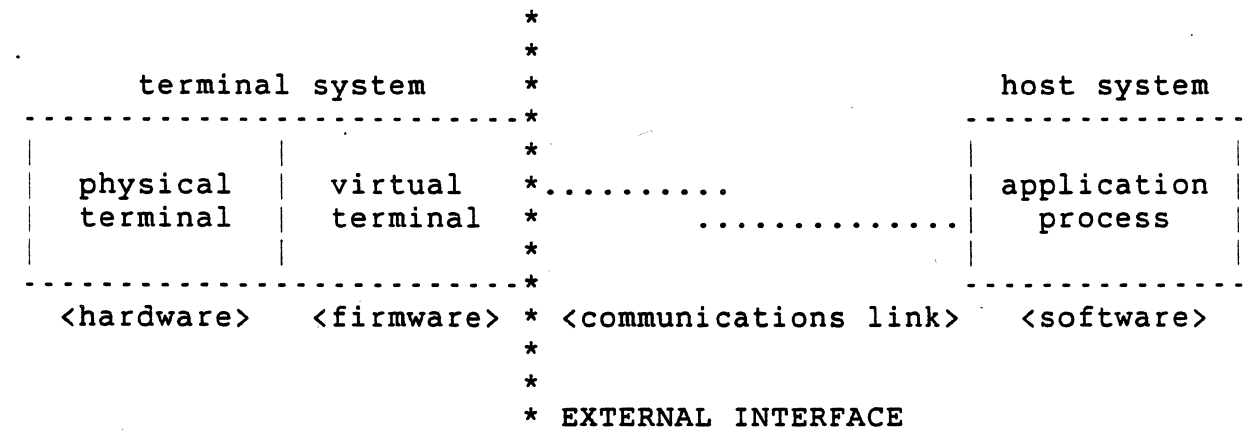
Terminal Management functions operate across the other layers of the architecture. These functions include all services which provide for identification, negotiation, and management of device resources. Examples of these services include Device Attributes (DA), Soft Terminal Reset (DECSTR), and other control functions which affect the physical state of the terminal device. In the future this layer may include multiplexing of virtual terminal connections (Virtual Terminal Management), service class selection, and window management (multiple display windows).

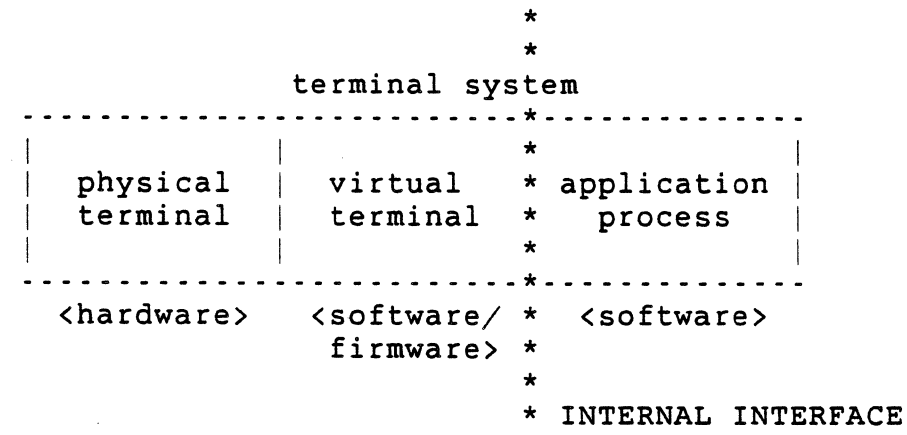
1.3.3 Service Classes

Within each layer of the architecture there may be classes of service. Functions within a layer are grouped into classes according to their logical relationship. For example, character oriented text displays might form a service class at the presentation layer which is distinct from a graphics protocol using logical pixel addressing. Furthermore, character oriented keyboard processing would form a class at the applications layer distinct from a forms editing protocol. By separating these functions according to classes it is possible to build a variety of compatible products which conform to a common set of functions within a given class of operations.

1.4 SCOPE

The interfaces defined within this specification apply to both internal and external product interfaces. External interfaces are interfaces between a terminal, personal computer or workstation and a remote system. Internal interfaces are interfaces between a terminal sub-system and software processes running within a terminal, personal computer or workstation.





External and Internal Terminal Interfaces

1.5 ARCHITECTURAL CONFORMANCE

The Video Systems Reference Manual is based on a modular architectural structure made up of discrete components which can be combined in a variety of ways to meet evolving product needs and requirements. For purposes of defining conformance, service class interfaces can be further subdivided into levels, extensions, and exceptions. The following definitions describe these concepts.

1.5.1 Definitions

1.5.1.1 Levels -

Levels reflect an agreement between software and hardware on a fixed group of functions within a class of operations that are to be adhered to for some period of time. When significant new functionality is required to be added, a new level will be defined and agreed to, which will be a super set of the levels below. Lower levels of the architecture may be discontinued from use as new higher levels are defined. The decision to discontinue a lower level will be based on marketing and product requirements.

Each level will consist of a number of functions which must be included in all products claiming to implement that level. These functions are referred to as "required". When a level n function is used in level n+1 operation, it must be defined in such a way that it will not affect any level n state differently than it does in level n, to insure compatibility for level n conforming software running in level n+1.

The advantage of having levels is that software can be written to a specific level of functionality based on hardware availability and compatibility requirements, and will be guaranteed all of the functionality of that level and all lower levels. As new levels are adopted, terminals can be upgraded to support the new levels without losing compatibility with existing software. Likewise, software can be intelligently upgraded to support new terminal products without losing backwards compatibility with previous levels of the architecture.

1.5.1.2 Extensions -

In order to minimize the number of service classes and levels supported by the architecture, and still meet a wide variety of product needs, each level of the architecture will have certain defined extensions. Extensions are fully specified by the architecture, but are optional to product implementors claiming to conform to a particular level of the architecture. (Products may,

in fact, permit the customer to optionally purchase extensions.) Functions which are included as extensions are not guaranteed to software, and therefore conforming software should not be dependent on those functions to operate in a logical and reliable manner. For example, 132-column mode is an extension to the Character Cell Display service class, and thus might not be included in all products. Software should be designed in such a way that it will behave in a logical manner if this feature is not present in a particular terminal product.

1.5.1.3 Exceptions -

Each level may also include a set of "exceptions". These functions are not required by product implementations, and are not to be used by conforming software. For example, Tektronix 4010 emulation might be required by particular products, but not be a part of the architecture because DEC software did not want to support this feature. The interface, however, would be documented and included as an appendix to the architecture, so that products implementing this feature could do so in a compatible manner.

Products must get management approval before using exceptions to the architecture, and any new exception specifications must be approved by the Terminal Interface Architecture Review Board before being implemented.

1.5.1.4 Exclusions -

In rare cases there may be certain functions which are explicitly excluded from a given level. This will typically be done to clarify that a particular function is not allowed for implementation at that level.

1.5.1.5 Waivers -

Products which cannot pass certification cannot claim conformance with the architecture unless they receive a formal waiver from Terminals and Workstations management. The following are examples of conditions for which a waiver might be granted:

1. If a product wished to implement functions which were not previously defined by the architecture, and were not warranted to be general enough to require new architectural development.

2. If a product were incapable of properly implementing a defined function for sound technical reasons, but in other ways conformed to the architecture.

3. If a product conforming to a certain level wished to include functions defined by a higher level, but not implement all of the functions of that level.

Waivers are intended to be used to resolve time-to-market issues and are temporary. Products not conforming are expected to come into conformance by changing the product, or by changing the architecture through the Architecture ECO process.

1.5.1.6 Deviations -

Deviations represent permanent non-conformance with the architecture. Deviations apply only to products which were developed before the architecture was defined, but are in general conformance with the architectural specifications. New products are not allowed deviations.

1.5.1.7 UNDEFINED Operations -

Occasionally there are operations whose effect cannot be guaranteed to be the same in all implementations. Such operations shall be identified in both the architectural and product specifications as UNDEFINED. Software conforming to the architecture shall not use UNDEFINED operations. Every attempt will be made to minimize the number of UNDEFINED operations in the architecture, since software that executes UNDEFINED functions may not operate the same on all terminal products.

1.5.2 User Preference Features

Certain states of the terminal are defined as "User Preference Features". User Preference Features that have a coded representation for interchange with a host are indicated to be of a special status in this specification.

The terminal user should be in charge of the state of all User Preference Features, not the general application program. Conforming software applications shall not change the state of any User Preference Feature, except in response to an explicit

request by the terminal user. It is preferred that these requests be handled as an operating system service. In cases where these services are not provided by the operating system, and these functions are changed by an application program, the operating system must be notified.

1.5.3 Rules For Conformance

Terminal and Workstation products will be required to provide standard interfaces for the purpose of synchronizing hardware product release with software product support. To this end, it is important that products claiming to conform to the architecture should follow certain guidelines in implementing the architectural interfaces. Products should not be excluded from implementing features not defined by the architecture. However, it should be made clear in both the architectural and product documentation that these features are not guaranteed in all products, and should not be used by conforming software. Products should not be allowed to selectively implement portions of the architecture without receiving an explicit waiver from Terminals and Workstations management prior to beginning any product development, as this will render the architecture useless in determining cross-product compatibility.

1.5.3.1 Hardware/Firmware Conformance -

1.5.3.1.1 Levels - Terminal hardware/firmware claiming to conform to Level n:

1. shall implement ALL functions of the lowest defined level to Level n,

2. may NOT implement any functions defined by a higher level unless it implements all of the functions of that level (in which case it would conform to the higher level),

3. may NOT implement any function not defined in any level (unless it is included in an appendix to both the architectural and product specifications and clearly identified as an exception to the architecture).

For example, a conforming Level 1 implementation cannot leave out bold, since it is a Level 1 function (however, bold may be implemented either by increasing the intensity or the number of pixels). A conforming Level 1 implementation cannot implement any Level 2 functions (or it must implement all of the Level 2 functions, in which case it is a Level 2 conforming terminal).

1.5.3.1.2 Extensions -

Terminal hardware/firmware claiming to implement an extension shall implement ALL functions of that extension and the associated conformance level, but need not implement any lower level extensions, unless specifically required by the specification of the extension.

1.5.3.1.3 Exceptions -

Finally, a conforming terminal may NOT implement any VT100/VT125 function that has been excluded from all of the levels by this document, unless it has been included in an appendix to this document. Such functions will be defined for backward compatibility with existing terminals. However, conforming DEC software shall NOT use those functions defined in an appendix.

1.5.3.2 Software Conformance -

1.5.3.2.1 Levels - Software claiming to conform to Level n:

1. may use ANY functions in the lowest defined level to Level n,
2. shall NOT use any functions of levels higher than n,
3. shall NOT use any functions defined in an appendix of the SRM,
4. shall NOT depend on product-specific side effects which are left UNDEFINED by the architecture.

1.5.3.2.2 Extensions -

Software supporting an extension may use any function in that extension (and any extensions which are required to support that extension), but shall use the functions in such a way as to make the software modular with respect to extensions when it makes sense to do so. Furthermore, software that supports an extension shall operate in a sensible way if that extension is NOT present in the terminal. The intent of this requirement is two-fold. First, no conforming software should ever blindly assume that it is using a terminal with an extension which it requires. All software must verify that its environment is correct. Second, if the software determines that it does not have the required facilities, it must behave rationally. This could mean that certain features of the software are inoperative. Or, if the entire software package requires an extension (e.g., DECgraph and ReGIS), then rational behavior means exiting gracefully with an appropriate informational message.

1.6 LEVEL 1 CONFORMANCE

Level 1 of the architecture provides a compatibility mode for software designed to operate with the VT100 family of display terminals, specifically the VT102 display terminal and VT125 graphics display terminal.

1.6.1 Conforming Products

For a list of conforming Level 1 products, consult the appendix "Product Reference".

1.6.2 Required Functions

The following controls are required for all conforming Level 1 implementations.

NUL	NUL
SPACE	SP
SHIFT IN (LOCKING SHIFT ZERO)	SI (LS0)
SHIFT OUT (LOCKING SHIFT ONE)	SO (LS1)
SINGLE SHIFT TWO	SS2
SINGLE SHIFT THREE	SS3
IDENTIFICATION	DECID
DEVICE STATUS REPORT	DSR
SET TOP AND BOTTOM MARGINS	DECSTBM
SELECT CHARACTER SET to G0, G1	
ASCII	
Line Drawing	
SELECT GRAPHIC RENDITION	SGR
Normal	
Bold	
Blink	
Underscore	
Reverse Video	
SINGLE WIDTH LINE	DECSWL
DOUBLE WIDTH LINE	DECDWL
DOUBLE HEIGHT LINE TOP	DECDHLT
DOUBLE HEIGHT LINE BOTTOM	DECDHLB
Set/Reset Mode	SM,RM
SCROLLING MODE	DECSCLM
SCREEN MODE	DECSCNM
ORIGIN MODE	DECOM
NEW LINE MODE	LNM
KEYBOARD ACTION MODE	KAM
AUTO REPEAT MODE	DECARM
CURSOR KEY MODE	DECCKM
KEYPAD MODE	DECKPAM,DECKPNM
CURSOR UP	CUU
CURSOR DOWN	CUD
CURSOR FORWARD	CUF
CURSOR BACKWARD	CUB
CURSOR POSITION	CUP
HORIZONTAL/VERTICAL POSITION	HVP
CURSOR POSITION REPORT	CPR
SAVE CURSOR	DECSC
RESTORE CURSOR	DECRC
BEL	BEL
BACK SPACE	BS
HORIZONTAL TAB	HT
LINE FEED	LF

VERTICAL TAB	VT
FORM FEED	FF
CARRIAGE RETURN	CR
SUBSTITUTE	SUB
INDEX	IND
REVERSE INDEX	RI
NEXT LINE	NEL
HORIZONTAL TABULATION SET	HTS
TABULATION CLEAR	TBC
ERASE IN LINE	EL
ERASE IN DISPLAY	ED

1.6.3 Extensions To Level 1

The following functions are each independent extensions to Level 1 as an implementation or customer option, and require the associated controls.

1.6.3.1 Identification And Conformance -

The following functions form an extension to Level 1 that is required in all future products, and in Level 2:

DEVICE ATTRIBUTES (primary)	DA(1)
DEVICE ATTRIBUTES (secondary)	DA(2)
SELECT CONFORMANCE LEVEL	DECSCL

1.6.3.2 Editing Controls -

The following editing controls form an extension to Level 1 that is required in all future products, and in Level 2:

Set/Reset Mode	SM,RM
INSERT/REPLACEMENT MODE	IRM
DELETE CHARACTER	DCH
INSERT LINE	IL
DELETE LINE	DL

1.6.3.3 132-Column Mode -

Set/Reset Mode	SM,RM
COLUMN MODE	DECCOLM

1.6.3.4 Printer Port -

DEVICE STATUS REPORT	DSR
Set/Reset Mode	SM,RM
PRINTER CONTROLLER MODE	DECPCM
AUTO PRINT MODE	DECAPM
PRINT FORM FEED MODE	DECPFF
PRINT EXTENT MODE	DECPEX
Media Copy	MC
PRINT SCREEN	
PRINT LINE	

1.6.3.5 ReGIS Graphics Display -

Set/Reset Mode	SM,RM
GRAPHICS CURSOR ENABLE MODE	DECGCEM
Media Copy	MC
Device Control String	DCS,ST
ENTER ReGIS GRAPHICS	
ReGIS Commands	
SCREEN CONTROL	S
POSITION CONTROL	P
WRITING CONTROL	W
VECTOR DRAWING	V
CURVE DRAWING	C
TEXT DRAWING	T
REPORT	R
LOAD CHARACTER SET	L
MACROGRAPH	M

1.6.3.6 Sixel Protocol -

As defined in the section "Sixel Graphics Extension"

1.6.3.7 Katakana Character Set -

SELECT CHARACTER SET to G0, G1
- JIS_Roman
- JIS_Katakana

1.6.4 Level 1 User Preference Features

The following functions are included in Level 1 operation, but are indicated as User Preference Features, and thus are not to be used by conforming DEC application programs (except as described above):

Set/Reset Mode	SM,RM
COLUMN MODE	DECCOLM
SCROLLING MODE	DECSCLM
SCREEN MODE	DECSCNM
AUTO REPEAT MODE	DECARM

1.7 LEVEL 2 CONFORMANCE

Level 2 defines a new level of architectural compatibility for video display terminals which is a complete super set of Level 1.

1.7.1 Conforming Products

For a list of conforming Level 2 products, consult the appendix "Product Reference".

1.7.2 Required Functions

The following controls are required for all conforming Level 2 implementations.

All Level 1 required functions, and all Level 1 extensions required for future products, plus:

SELECT 7-BIT C1 TRANSMISSION	S7C1T
SELECT 8-BIT C1 TRANSMISSION	S8C1T
LOCKING SHIFT TWO	LS2
LOCKING SHIFT THREE	LS3
LOCKING SHIFT ONE RIGHT	LS1R
LOCKING SHIFT TWO RIGHT	LS2R
LOCKING SHIFT THREE RIGHT	LS3R
SOFT TERMINAL RESET	DECSTR
SELECT CHARACTER SET to G0, G1, G2, G3	
ASCII	
Line Drawing	
Supplemental	
Set/Reset Mode	SM,RM
TEXT CURSOR ENABLE MODE	DECTCEM
INSERT CHARACTER	ICH
ERASE CHARACTER	ECH

1.7.3 Extensions To Level 2

The following functions are each independent extensions to Level 2 as an implementation or customer option, and require the associated controls.

1.7.3.1 132-Column Mode -

All Level 1 required functions for this extension.

1.7.3.2 Printer Port -

All Level 1 required functions for this extension.

1.7.3.3 ReGIS Graphics Display -

All Level 1 required functions for this extension.

1.7.3.4 Sixel Protocol -

All Level 1 required functions for this extension.

1.7.3.5 Katakana Character Set -

All Level 1 required functions for this extension.

1.7.3.6 Selectively Erasable Characters -

SELECT CHARACTER ATTRIBUTE	DECSCA
SELECTIVE ERASE IN LINE	DECSEL
SELECTIVE ERASE IN DISPLAY	DECSED

1.7.3.7 DRCS (Dynamically Redefinable Character Set) -

DOWN LINE LOAD DRCS	DECDLD
SELECT CHARACTER SET to G0, G1, G2, G3 DRCS	

1.7.3.8 UDK (User Defined Keys) -

DEVICE STATUS REPORT	DSR
USER DEFINED KEYS	DECUDK

1.7.4 Level 2 User Preference Features

The following functions are included in Level 2 operation, but are indicated as User Preference Features, and thus are not to be used by conforming DEC application programs (except as described above):

Set/Reset Mode	SM,RM
COLUMN MODE	DECCOLM
SCROLLING MODE	DECSCLM
SCREEN MODE	DECSCNM
AUTO REPEAT MODE	DECARM

1.8 CERTIFICATION

Certification testing is performed on products claiming to conform to the architecture by the Terminals and Workstations Certification Group. Products may not claim compatibility with the architecture or with other Terminals and Workstations products unless they have passed certification testing, or have been given specific clearance to do so by Terminals and Workstations management.

The goal of Certification is to prove a product's compatibility claims, not according to its own interpretation of what is required for compatibility, but as defined in The Video Standards Systems Reference Manual (DEC Standard 070). This means that multiple products will use the same set of Certification tests.

Certification assumes the unit functions, but checks if the functions were implemented in a compatible manner. It is also important to note that certification testing is limited to that subset of functions which are either required functions or extensions to the architecture. This implies that a product that passes certification may have problems executing commands which are not part of that subset.

Outstanding certification issues are handled through the Waiver Process, and are not likely to cause a delay in shipment. Non-compliances are brought to the attention of upper management who can then decide to restrict the advertising of certain compatibility claims until the product passes certification.

1.8.1 Certification Test Process

Certification is a five step process:

1. Review Architectural and Product Specifications

   The Certification Group participates in the review of new architectural elements, and attempts to uncover discrepancies between the architecture and the implementation even before hands-on testing begins.

   The Certification Group, in cooperation with Architecture Group, will decide the LEVEL that the product must be tested against based on the list of implemented functions contained in the Product Specification.

2. Design Tests

3. Run Tests

4. Report Results

   Certification Reports list deviations between the architecture and the implementation. Interim reports are produced periodically, and a final certification report is published at the end of the testing. It is the final certification report that is used as the input to the Waiver Process.

5. Chair the Waiver Panel

   The Certification Group conducts Waiver Panel meetings, seeks resolution of compatibility issues, obtains commitments for any proposed changes, and publishes the results.

1.8.2 The Waiver Process

1.8.2.1 General Information - The Terminals & Workstations Certification Waiver Panel is composed of three people: one representative from Product Development, one from Architecture, and a non-voting chair person from Certification. Others may attend the meetings of the Waiver Panel and can participate in the discussions, but they have no vote.

The Waiver Panel is responsible for reviewing the deviations between the architecture and implementation as documented in the Certification Report, for assessing the technical and business impact of every deviation, and for recommending an appropriate resolution for every deviation.

1.8.3 Resolution Of Open Issues

There are six possible recommendations that the Waiver Panel will make for each level of certification under consideration:

1. Change The Product:

   It is agreed that the problem must be fixed, but that because plans are in place to do so by a specified time, a waiver is granted.

2. Change The Architecture:

   It is agreed that the product was implemented in a reasonable fashion and the Architecture may need clarification.

3. Change The Certification Procedure:

   It is agreed that the certification process was in error in that it either failed to properly interpret the architecture or failed to properly test the product.

4. Waive By Agreement of The Panel:

   It is agreed that no change is to be made at this time to the product, the architecture or the certification procedures. By granting this waiver, the Panel is expressing the opinion that it considers the issue insufficient cause for the product to fail certification.

5. Restrict:

   It is agreed that although no change to the product is involved at this time, the deviation is serious enough to warrant placing a restriction on what compatibility or other claims the product should be allowed to make.

6. No Resolution:

   The Waiver Panel was unable to agree on a resolution for the issue or on the proposed time frame for resolution, and will present it to upper management for a decision.

1.8.3.1 Results Of The Waiver Process - The Waiver Panel will determine a final disposition for each level of certification under consideration. Accordingly, the results for a particular level can be:

- PASSED CERTIFICATION
- PASSED CERTIFICATION WITH WAIVERS - No category 5 or category 6 deviations
- PASSED CERTIFICATION WITH RESTRICTION - No category 6 deviations, one or more category 5 deviations
- FAILED CERTIFICATION - One or more category 6 deviations

In addition, the Waiver Panel may wish to state any other restrictions which it agrees should be placed on the product.

1.9 CHANGE HISTORY

1.9.1 Revision 0.0 To AX10

1. This section was merged with the introductory section, the remainder of which now forms section 0 of the standards.

2. Removed DELETE as a required Level 1 function. (It is ignored as an unimplemented C1 control. Its use as a pad character should be discouraged in favor of NUL).

3. Removed Answerback (ENQ) and Test (DECTST) as required functions for both Level 1 and Level 2 conformance. They will be included as an appendix to the SRM.

4. Added VT125-type Media Copy (MC) controls to the ReGIS extension requirements for Levels 1 and 2.

5. Removed Hard Copy (DECHCP) as a requirement for the sixel extension in Levels 1 and 2. The only required way to access the sixel format is through the ReGIS S(H) command.

6. Removed Control Representation Mode (CRM) as a Level 2 requirement. CRM will be defined in an appendix to the SRM.

7. Added Insert Character (ICH) and Erase Character (ECH) as Level 2 requirements.

8. Added Selectively Erasable Characters as a Level 2 Extension.

9. Added Device Status Report (DSR) to the UDK extension for Level 2. DSR is used to return the lock conditions of the User Defined Keys.

10. Made IRM, DL, IL, and DCH part of an Editing Extension to Level 1 that is required in all future products (and in Level 2).

11. Added a complete description of the Certification process and the waiver procedure.

Section Index

-A-

-C-

VIDEO SYSTEMS REFERENCE MANUAL

Specification Program Structure

Document Identifier: A-DS-EL00070-02-0 Rev. AX11, 18-Mar-1985

ABSTRACT: This document describes the algorithmic specification language used in the Video Systems Reference Manual and the program flow across sections of the Manual.

APPLICABILITY: SARA Formal Cross-Component Standard. Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the section "Concepts and Conformance Criteria".

STATUS: FOR REVIEW ONLY

+----------+
| This document has been placed in the SARA "Formal Cross-Component Standard" category. The material contained within this document is assumed to define mandatory standards unless it is clearly marked as (a) not mandatory or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements. The procedures for modifying or evolving this standard are contained within the contents of this document. |
+----------+

+----------+
| FOR INTERNAL USE ONLY |
+----------+

TITLE: VIDEO SYSTEMS - Specification Program Structure

DOCUMENT IDENTIFIER: A-DS-EL00070-02-0 Rev. AX11, 18-Mar-1985

REVISION HISTORY: Original Draft 25-Oct-1982
Revision AX01 28-Feb-1983
Revision AX10 15-May-1984

FILES: User Documentation EL070S2.mem

Internal Documentation EL070S2.rno
EL070S2.rnt
EL070S2.rnx

Document Management Group: Terminal Interface Architecture
Responsible Department: Video Architecture Group
Responsible Individual: Peter Sichel

ACCEPTANCE: This document has been approved by the Manager of the Video Architecture Group based on a comprehensive review of its individual sections by the members of the SARA component groups working Terminal Interface Architecture issues. The list of individuals on the review and approval list are on file in Standards and Methods Control.

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, RANI::VIDARCH

Copies of this document can be ordered from:

Standards and Methods Control
APO1/F7, DTN 289-1414, JOKUR::SIMONETTI

CONTENTS

2.1 INTRODUCTION

This section contains information regarding the coding methods used within the Video Systems Reference Manual to specify the data types for terminal state information, and to describe the control operations which affect the state of the device. It also describes the program structure across sections of the manual, and includes the top level routines which bind the program sections together. It should serve as a reference point for understanding the relationship of the detailed algorithms which appear throughout the specifications. However, it is not intended to be a complete or thorough implementation of the architecture as described herein.

2.1.1 Algorithmic Specification

The coding of algorithms within this specification is intended to serve as a guideline to implementors, and as a clear specification of the interfaces rules against which product certification can be performed. The code is optimized for architectural clarity. It is intended to be as efficient as possible, but it is recognized that specific implementations may alter the algorithms or program structure in order to optimize for memory or execution speed. They must do so, however, without sacrificing the functionality or rules provided by the algorithms.

In other words, for all possible received interchange, conforming implementation algorithms shall produce identical results to the program specifications in terms of the presentation on the visual display and the state variables that an application process can solicit from the terminal.

2.1.2 Use Of Pascal

Pascal was selected as an architectural implementation language because of its rigid structure and type checking, simplicity, and wide usage both inside and outside of DEC. Standard Pascal has been used throughout, with the following exceptions:

- underline is allowable in user-defined symbols
- forward declaration of procedures and functions is implied
- OTHERWISE is used as an exception condition for case statements

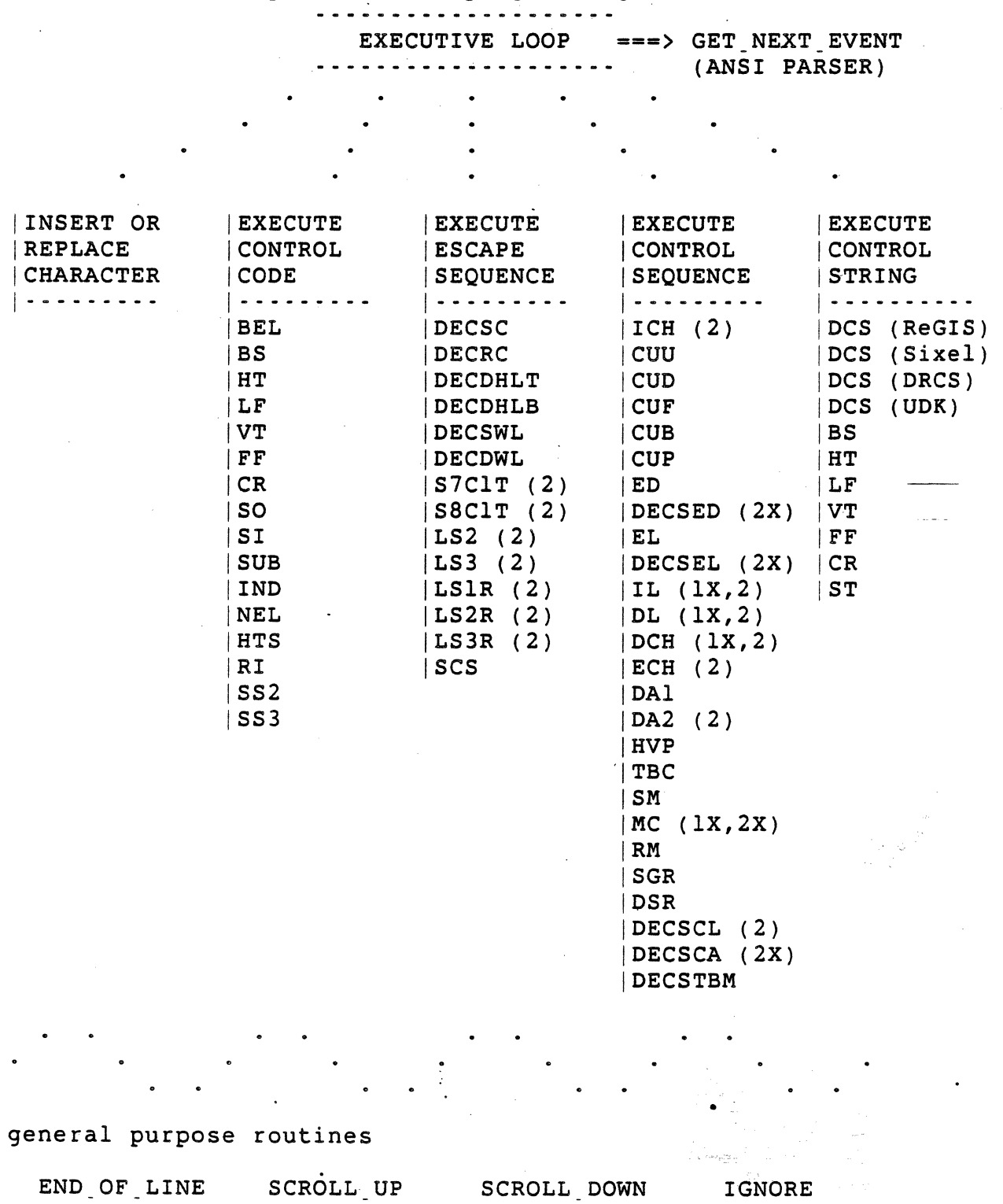
Implementation Notes:

1. Because PACKED ARRAYS are provided by Pascal as an implementation technique, to optimize storage requirements at the expense of execution speed, they are not utilized within this specification.

2. The internal procedure IGNORE is a null operation, which is used throughout the specification in order to leave as little room as possible for interpretation of undefined states of the device.

2.2 SPECIFICATION PROGRAM STRUCTURE

The following diagram is provided as an overview to the structure of the coded algorithms within this manual. It is not intended as a detailed description of the program logic.



Program Flow Diagram

2.3 SUMMARY OF STATE

2.3.1 Code Extension Layer - Parsing

```
-----------------------------------------------------------------

CONST   MAX_NUM_INTERMEDIATES = 3;
        MAX_NUM_PARAMETERS = 16;
        MAX_PARAMETER_VALUE = 16384;
        RANGE_PRIVATES = 2;

TYPE    EVENT_TYPE = (GRAPHIC_CODE,CONTROL_CODE,ESCAPE_SEQUENCE,
          CONTROL_SEQUENCE,DCS_INTRODUCER);

        CODE_TYPE = 0..255;          (* legal 8-bit code values *)
        INTERMEDIATE_TYPE = 32..47; (* legal intermediate values
*)
        PARAMETER_TYPE = 0..MAX_PARAMETER_VALUE;
        ROW_COLUMN_TYPE = 0..15;

        SEQUENCE_TYPE = (ESCAPE,CONTROL,DCS);

        EVENT_BUFFER_TYPE = RECORD
          EVENT_CODE: EVENT_TYPE;
          CODE_VALUE: CODE_TYPE;
          INTERMEDIATE_COUNT: 0..MAX_NUM_INTERMEDIATES;
          INTERMEDIATES:
            ARRAY [1..MAX_NUM_INTERMEDIATES] OF INTERMEDIATE_TYPE;
          PARAMETER_COUNT: 0..MAX_NUM_PARAMETERS;
          PARAMETERS:
            ARRAY [1..MAX_NUM_PARAMETERS] OF PARAMETER_TYPE;
          PRIVATE_PARAMETER: 0..RANGE_PRIVATES;
          VALID_SYNTAX: BOOLEAN;
          (* The following values are internal state variables
             required by the parser. Note that the boolean flag
             SEQUENCE_IN_PROGRESS must be initialized to FALSE
             by the calling routine prior to the first time the
             parser is invoked. *)
          SEQUENCE_IN_PROGRESS: BOOLEAN;
          KIND_OF_SEQUENCE: SEQUENCE_TYPE;
          PARAMETER_OVERFLOW: BOOLEAN;
          END;


VAR     EVENT: EVENT_BUFFER_TYPE;
```

2.3.2 Code Extension Layer - Graphics

```
TYPE    GRAPHIC_CHARACTER_SET_TYPE =
          (ASCII_G,SUPPLEMENTAL_G,LINE_DRAWING,  (* standard *)
          JIS_ROMAN,JIS_KATAKANA,                (* extension *)
          DRCS);                                 (* extension *)

        C0_CONTROL_CHARACTER_SET_TYPE = (ASCII_C);
        C1_CONTROL_CHARACTER_SET_TYPE = (SUPPLEMENTAL_C);

        DESIGNATED_GRAPHIC_SET_TYPE = (G0,G1,G2,G3);

        IN_USE_TABLE_TYPE = RECORD
          C0: C0_CONTROL_CHARACTER_SET_TYPE;
          GL: GRAPHIC_CHARACTER_SET_TYPE;
          INVOKED_GL: DESIGNATED_GRAPHIC_SET_TYPE;
          C1: C1_CONTROL_CHARACTER_SET_TYPE;
          GR: GRAPHIC_CHARACTER_SET_TYPE;
          INVOKED_GR: DESIGNATED_GRAPHIC_SET_TYPE;
          END;

        ENVIRONMENT_TYPE = (SEVEN_BIT,EIGHT_BIT);


VAR     IN_USE_TABLE: IN_USE_TABLE_TYPE;
        DESIGNATED_GRAPHIC_SETS:
          ARRAY [G0..G3] OF GRAPHIC_CHARACTER_SET_TYPE;

        HOST_PORT_ENVIRONMENT: ENVIRONMENT_TYPE;
        C1_TRANSMISSION_MODE: ENVIRONMENT_TYPE;
        SINGLE_SHIFT: (NONE, SS2, SS3);
```

2.3.3 Terminal Management

```
CONST   MAX_NUM_OPTIONS =      (* defined by implementation *);
        MAX_POSITION = 20      (* Level 1 *);
                       30      (* Level 2 *);
TYPE    EXTENSION_TYPE =
          (ONE_THIRTY_TWO_COLUMN,PRINTER_PORT_EXT,
          REGIS_GRAPHICS,SIXEL_GRAPHICS,KATAKANA_EXT,
          SELECTIVELY_ERASABLE_CHARACTERS,DRCS_EXT,UDK_EXT);

VAR     SERVICE_CLASS: INTEGER;
        CONFORMANCE_LEVEL: (LEVEL_1,LEVEL_2);
        LEVEL_1_EXTENSIONS:
          ARRAY [ONE_THIRTY_TWO_COLUMN..KATAKANA_EXT] OF BOOLEAN;
        LEVEL_2_EXTENSIONS:
          ARRAY [ONE_THIRTY_TWO_COLUMN..UDK_EXT] OF BOOLEAN;
        PRODUCT_ID: INTEGER;
        REVISION_LEVEL: INTEGER;
        PRODUCT_OPTIONS: ARRAY [0..MAX_NUM_OPTIONS] OF INTEGER;

        ANSWERBACK_MESSAGE_BUFFER:
          ARRAY [1..MAX_POSITION] OF CODE_TYPE;

        DEVICE_STATUS: (READY,MALFUNCTION);

        POWER_ON: BOOLEAN;
        IN_CONTROL_STRING: BOOLEAN;
```

2.3.4 Character Cell Display

```
CONST   MAX_NUM_LINES = 24;
        MAX_NUM_COLUMNS = 132;

        EMPTY_CHARACTER = 0;

        MAX_NUM_CHARACTER_SETS = 4; (* for Level 2 with DRCS *)

TYPE    LINE_TYPE = 1..MAX_NUM_LINES;
        COLUMN_TYPE = 1..MAX_NUM_COLUMNS;

        CHARACTER_POSITION_TYPE = RECORD
          LINE: LINE_TYPE;
          COLUMN: COLUMN_TYPE;
          END;

        GRAPHIC_RENDITION_TYPE =
          (BOLD,BLINK,UNDERSCORE,REVERSE);

        CHARACTER_RENDITION_TYPE =
          ARRAY [GRAPHIC_RENDITION_TYPE] OF BOOLEAN;

        LOGICAL_ATTRIBUTE_TYPE = (SELECTIVELY_ERASABLE);

        CHARACTER_ATTRIBUTE_TYPE =
          ARRAY [LOGICAL_ATTRIBUTE_TYPE] OF BOOLEAN;

        LINE_RENDITION_TYPE =
          (DOUBLE_HEIGHT_TOP,DOUBLE_HEIGHT_BOTTOM,
          SINGLE_WIDTH,DOUBLE_WIDTH);

        CHARACTER_TYPE = RECORD
          CODE: 0..126; (* 0 is used for empty characters *)
          RENDITION: CHARACTER_RENDITION_TYPE;
          ATTRIBUTE: CHARACTER_ATTRIBUTE_TYPE;
          CHARACTER_SET: GRAPHIC_CHARACTER_SET_TYPE;
          END;

        CHARACTER_SET_DESIGNATOR_TYPE = RECORD
          NAME: GRAPHIC_CHARACTER_SET_TYPE;
          FIRST_INTERMEDIATE: 0..47; (* 0 = no intermediate *)
          SECOND_INTERMEDIATE: 0..47;
          FINAL: 0..126; (* 0 indicates no table entry *)
          END;

        ORIGIN_MODE_TYPE = (ABSOLUTE,DISPLACED);
```

```
        SAVE_BUFFER_TYPE = RECORD
          POSITION: CHARACTER_POSITION_TYPE;
          RENDITION: CHARACTER_RENDITION_TYPE;
          ORIGIN_MODE: ORIGIN_MODE_TYPE;
          LEFT: GRAPHIC_CHARACTER_SET_TYPE;
          RIGHT: GRAPHIC_CHARACTER_SET_TYPE;
          SETS: ARRAY [G0..G3] OF GRAPHIC_CHARACTER_SET_TYPE;
          ATTRIBUTE: CHARACTER_ATTRIBUTE_TYPE;
          END;


VAR     DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
        LINE_RENDITION: ARRAY [LINE_TYPE] OF LINE_RENDITION_TYPE;
        ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
        TOP_MARGIN: LINE_TYPE;
        BOTTOM_MARGIN: LINE_TYPE;
        CURSOR_SAVE_BUFFER: SAVE_BUFFER_TYPE;
        CURRENT_RENDITION: CHARACTER_RENDITION_TYPE;
        CURRENT_ATTRIBUTE: CHARACTER_ATTRIBUTE_TYPE;
        CHARACTER_SET_TABLE:
          ARRAY [1..MAX_NUM_CHARACTER_SETS] OF
          CHARACTER_SET_DESIGNATOR_TYPE;
        HORIZONTAL_TAB_STOPS: ARRAY [COLUMN_TYPE] OF BOOLEAN;
        COLUMN_MODE: (EIGHTY,ONE_THIRTY_TWO);
        SCROLLING_MODE: (JUMP,SLOW);
        SCREEN_MODE: (NORMAL_SCREEN,REVERSE_SCREEN);
        ORIGIN_MODE: ORIGIN_MODE_TYPE;
        AUTO_WRAP_MODE: (WRAP_OFF,WRAP_ON);
        TEXT_CURSOR_ENABLE_MODE: (TEXT_CURSOR_OFF,TEXT_CURSOR_ON);
        INSERT_REPLACEMENT_MODE: (REPLACE,INSERT);
        NEW_LINE_MODE: (NEW_LINE_OFF,NEW_LINE_ON);
```

2.3.5 Keyboard Processing

```
VAR     KEYBOARD_ACTION_MODE: (UNLOCKED,LOCKED);
        AUTO_REPEAT_MODE: (REPEAT_OFF,REPEAT_ON);
        CURSOR_KEY_MODE: (CURSOR,CK_APPLICATION);
        KEYPAD_MODE: (NUMERIC,KP_APPLICATION);

        KEYBOARD_USAGE_MODE: (TYPEWRITER,DATA_PROCESSING);
        CAPS_SHIFT_LOCK_MODE: (CAPS_LOCK,SHIFT_LOCK);
        HOLD_SCREEN_MODE: (HOLD_OFF,HOLD_ON);
        LOCK_MODE: (LOCK_OFF,LOCK_ON);
        COMPOSE_MODE: (COMPOSE_OFF,COMPOSE_ON);
        KEYCLICK_MODE: (KEYCLICK_OFF,KEYCLICK_ON);
```

2.3.6 Printer Port Extension

```
VAR     PRINTER_PORT_ENVIRONMENT: ENVIRONMENT_TYPE;
        PRINTER_STATUS: (PRINTER_READY,PRINTER_NOT_READY,
          PRINTER_NOT_PRESENT);
        PRINTER_STYLE: (ASCII_ONLY,ASCII_PLUS,FULL_8_BIT);
        PRINTER_CONTROLLER_MODE:
          (PRINTER_CONTROLLER_OFF,PRINTER_CONTROLLER_ON);
        AUTO_PRINT_MODE: (AUTO_PRINT_OFF,AUTO_PRINT_ON);
        PRINT_FORM_FEED_MODE: (PRINT_FF_OFF,PRINT_FF_ON);
        PRINT_EXTENT_MODE: (PRINT_SCROLLING_REGION,PRINT_DISPLAY);
```

2.3.7 Extensions

```
VAR     GRAPHICS_CURSOR_ENABLE_MODE:
          (GRAPHICS_CURSOR_ON,GRAPHICS_CURSOR_OFF);

        REGIS_GRAPHICS_MODE: (REGIS_OFF,REGIS_ON);

        SIXEL_GRAPHICS_MODE: (SIXEL_OFF,SIXEL_ON);

        LOAD_DRCS_MODE: (DRCS_LOAD_OFF,DRCS_LOAD_ON);

        LOAD_UDK_MODE: (UDK_LOAD_OFF,UDK_LOAD_ON);
```

2.4 EXECUTIVE PROCEDURES

2.4.1 Executive Loop

Purpose: Describe the power on state.

Description: When power is applied to the terminal all state is initialized to its default values. Input and output buffers are cleared and interrupts are enabled. The terminal then enters a loop in which it makes repeated calls to the parsing routine, executing events as they occur. This process continues until power is turned off.

State Affected: None

Algorithm:

```
PROCEDURE EXECUTIVE_LOOP;
BEGIN
POWER_ON:= TRUE;
INITIALIZE_TERMINAL;
INITIALIZE_PARSER (EVENT);
WHILE POWER_ON DO
  BEGIN
  GET_NEXT_EVENT (EVENT);
  IF NOT IN_CONTROL_STRING THEN
    BEGIN
    (* check for single shifts *)
    SINGLE_SHIFT:= NONE;
    WHILE (EVENT.EVENT_CODE = CONTROL_CODE) AND
      (EVENT.CODE_VALUE = 142) OR (EVENT.CODE_VALUE = 143) DO
      BEGIN
      CASE EVENT.CODE_VALUE OF
        142: SINGLE_SHIFT_TWO;
        143: SINGLE_SHIFT_THREE;
        END;
      END;
    (* then process the function *)
    CASE EVENT.EVENT_CODE OF
      GRAPHIC_CODE: INSERT_OR_REPLACE_CHARACTER;
      CONTROL_CODE: EXECUTE_CONTROL_CODE;
      ESCAPE_SEQUENCE: IF EVENT.VALID_SYNTAX THEN
        EXECUTE_ESCAPE_SEQUENCE;
      CONTROL_SEQUENCE: IF EVENT.VALID_SYNTAX THEN
        EXECUTE_CONTROL_SEQUENCE;
      DCS_INTRODUCER: IF EVENT.VALID_SYNTAX THEN
        EXECUTE_DCS_INTRODUCER_SEQUENCE;
      OTHERWISE IGNORE;
```

```
      END;
    END
  ELSE (* in control string *)
    BEGIN
    IF (EVENT.EVENT_CODE = CONTROL_CODE)
      AND (EVENT.CODE_VALUE = 156) THEN (* String Terminator *)
        IN_CONTROL_STRING:= FALSE
    ELSE
      BEGIN
      IF REGIS_GRAPHICS_MODE = REGIS_ON THEN
        (* pass all events to ReGIS parser for processing *)
        EXECUTE_REGIS (EVENT);
      IF SIXEL_GRAPHICS_MODE = SIXEL_ON THEN
        (* pass all events to sixel parser for processing *)
        EXECUTE_SIXEL (EVENT);
      IF LOAD_DRCS_MODE = DRCS_LOAD_ON THEN
        (* pass all events to DRCS load routine for processing *)
        LOAD_DRCS (EVENT);
      IF LOAD_UDK_MODE = UDK_LOAD_ON THEN
        (* pass all events to UDK load routine for processing *)
        LOAD_UDK (EVENT);
      END;
    END;
  END;
END;
```

2.4.2 Event Handling Tables

2.4.2.1 Execute Control Code -

Purpose: Execute valid control codes.

Description: When a control code is received by the terminal it is immediately dispatched to the appropriate action routine by this table of 8-bit code values.

State Affected: None

Algorithm:

```
PROCEDURE EXECUTE_CONTROL_CODE;
BEGIN
CASE EVENT.CODE_VALUE OF
  7: WARNING_BELL;
  8: BACK_SPACE;
  9: HORIZONTAL_TAB;
  10: LINE_FEED;
  11: VERTICAL_TAB;
  12: FORM_FEED;
  13: CARRIAGE_RETURN;
  14: SHIFT_OUT; (* Locking Shift One *)
  15: SHIFT_IN; (* Locking Shift Zero *)
  26: SUBSTITUTE;
  132: INDEX;
  133: NEXT_LINE;
  136: HORIZONTAL_TABULATION_SET;
  141: REVERSE_INDEX;
  157: IN_CONTROL_STRING:= TRUE; (* Operating System Command *)
  158: IN_CONTROL_STRING:= TRUE; (* Privacy Message *)
  159: IN_CONTROL_STRING:= TRUE; (* Application Program Command *)
  OTHERWISE IGNORE;
  END;
END;
```

Known Deviations: None

2.4.2.2 Execute Escape Sequence -

Purpose: Execute valid escape sequences.

Description: Escape sequences exhibiting valid syntax on termination are acted upon immediately upon receipt of the final character. The parser returns the final character code as well as the count of code values of any intermediate characters received during processing of the sequence.

State Affected: None

Algorithm:

```
PROCEDURE EXECUTE_ESCAPE_SEQUENCE;
VAR     N: INTEGER;
BEGIN
IF EVENT.INTERMEDIATE_COUNT = 0 THEN
  BEGIN
  CASE EVENT.CODE_VALUE OF
    55: SAVE_CURSOR;
    56: RESTORE_CURSOR;
    110: IF CONFORMANCE_LEVEL = LEVEL_2 THEN LOCKING_SHIFT_TWO
         ELSE IGNORE;
    111: IF CONFORMANCE_LEVEL = LEVEL_2 THEN LOCKING_SHIFT_THREE
         ELSE IGNORE;
    126: IF CONFORMANCE_LEVEL = LEVEL_2 THEN
         LOCKING_SHIFT_ONE_RIGHT ELSE IGNORE;
    125: IF CONFORMANCE_LEVEL = LEVEL_2 THEN
         LOCKING_SHIFT_TWO_RIGHT ELSE IGNORE;
    124: IF CONFORMANCE_LEVEL = LEVEL_2 THEN
         LOCKING_SHIFT_THREE_RIGHT ELSE IGNORE;
    OTHERWISE IGNORE;
    END;
  END
ELSE
  BEGIN
  CASE EVENT.INTERMEDIATE_COUNT OF
    1: (* three character escape sequences *)
      BEGIN
      CASE EVENT.INTERMEDIATES[1] OF
        35: (* #, 2/3 *)
          BEGIN
          CASE EVENT.CODE_VALUE OF
            51: DOUBLE_HEIGHT_LINE_TOP;
            52: DOUBLE_HEIGHT_LINE_BOTTOM;
            53: SINGLE_WIDTH_LINE;
            54: DOUBLE_WIDTH_LINE;
            OTHERWISE IGNORE;
```

```
            END;
          END;
        32: (* SP, 2/0 *)
          BEGIN
          CASE EVENT.CODE_VALUE OF
            70: IF CONFORMANCE_LEVEL = LEVEL_2 THEN
                SELECT_7_BIT_C1_TRANSMISSION ELSE IGNORE;
            71: IF CONFORMANCE_LEVEL = LEVEL_2 THEN
                SELECT_8_BIT_C1_TRANSMISSION ELSE IGNORE;
            OTHERWISE IGNORE;
            END;
          END;
        40,41,42,43:
          FOR N:= 1 TO 4 DO
            IF EVENT.CODE_VALUE =
            CHARACTER_SET_TABLE[N].FINAL THEN
              DESIGNATE_CHARACTER_SET;
        OTHERWISE IGNORE;
        END;
      END;
    2: (* four character escape sequences *)
      BEGIN
      IF CONFORMANCE_LEVEL = LEVEL_2 THEN
        BEGIN
        CASE EVENT.INTERMEDIATES[1] OF
          40,41,42,43:
            IF (EVENT.INTERMEDIATES[2] =
            CHARACTER_SET_TABLE[4].SECOND_INTERMEDIATE)
            AND (EVENT.CODE_VALUE =
            CHARACTER_SET_TABLE[4].FINAL) THEN
              DESIGNATE_CHARACTER_SET;
          OTHERWISE IGNORE;
          END;
        END
      ELSE IGNORE;
      END;
    OTHERWISE IGNORE;
    END;
  END;
END;
```

Known Deviations: None

2.4.2.3 Execute Control Sequence -

Purpose: Execute valid control sequences.

Description: Control sequences exhibiting valid syntax on termination are acted upon immediately upon receipt of the final character. The parser returns the final character code as well as the count of code values of any parameters and/or intermediate characters received during processing of the sequence. It also indicates whether a private parameter value was received.

State Affected: None

Algorithm:

```
PROCEDURE EXECUTE_CONTROL_SEQUENCE;
BEGIN
IF EVENT.INTERMEDIATE_COUNT = 0 THEN
  BEGIN
  CASE EVENT.CODE_VALUE OF
    64: IF CONFORMANCE_LEVEL = LEVEL_2 THEN INSERT_CHARACTER;
    65: CURSOR_UP;
    66: CURSOR_DOWN;
    67: CURSOR_FORWARD;
    68: CURSOR_BACKWARD;
    72: CURSOR_POSITION;
    74: ERASE_IN_DISPLAY; (* also Selective Erase In Display *)
    75: ERASE_IN_LINE;    (* also Selective Erase In Line *)
    76: INSERT_LINE;
    77: DELETE_LINE;
    80: DELETE_CHARACTER;
    88: IF CONFORMANCE_LEVEL = LEVEL_2 THEN ERASE_CHARACTER;
    99: DEVICE_ATTRIBUTES;
   102: HORIZONTAL_VERTICAL_POSITION;
   103: TABULATION_CLEAR;
   104: SET_MODE;
   105: MEDIA_COPY;
   108: RESET_MODE;
   109: SELECT_GRAPHIC_RENDITION;
   110: DEVICE_STATUS_REPORT;
   112: IF (EVENT.INTERMEDIATE_COUNT = 1) AND
        (EVENT.INTERMEDIATES[1] = 34) THEN
        SELECT_CONFORMANCE_LEVEL;
   113: IF (CONFORMANCE_LEVEL = LEVEL_2) AND
        (LEVEL_2_EXTENSIONS[SELECTIVELY_ERASABLE_CHARACTERS])
THEN
        SELECT_CHARACTER_ATTRIBUTE;
   114: SET_TOP_AND_BOTTOM_MARGINS;
   OTHERWISE IGNORE;
```

```
    END;
  END
ELSE
  (* no intermediates defined yet for control sequences *)
  IGNORE;
END;
```

Known Deviations: None

2.4.2.4 Execute Device Control String -

Purpose: Execute valid device control strings.

Description: Device Control String Introducer sequences exhibiting valid syntax on termination are acted upon immediately upon receipt of the final character. The parser returns the final character code as well as the count of code values of any parameters and/or intermediate characters received during processing of the sequence. It also indicates whether a private parameter value was received.

State Affected: None

Algorithm:

```
PROCEDURE EXECUTE_DCS_INTRODUCER_SEQUENCE;
BEGIN
IN_CONTROL_STRING:= TRUE;
IF EVENT.INTERMEDIATE_COUNT = 0 THEN
  BEGIN
  CASE EVENT.CODE_VALUE OF
    112: REGIS_GRAPHICS_MODE:= REGIS_ON;
    113: SIXEL_GRAPHICS_MODE:= SIXEL_ON;
    123: LOAD_DRCS_MODE:= DRCS_LOAD_ON;
    124: LOAD_UDK_MODE:= UDK_LOAD_ON;
    OTHERWISE IGNORE;
    END;
  END
ELSE
  IGNORE;
  (* no intermediates defined yet for DCS introducers *)
END;
```

Known Deviations: None

2.4.3 Device Attributes

Purpose: Request device identification.

Format:	CSI	Pp	0	c	default Pp: none
	9/11	Pp	3/0	6/3	

Description: The DA control with a zero or omitted parameter prompts the terminal to return an identifying sequence in one of two forms, depending on the setting of the private parameter value, Pp. If the private parameter is omitted, the terminal will return a Primary Device Attributes response (see the section "Terminal Management" for more details on the terminal's response to this request). If the private parameter value is > (3/14) and the device is a Level 2 or higher terminal, it will return a Secondary Device Attributes response.

State Affected: None

Algorithm:

```
PROCEDURE DEVICE_ATTRIBUTES;
BEGIN
IF EVENT.PARAMETERS[1] = 0 THEN
  BEGIN
  CASE EVENT.PRIVATE_PARAMETER OF
    0: DEVICE_ATTRIBUTES_1;
    2: DEVICE_ATTRIBUTES_2;
    OTHERWISE IGNORE;
    END;
  END
ELSE IGNORE;.
END;
```

Known Deviations: None

2.4.4 Device Status Report

Purpose: Request or report the general status of the device.

Format:	CSI	Ps	n	default Ps: 0
	9/11	Ps	6/14	

Description: The DSR control is used as an inquiry of the current state of the terminal device. Three levels of request are provided: device availability, position reporting, and printer availability. If the parameter value is five (5) the device will respond with a DSR control, the parameter of which will be one of the following:

0 - Device ready, no malfunctions detected
3 - Device not ready, malfunction detected

If the parameter value is six (6) the device will respond with a Cursor Position Report (CPR) control.

Printer status requests use the private parameter value ?15 (3/15 3/1 3/5).

Requests for the lock status of User Defined Keys use the private parameter value ?25 (3/15 3/2 3/5).

Notes:

1. There is no guarantee of synchronization between the status reports transmitted by the device in response to this command and the actual device states at the time they are received by a communicating party. They are only indicators of the device state at the time the DSR control is executed by the device.

State Affected: None

Algorithm:

```
PROCEDURE DEVICE_STATUS_REPORT;
BEGIN
CASE EVENT.PRIVATE_PARAMETER OF
  0: (* not a private *)
    BEGIN
    CASE EVENT.PARAMETERS[1] OF
```

```
      5: REPORT_DEVICE_STATUS;
         (* see section "Terminal Management" *)
      6: CURSOR_POSITION_REPORT;
         (* see section "Character Cell Display" *)
      OTHERWISE IGNORE;
      END;
    END;
  1: (* private parameter '?' *)
    BEGIN
    CASE EVENT.PARAMETERS[1] OF
      15: REPORT_PRINTER_STATUS;
          (* see section "Printer Port Extension *)
      25: REPORT_UDK_LOCK_STATUS;
          (* see section "UDK Extension *)
      OTHERWISE IGNORE;
      END;
    END;
  OTHERWISE IGNORE;
  END;
END;
```

Known Deviations: None

2.4.5 Set And Reset Modes

2.4.5.1 Set Mode -

Purpose: Place one or more mode values in the set state.

Format:

	CSI	Ps ; ... ; Ps	h	default Ps: none
	9/11	Ps ; ... ; Ps	6/8	

Description: The SM control provides a means of placing one or more of the terminal mode values into the set state.

Notes:

1. The parameter list sent with a single instance of this control should not exceed the maximum allowable for the parser implementation. (Note: for a detailed description of the parser, see the section "Code Extension Layer".)

Algorithm:

```
PROCEDURE SET_MODE;
VAR     N: INTEGER;
BEGIN
IF EVENT.PARAMETER_COUNT = 0 THEN EVENT.PARAMETER_COUNT:= 1;
(* check all parameters *)
FOR N:= 1 TO EVENT.PARAMETER_COUNT DO
  BEGIN
  (* check private parameter '?' (3/15) *)
  IF EVENT.PRIVATE_PARAMETER = 1 THEN
    BEGIN
    CASE EVENT.PARAMETERS[N] OF
      1: SET_CURSOR_KEY_MODE;
      3: SET_COLUMN_MODE;
      4: SET_SCROLLING_MODE;
      5: SET_SCREEN_MODE;
      6: SET_ORIGIN_MODE;
      8: SET_AUTO_REPEAT_MODE;
      18: SET_PRINT_FORM_FEED_MODE;
      19: SET_PRINT_EXTENT_MODE;
      25: IF CONFORMANCE_LEVEL = LEVEL_2 THEN
          SET_TEXT_CURSOR_ENABLE_MODE;
      OTHERWISE IGNORE;
      END;
    END
  (* check standard parameters *)
```

```
  ELSE IF EVENT.PRIVATE_PARAMETER = 0 THEN
    BEGIN
    CASE EVENT.PARAMETERS[N] OF
      2: SET_KEYBOARD_ACTION_MODE;
      4: SET_INSERT_REPLACEMENT_MODE;
      20: SET_NEW_LINE_MODE;
      OTHERWISE IGNORE;
      END;
    END;
  END;
END;
```

Known Deviations: None

2.4.5.2 Reset Mode -

Purpose: Place one or mode values in the reset state.

Format:

	CSI	Ps ; ... ; Ps	1	default Ps: none
	9/11	Ps ; ... ; Ps	6/12	

Description: The RM control provides a means of placing one or more of the terminal mode values into the reset state.

Notes:

1. The parameter list sent with a single instance of this control should not exceed the maximum allowable for the parser implementation (sixteen or less is recommended for backwards compatibility).

Algorithm:

```
PROCEDURE RESET_MODE;
VAR     N: INTEGER;
BEGIN
IF EVENT.PARAMETER_COUNT = 0 THEN EVENT.PARAMETER_COUNT:= 1;
(* check all parameters *)
FOR N:= 1 TO EVENT.PARAMETER_COUNT DO
  BEGIN
  (* check private parameter '?' (3/15) *)
  IF EVENT.PRIVATE_PARAMETER = 1 THEN
    BEGIN
    CASE EVENT.PARAMETERS[N] OF
      1: RESET_CURSOR_KEY_MODE;
      3: RESET_COLUMN_MODE;
      4: RESET_SCROLLING_MODE;
      5: RESET_SCREEN_MODE;
      6: RESET_ORIGIN_MODE;
      8: RESET_AUTO_REPEAT_MODE;
      18: RESET_PRINT_FORM_FEED_MODE;
      19: RESET_PRINT_EXTENT_MODE;
      25: IF CONFORMANCE_LEVEL = LEVEL_2 THEN
          RESET_TEXT_CURSOR_ENABLE_MODE;
      OTHERWISE IGNORE;
      END;
    END
  (* check standard parameters *)
  ELSE IF EVENT.PRIVATE_PARAMETER = 0 THEN
    BEGIN
    CASE EVENT.PARAMETERS[N] OF
```

```
      2: RESET_KEYBOARD_ACTION_MODE;
      4: RESET_INSERT_REPLACEMENT_MODE;
      20: RESET_NEW_LINE_MODE;
      OTHERWISE IGNORE;
      END;
    END;
  END;
END;
```

Known Deviations: None

2.5 CHANGE HISTORY

2.5.1 Rev 0.0 To AX10

1. Removed ENQ, DECTST, and DECHCP from program flow diagram. Added ICH, ECH, DECSCA, DECSEL, and DECSED.

2. Corrected value range for Character_Type.Code from 32..126 to 0..126 to permit 0 to be used to represent empty character positions in the display state.

3. Removed Control_Representation_Mode from the state tables as well as the Set_Mode and Reset_Mode algorithms.

4. Corrected the name of ReGIS_Graphics_Mode, and added Sixel_Graphics_Mode, Load_DRCS_Mode, and Load_UDK_Mode to the state table for Extensions.

5. Made minor editorial corrections to the Executive_Loop algorithm, and added code to process Single Shifts and the Load_DRCS and Load_UDK control strings.

6. Removed ~~Ans~~werback from the Execute_Control_Code algorithm. Removed the processing of Single_Shift_Two and Single_Shift_Three from this routine.

7. Removed Hard_Copy from the Execute_Escape_Sequence algorithm, and changed the character set designator value 50 to 60.

8. Added Insert_Character, Erase_Character, and Select_Character_Attribute to the Execute_Control_Sequence algorithm. Removed DECTST.

9. Changed the name of Execute_Device_Control_String to Execute_DCS_Introducer_Sequence.

10. Changed the escape sequence handling algorithm to support the character set designation scheme for DRCS which allows designators to be redefined dynamically.

2.5.2 Rev AX10 To AX11

1. Removed Rev AX10 change bars. Added change bars to any change that could affect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

2. Modified code to test intermediate count so that Control Sequences are uniquely recognized.

3. Modified code to test intermediate count so that DCS introducer sequences are uniquely recognized.

This page deliberately left blank.

Section Index

DEC STD 070-3 VIDEO SYSTEMS STANDARD -

CODE EXTENSION LAYER

Document Identifier: A-DS-EL00070-03-0 Rev A, 14-Apr-1989

ABSTRACT: This document defines the form and syntax of the coded communication between terminal products and other system components. It applies directly to serial line communications, consisting of a stream of 7-bit or 8-bit combinations referred to as "character codes". The mechanisms employed for achieving this form of communication, however, may be applied within the context of block oriented line protocols, networking packet protocols, and program call interfaces over high-bandwidth bus interconnects.

APPLICABILITY: This document is Mandatory for Engineers designing hardware for terminal products and software engineers designing programs using terminal interfaces. Additional requirements are defined in DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria.

STATUS: APPROVED 14-Apr-1989; type $ VTX SMC for expiration date.

The material contained within this document is assumed to define mandatory standards unless it is clearly marked as: (a) not mandatory; or (b) guidelines. Material that is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques that are considered to be good practice, but should not be considered as requirements.

digital™

TITLE: DEC STD 070-3 VIDEO SYSTEMS STANDARD - CODE EXTENSION LAYER

| DOCUMENT IDENTIFIER: A-DS-EL00070-03-0 Rev A, 14-Apr-1989

REVISION HISTORY:

Original Draft	05-Jun-1982
Revision 0.1	04-Sep-1982
Revision 0.2	20-Oct-1982
Revision 0.3	08-Dec-1982
Revision AX04	28-Feb-1983
Revision AX10	15-May-1983
Revision AX11	18-Mar-1985
Revision AX12	06-Feb-1988
Revision A	14-Apr-1989

Document Management Category: Terminal Interface Architecture (STI)
Responsible Department: DSG Terminals Architecture
Responsible Person: Peter Sichel
SMC Writer: Patricia Winner

APPROVAL: This document prepared by the Desktop Systems Group has been reviewed and recommended for approval by the general review group for its category for use throughout Digital.

Peter Conklin, Technical Director,
Desktop Systems Group

Peter A. Sichel

Peter Sichel, Desktop Systems Group

Eric A Williams

Eric Williams, Standards Process Manager

| Direct requests for further information to Peter Sichel,
| DSG1-2/C7, DTN 235-8374, HANNAH::TERMARCH

| Copies of this document can be ordered from Standards and Methods
| Control, $ VTX SMC, JOKUR::SMC, DTN: 287-3724, or CTS1-2/D4.

Please supply your name, badge number, cost center, mailstop, and ENET node when ordering.

digital™

CONTENTS

3.1 INTRODUCTION

This specification defines the form and syntax of the coded interchange between terminal products and other system components. It applies directly to serial line communications consisting of a stream of 7-bit or 8-bit combinations referred to as "coded characters". The mechanisms employed for achieving this form of communication, however, may be applied within the context of block oriented line protocols, networking packet protocols, and program call interfaces over high-bandwidth bus interconnects.

3.1.1 SCOPE

The recognition and coding of characters as described in this specification applies uniformly to all display terminals, printing terminals, computing terminals and workstations, and is independent of the actual character sets implemented in a particular terminal product. The syntax defined herein applies equally to the transmission and receipt of coded interchange at the terminal interface.

3.1.2 NATIONAL REPLACEMENT CHARACTER SET (NRCS) EXTENSION

Support for National Replacement Character Sets was introduced as an extension to the VT200 family of Digital terminals for backward compatibility with national versions of the VT100 family. The NRCS extension allows U.S. ASCII to be replaced by one of 12 NRC sets corresponding to the keyboard dialect selected (when the keyboard dialect is other than "North American"). These 7-bit character sets are similar to U.S. ASCII, but replace certain reserved character positions with national use characters. The National Replacement Character Mode (DECNRCM) is intended for backward compatibility and restricts the terminal to 7-bit characters. New applications are encouraged to use one of the 8-bit multinational character sets. See "Character Set Mode" in DEC STD 070-6 Video Systems Reference Manual - Keyboard Processing. The rest of this section of DEC STD 070 does not assume the NRCS Extension is present.

3.1.3 8-BIT INTERFACE ARCHITECTURE EXTENSION

Support for 96-character graphic character sets, ISO Latin-1 Supplemental, and the User Preference Supplemental Set (UPSS) was introduced on the VT300 family of Digital terminals. The 8-bit Interface Architecture Extension is required for Level 3 conformance to the Character Cell Display service class, and is strongly recommended (but not mandatory) for Level 2. It is also mandatory for Level 2 of the Character Cell Printer service class. The rest of this section for DEC STD 070 assumes the 8-bit Interface Architecture Extension is present.

3.1.4 RELATIONSHIP TO TERMINAL INTERFACE ARCHITECTURE (TIA)

Character encoding forms a part of the Code Extension Layer of the Terminal Interface Architecture. It is independent of any specific service class, and therefore applies across all products implementing any aspect of the architecture.

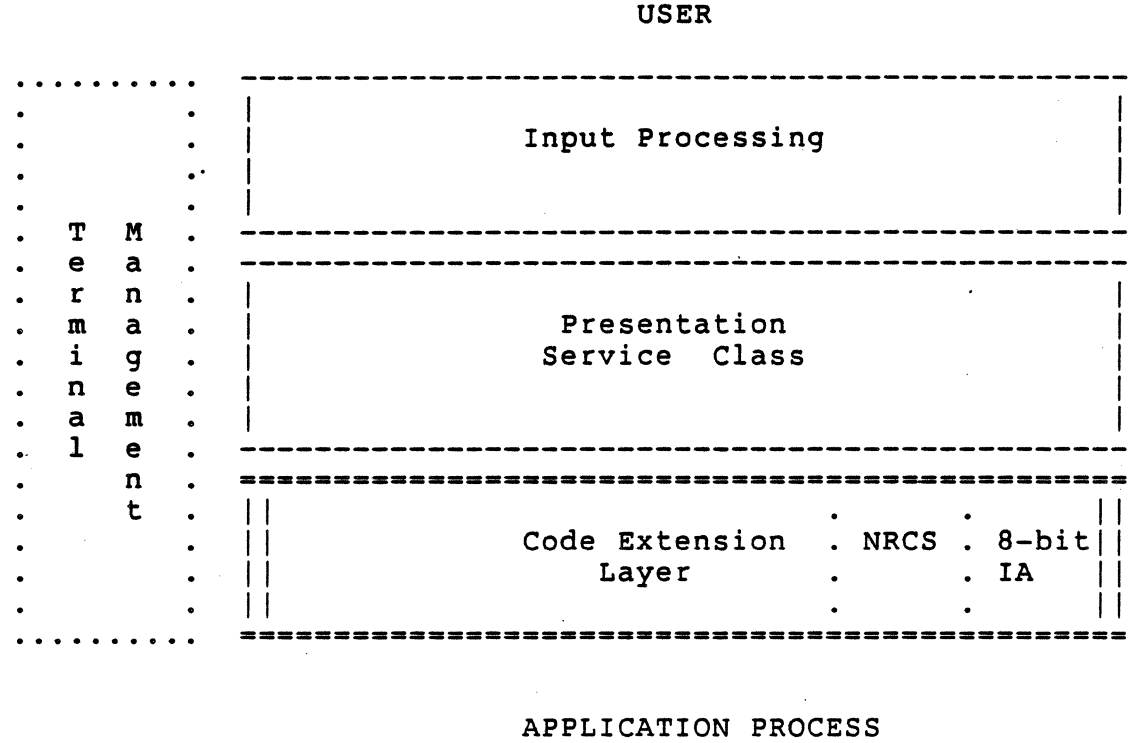
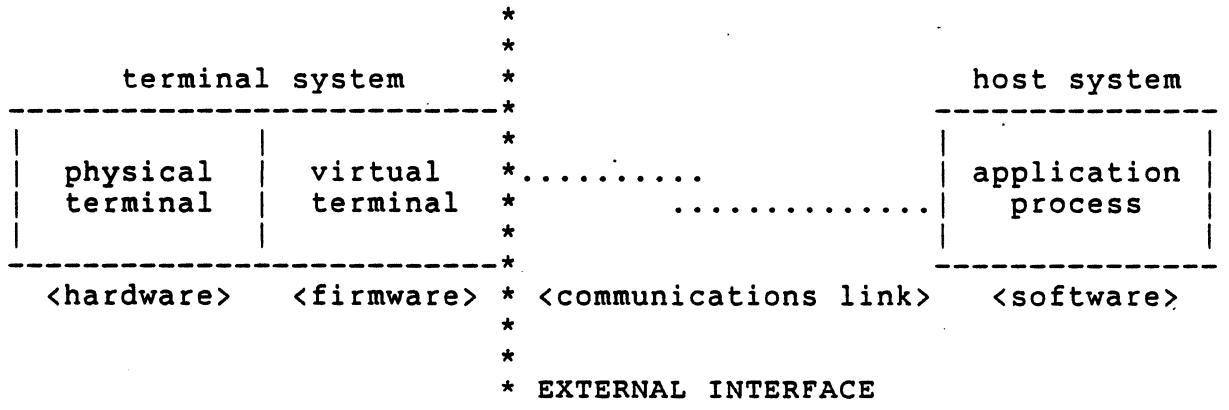


Figure 1. Structuring of the Terminal Interface Architecture

The interfaces defined within this standard apply to both internal and external product interfaces. External interfaces are interfaces between a terminal, personal computer, or workstation and a remote system. Internal interfaces are interfaces between a terminal subsystem and software processes running within a terminal, personal computer, or workstation.



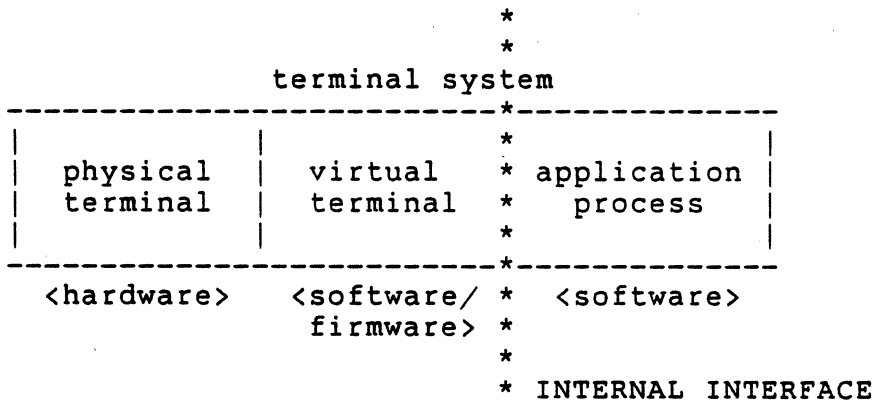


Figure 2. External and Internal Terminal Interfaces

3.2 TERMINOLOGY

bit combination - Ordered set of bits that represent a character.

byte - A 7-bit or 8-bit string that is used to represent control or graphic information. The size of the byte is independent of redundancy or framing technique. (Note: this term commonly refers to the 8-bit unit.)

character - Member of a set of elements that is used for the organization, control and representation of data.

code; Coded Character Set - Set of unambiguous rules that establishes a character set and the one-to-one relationship between the characters of the set and their bit combinations.

code extension - Techniques for the encoding of characters that are not included in the character set of a given code.

code table - Table showing the character corresponding to each bit combination in a code.

control character - Control function of which the coded representation consists of a single bit combination.

control function - Action that affects the recording, processing, transmission, or interpretation of data. The coded representation of a control function consists of one or more bit combinations.

control sequence - Bit string that is for control purposes in code extension procedures consisting of two or more bit combinations, beginning with a Control Sequence Introducer (CSI) control function, and that may contain parametric information for the control function.

control string - A string of characters that is used to perform a control function and is delimited by an opening and closing delimiter control. The opening delimiter is one of a small set of introducer control functions (APC, DCS, OSC, PM), and the closing delimiter is String Terminator (ST). Within Digital, the syntax of control strings has been extended to include an introducer sequence that acts as a protocol selector and specifies how to interpret subsequent control string data.

designate - To identify a set of characters that are to be represented, in some cases immediately and in others upon further occurrence of a control function, in a prescribed manner.

environment - The characteristic that identifies the number of bits used to represent a character in a data processing or data communication system, or in part of such a system.

escape sequence - Bit string that is for control purposes in code extension procedures consisting of two or more bit combinations, the first of which is an Escape (ESC) character.

expansion escape sequence (Fe) - 2-character escape sequence in which the final character is in columns 4 and 5, and is used to extend the 7-bit code table by being the row equivalent to the corresponding C1 set of controls in columns 8 and 9 of an 8-bit code.

final character - The character whose bit combination terminates an escape sequence, control sequence, or control string introducer sequence.

graphic character - Character, other than a control character, that has a handwritten, printed, or displayed visual representation and that has a coded representation consisting of one bit combination.

intermediate character - A character whose bit combination precedes a Final character in an escape sequence, control sequence, or control string introducer sequence.

invoke - To cause a designated set of characters to be represented by the prescribed bit combinations whenever those bit combinations occur (until an appropriate code extension function occurs).

numeric parameter - A control sequence parameter used for passing numeric values.

parameter string - A character string, the bit combination of which occurs between the Control Sequence Introducer character and the first intermediate character or final character of a control sequence.

position - Item in a code table identified by its column and row coordinates.

represent - 1) To use a prescribed bit combination with the meaning of a character in a set of characters that has been designated and invoked. 2) To use an escape sequence with the meaning of an additional control function.

selective parameter - A control sequence parameter used to select particular entries from a specified list.

3.3 CHARACTER CODING DESCRIPTION

3.3.1 CHARACTER CODES

A character code is a unique bit combination that defines a position in a coded character set by its column and row coordinate within that set. Characters may be coded using 7-bit combinations or 8-bit combinations. The choice of whether 7-bit or 8-bit character codes are used is dependent on the "width" (frame size) of the communications channel (referred to in this specification as the "environment").

If the channel is only seven bits wide, then only 7-bit character codes may be used. If the channel is eight bits wide, then either 7-bit or 8-bit character codes may be used (7-bit character codes are a specific subset of the 8-bit code structure). The following table describes how the column and row position in the coded character set is derived from both 7-bit and 8-bit character codes.

Bits of a 7-bit combination	--	b7	b6	b5	b4	b3	b2	b1
Bits of an 8-bit combination	a8	a7	a6	a5	a4	a3	a2	a1
	COLUMN				ROW			

Column and Row Reference

3.3.2 7-BIT CHARACTER SETS

A 7-bit coded character set has the following features.

a. A set of 32 control characters allocated to columns 00 and 01 (bit combinations in the range 0/0 to 1/15 inclusive).

AND EITHER

b. A character Space (SP) in position 2/0, that may be regarded as either a control character or a graphic character.

c. A set of 94 graphic characters allocated to columns 02 to 07 (bit combinations from 2/1 to 7/14, inclusive).

d. A control character Delete (DEL) in position 7/15.

OR

e. A set of 96 graphic characters allocated to columns 02 to 07 (bit combinations from 2/0 to 7/15, inclusive).

The following table describes the structure of a 7-bit coded character set including a 94-character graphic character set.

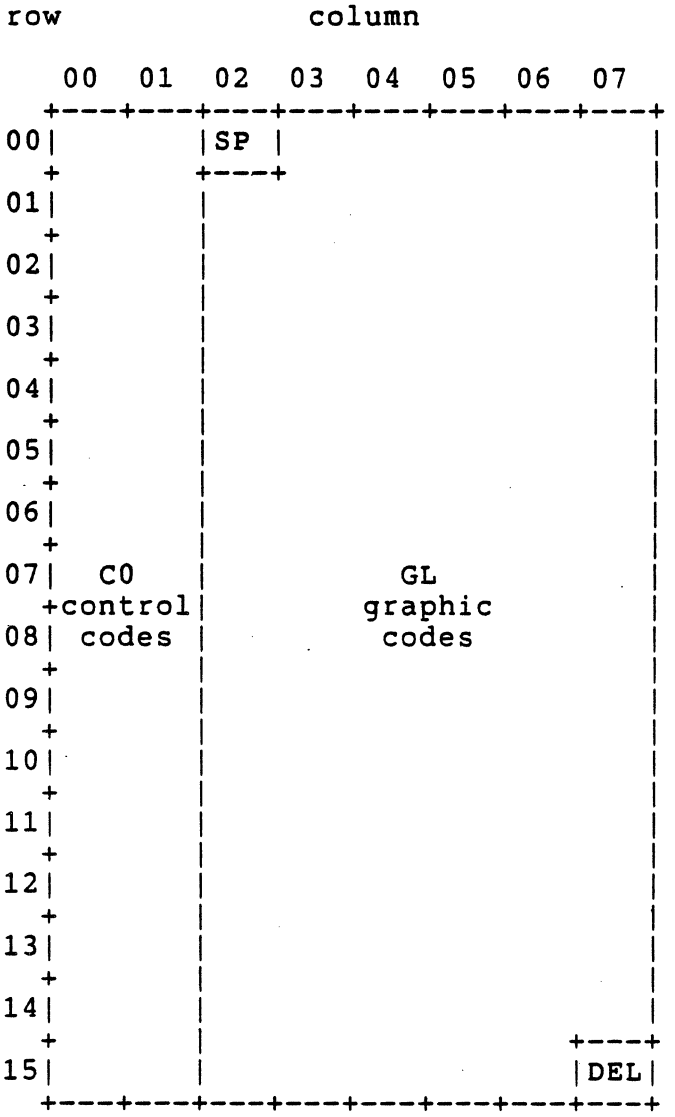


Figure 3. 7-bit Code Table Structure

3.3.3 8-BIT CHARACTER SETS

An 8-bit coded character set has the following features.

a. All the features of a 7-bit coded character set as previously listed.

b. A set of thirty-two control characters allocated to columns 08 and 09, called the C1 control set (bit combinations from 8/0 to 9/15, inclusive).

AND EITHER

c. A set of 94 graphic characters allocated to columns 10 to 15 (bit combinations from 10/1 to 15/14, inclusive), called the GR graphics set (for right-hand graphics set).

OR

d. A set of 96 graphic characters allocated to columns 10 to 15 (bit combinations from 10/0 to 15/15, inclusive).

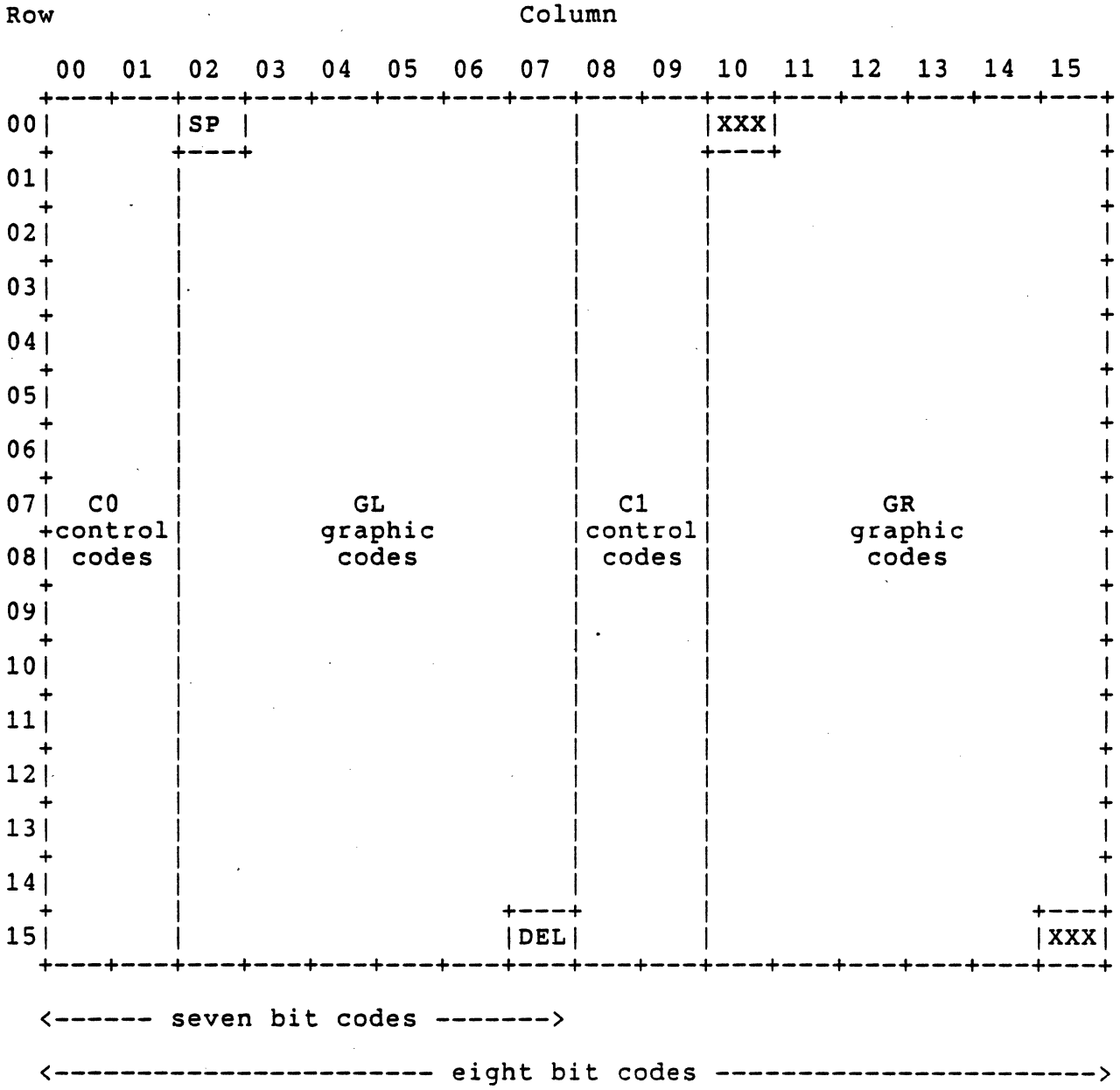
Note

2/0, 7/15, 10/0 and 15/15 are not included in C0, GL, C1, or GR for 94-character graphic sets.

2/0 and 7/15 are included in GL when a 96-character set is invoked into GL.

10/0 and 15/15 are included in GR when a 96-character set is invoked into GR.

The following table describes the structure of an 8-bit coded character set with 94-character graphic character sets.



3.3.4 CONTROL AND GRAPHIC CODES

The coded character stream can be separated into two categories: control functions and coded graphic characters. Control functions are actions that affect the recording, processing, transmission, or interpretation of data. Coded graphic characters represent actual presentation data. All bit combinations in the coded character stream are specifically categorized as being either part of a control function or a coded graphic character.

3.3.4.1 Control Characters

A control character is a single character whose occurrence in a particular context initiates, modifies, or stops a control function. The control characters are referred to as C0 and C1. DEL (delete, 7/15) is also a control character, although it is not a member of the C0 or C1 sets. SP (space, 2/0) is also not a member of C0 or C1, but it may be considered to be either a control character or a graphic character.

In a 7-bit environment, the values of a C0 control character are in the range of columns 00 and 01 of the code table. The C1 control characters are represented in 7-bit environments by 2-character escape sequences.

In an 8-bit environment, the values of C0 control characters are from columns 00 and 01 and the values of C1 control characters are from columns 08 and 09.

3.3.4.2 Graphic Characters

Graphic characters are characters other than a control character having a visual representation which is normally printed or displayed. This includes all characters from the GL and GR portions of the code table (2/1 to 7/14 inclusive and 10/1 to 15/14 inclusive for 94 character sets; 2/0 to 7/15, inclusive and 10/0 to 15/15 inclusive for 96 character sets). Graphic characters are frequently referred to as "printing characters".

Bit combinations in this same range that occur within escape or control sequences are not treated as graphic characters, but are processed as part of the control function.

3.4 SPECIAL CHARACTERS

When 94-character graphic character sets are used, the following characters are not part of either the control or graphic sets in a 7-bit or 8-bit code, and therefore have special processing as described below.

3.4.1 2/0 (SPACE)

The Space character may be interpreted as a graphic character, a control character, or both. As a graphic character it has the visual representation of the absence of a graphic symbol. The Space character will be displayed using whatever display attributes are currently active.

3.4.2 7/15 (DELETE)

When received by a terminal, the Delete character is treated as an unimplemented control character and shall be ignored. Conforming software shall not use Delete as a pad character. The Delete character may be issued as a result of keyboard processing. Refer to DEC STD 070-6 for more information.

3.4.3 10/0

The character code 10/0 in 8-bit environments is treated as an unassigned graphic character code, and will cause the error character (usually imaged as a REVERSE QUESTION MARK) to be displayed on conforming devices. Within escape sequences and control sequences, the high order bit of this character will be ignored, and it will be treated as syntactically equivalent to Space (2/0).

3.4.4 15/15

The character code 15/15 in 8-bit environments is treated as an unimplemented control code and shall be ignored.

3.5 CONTROL CODE EXTENSION TECHNIQUES

Because the number of controls typically required for effective communication far exceeds the number provided in either the 7-bit or 8-bit code tables, special methods of providing additional control functions have evolved. These methods can be divided into three categories: escape sequences, control sequences, and control strings.

3.5.1 GENERAL RULES

3.5.1.1 Precedence of Control Functions

Control characters, escape sequences, control sequences, control strings, and graphic characters are processed serially in the forward direction in the data stream with the following precedence:

1. Control characters
2. Escape sequences, control sequences, and control strings (equal precedence)
3. Graphic characters

Thus, if a control character is transmitted in the middle of an escape sequence, the control character is executed as if the escape sequence were not present (that is, as if the control character had been received immediately before the escape sequence), with the exceptions noted below. The next character received is assumed to be part of the escape sequence. However, when transmitting to a device, implementors are warned not to count on this property if they wish to follow the emerging ANSI and ISO standards that specify this as an error condition whose recovery is not specified. Control characters, escape sequences, control sequences, and control strings should not (not mandatory) be embedded in other escape sequences, control sequences, or control strings.

3.5.1.2 Termination Conditions

Escape sequences and control sequences normally terminate only on the receipt of a valid final character. However, there are certain conditions that will cause a sequence to be terminated without valid interpretation. In all of these cases, the terminated sequence is ignored.

3.5.1.2.1 Cancel

Cancel (CAN, 1/8) is used to indicate that the data with which it is sent is in error or is to be disregarded. Therefore, the receipt of CAN causes immediate termination, without execution, of any sequence in progress. The CAN character itself receives no further processing. The characters following the CAN are not interpreted as part of the escape or control sequence, but rather are interpreted normally.

3.5.1.2.2 Substitute

Substitute (SUB, 1/10) is used to indicate replacement of a character that could not be represented. The receipt of SUB causes immediate termination, without execution, of any sequence in progress. The SUB character itself, and all subsequent characters, are not interpreted as part of the escape or control sequence, but rather are interpreted normally.

3.5.1.2.3 Escape

If an Escape (ESC, 1/11) is embedded in an escape or control sequence, the sequence is canceled and a new sequence begins with that ESC.

3.5.1.2.4 C1 Control Codes

Since C1 control functions may be represented as ESC Fe sequences in 7-bit environments, C1 control characters embedded in an escape sequence or control sequence shall cause the sequence to be canceled and a new control function to begin with the C1 control character. This provides for compatibility between 7-bit and 8-bit implementations.

3.5.1.2.5 Universal Terminator

In many instances it is important that the communications link be set into a known state in order for conforming interchange to proceed. Because of the complexity of the parsing algorithms and rules regarding control code processing, it is desirable to provide a single escapement that aborts any code processing in progress and restore the terminal to a base state. This may be accomplished using the String Terminator control code, transmitted in its 7-bit format as ESC \ (1/11 5/12). This control terminates the following conditions.

a. Escape Sequence in Progress - The initial Escape control character will cause the sequence in progress to be canceled. The String Terminator will then be processed as an ESC Fe sequence (7-bit C1 control code). If no control string is in progress, the String Terminator will be ignored.

b. Control Sequence in Progress - Same as Escape Sequence in Progress.

c. Control String in Progress - If received before the Final Character, the control string will be treated as a Control Sequence in Progress. If the String Terminator is received after the Final Character, it causes normal termination of the control string.

d. No Control Function in Progress - If a control function is not in progress when the String Terminator is received, it will be processed as an ESC Fe sequence (7-bit C1 control code) and will be ignored by the receiving party.

NOTE

This use of String Terminator applies to ANSI conforming interchange as described in this standard. Specific products may choose to implement non-ANSI modes that do not recognize String Terminator. Such modes should be documented exceptions.

3.5.1.3 Unimplemented Functions

Unimplemented functions shall be ignored as if they were not received. This applies to unimplemented control characters, escape sequences, control sequences, parameters of control sequences, and control strings.

For example, unimplemented selective parameter values of control sequences shall be ignored as if that parameter were not received. Processing of the sequence shall continue with the next parameter. For example, if the control sequence Select Graphic Rendition (SGR) is sent with parameters: 1 (bold), 3 (italicized), 7 (negative image), and 21 (doubly underlined), and the terminal implements bold and negative image, but does not implement italicized or doubly underlined, the parameters 1 and 7 shall be executed, and 3 and 21 shall be ignored.

3.5.1.4 Transformation Between 7-bit and 8-bit Environments

To simplify the transformation between 7-bit and 8-bit environments, relaxations in constraints for the structure of Control Sequences (introduced by CSI), control string introducer sequences (introduced by APC, DCS, PM, and OSC), G2 sequences (introduced by SS2), and G3 sequences (introduced by SS3) are defined as follows:

In an 8-bit code, the bit combinations of columns 10 to 15 (except 15/15) are permitted to represent:

a. Parameters, intermediates, and finals of a control sequence

b. The contents of a control string introducer sequence

c. The operand of a single-shift character

In these situations, the bit combinations in the range 10/0 to 15/15 have the same meaning as the bit combinations in the corresponding row from columns 02 to 07 (bit combinations in the range 2/0 to 7/15 inclusive).

In a 7-bit code, the control characters Shift Out (SO, 0/14) and Shift In (SI, 0/15) are permitted to occur:

a. Between the Control Sequence Introducer (CSI, 9/11) and the final bit combination of a control sequence

b. Between the opening delimiter of a control string and the String Terminator (ST, 9/12)

c. Between a single-shift character and its operand

Shift Out and Shift In have no effect on the interpretation of a control sequence, a control string, or the operand of a single shift character, but may affect the meanings of sebsequent bit combinations in the data stream.

3.5.2 ESCAPE SEQUENCES

An escape sequence is a bit string that is used for control purposes in code extension procedures and that consists of two or more bit combinations, of which the first is the bit combination corresponding to the Escape character (1/11).

The general form of an escape sequence is:

ESC I..I F

where:

a. ESC (Escape) is the introducer character, coded as 1/11 in both 7-bit and 8-bit environments.

b. I..I are zero or more intermediate bit combinations used in combination with the final character to specify the particular function. Intermediate characters for escape sequences are bit combinations in the range of 2/0 to 2/15, inclusive.

   Conforming implementations shall recognize at least three intermediate characters. If more intermediates than can be recognized are received, the entire sequence shall be ignored up to and including the final character.

c. F is a final, function defining character. It is used either by itself or in combination with the intermediate characters (if any) to establish the encoded function. Final characters for escape sequences are bit combinations in the range 3/0 to 7/14 inclusive.

d. The occurrence of bit combinations 0/0 to 1/15, inclusive, as well as 7/15 and 15/15 in escape sequences are special conditions that are handled by immediately executing the control function indicated, and by continuing processing of the escape sequence as if the intervening control code had not been received. Note that if the control code is CAN, ESC, or SUB, the sequence in progress is terminated.

e. The receipt of any 8-bit C1 control character in the range 8/0 to 9/15, inclusive, during the processing of an escape sequence causes the sequence in progress to be terminated. (See subhead 3.6.1.2.4 under Termination Conditions)

f. Characters from columns 10 to 15 (10/0 to 15/14 inclusive) may occur in escape sequences and have the same interpretation as the character from the corresponding row in columns 02 to 07. Thus, the 8th bit of graphic characters should be ignored when interpreting an escape sequence.

3.5.2.1 Expansion Escape Sequence (Fe)

An expansion escape sequence is a special 2-character escape sequence (ESC Fe), where Fe is in columns 04 and 05 (bit combinations from 4/0 to 5/15 inclusive). Expansion escape sequences are used to expand the 7-bit code table by being equivalent to the 8-bit C1 control code in the row in columns 08 and 09 corresponding to the row of the Fe character.

3.5.3 CONTROL SEQUENCES

A control sequence is a bit string consisting of two or more bit combinations, beginning with the Control Sequence Introducer (CSI), and that may contain parametric information for the control function.

The general form of a control sequence is:

CSI P..P I..I F

where:

a. CSI is the introducer character coded as 9/11 in 8-bit environments and the 2-character escape sequence 1/11 5/11 (ESC [ ) in 7-bit environments.

b. P..P is called the "parameter string". The minimum length is zero and the maximum length is defined by the implementation. However, all bit combinations are from 3/0 to 3/15 inclusive.

c. I..I are the intermediate characters that may be used to expand the repertoire of functions beyond the limit of 63 implied by the single final character value.

   Intermediate characters for control sequences are bit combinations in the range 2/0 to 2/15 inclusive. If 3/0 to 3/15 occur after the occurrence of one or more intermediate characters, the sequence is invalid. However, the end of the control sequence is still defined by the final character. Interpretation will continue until a valid final character is received.

   Conforming implementations shall recognize at least three intermediate characters. If more intermediates than can be recognized are received, the entire sequence shall be ignored, up to and including the final character.

d. F is a final, function defining character. It is used either by itself or with the intermediate characters (if any) to establish what function is encoded. Final characters are bit combinations in the range 4/0 to 7/14, inclusive.

e. The occurrence of bit combinations 0/0 to 1/15, inclusive, as well as 7/15 and 15/15 in control sequences, are special conditions that are handled by immediately executing the control function indicated, and continuing processing of the control sequence as if the intervening control code had not been received. Note that if the control code is CAN, ESC, or SUB, the sequence in progress is terminated.

f. The receipt of any 8-bit C1 control character in the range 8/0 to 9/15 inclusive during the processing of a control sequence causes the sequence in progress to be terminated. (See subhead 3.6.1.2.4 under Termination Conditions)

g. Characters from columns 10 to 15 (10/0 to 15/14 inclusive) may occur in control sequences and have the same interpretation as the character from the corresponding row in columns 02 to 07. Thus, the 8th bit of graphic characters should be ignored when interpreting a control sequence.

3.5.3.1 PARAMETER VALUES IN CONTROL SEQUENCES

A parameter string consists of bit combinations from 3/0 to 3/15 inclusive. The string represents one or more numeric or selective parameter values. The maximum value of each parameter is defined by implementation. A minimum of 16384 decimal (14 bits) is recommended. Parameter values larger than the maximum supported by an implementation should be mapped to the largest value supported (although this is not mandatory).

The maximum number of parameters that can be contained in a single parameter string is also implementation defined; however, support for up to 16 parameters is required for compatibility. Parameters beyond the maximum number supported may be ignored.

Each parameter value consists of zero or more bit combinations from 3/0 to 3/9 inclusive, representing the decimal digits 0 to 9. If more than one parameter value is to be supplied, the parameters are separated from each other by the bit combination 3/11 (ASCII semicolon ";"). The bit combination 3/10 is reserved for future standardization. If 3/10 is received within a parameter string, the entire sequence up to and including the final character shall be ignored. Bit combinations 3/12 through 3/15 are used for private parameter strings (see subhead 3.5.3.4).

In each parameter, leading bit combinations of 3/0 are not significant and may be omitted. A zero length parameter, or one consisting only of 3/0 bit combinations, represents a default whose value depends upon the control function.

Representation of a Default Parameter Value

The latest version of ISO 6429 has instituted a change that permits a parameter consisting only of 3/0 bit combinations to be treated as a parameter value of zero, not necessarily the default value. A selectable mode has been defined in the ISO standard that controls the interpretation of a parameter string consisting only of 3/0 bit combinations as the default value.

This standard specifies that implementations continue to recognize a zero-length parameter or one consisting of only 3/0 bit combinations as representing a default value for the control sequence. (A small number of Digital private control functions violate this rule. Refer to DEC STD 138-0 Registry of Control Functions for Character-Imaging Devices for further information.)

3.5.3.2 NUMERIC PARAMETERS

Numeric parameters are used for passing numeric values. If more than one parameter value is supplied, the ordering of parameters is significant and must be preserved by the use of additional separators to replace omitted parameter values. A zero or omitted numeric parameter represents a default that can apply to that one parameter or to the entire parameter string, depending on the control function.

3.5.3.3 SELECTIVE PARAMETERS

Selective parameters are used for selecting options from a list. The list of options and corresponding parameter values is defined by the control function. A control sequence with selective parameters is specified as taking either a fixed or variable number of selective parameters.

If a fixed number of selective parameters is specified, the meaning of each parameter is determined from a separate list of values or actions. All of the parameters are required (or are assumed to be defaulted) for the particular control function to take effect.

If a variable number of selective parameters is specified, each parameter selects a particular entry from a single list identified by the control.

In control sequences with a variable number of parameters, the parameters shall be processed sequentially beginning with the first parameter. A control sequence containing more than one such selective parameter shall have the same effect as a corresponding number of separate control sequences, each with a single parameter.

For example, if Select Graphic Rendition (SGR) is sent with the following parameters, the resulting rendition will be negative image, slowly blinking, since the 0 parameter, executed after setting bold and underline, would return the current rendition value to normal.

1 (bold)

4 (underlined)

0 (all attributes off)

7 (negative image)

5 (slowly blinking)

3.5.3.4 PRIVATE PARAMETER STRINGS

If a parameter string begins with a bit combination from 3/12 through 3/15, inclusive, the entire parameter string is subject to private interpretation. This means that the control has a special interpretation that is not specified in national or international standards. If bit combinations 3/12 through 3/15 occur anywhere else in the control sequence other than as the first character of the parameter string, the entire sequence up to and including the final character shall be ignored.

Syntactically, the subsequent processing of private parameter strings is identical to the processing of standard parameter strings.

3.5.3.5 Examples of Parameter Strings

Table 1. Examples of Parameter Stings

Char Form	Column/row Form	Explanation
7	3/7	A single parameter value of 7
0007	3/0 3/0 3/0 3/7	A single parameter value of 7
98	3/9 3/8	A single parameter value of 98
4;2	3/4 3/11 3/2	Two parameters with values 4 and 2
?3	3/12 3/3	A private parameter with value of 3
2;	3/2 3/11	Two parameters with the first having the value 2 and the second having the default value
;5	3/11 3/5	Two parameters with the first having the default value and the second having the value 5
1;;4	3/1 3/11 3/11 3/4	Three parameters with the first having the value 1, the second having the default value, and the third having the value 4
?3;4	3/15 3/3 3/11 3/4	A private parameter string containing two parameter values, 3 and 4
3;?4	3/3 3/11 3/15 3/4	An error. The occurrence of 3/15 as any but the first character of the parameter string invalidates the control sequence.

3.5.4 CONTROL STRINGS

Control strings are a special class of control code extensions that provide a wide range of functions. There are four types of control strings, each of which has the same general structure with its own introducer character and internal variations. The four types and their formats are:

Application Program Command	APC (9/15)	D..D	ST (9/12)
Device Control Strings	DCS (9/0)	D..D	ST (9/12)
Operating System Command	OSC (9/13)	D..D	ST (9/12)
Privacy Message	PM (9/14)	D..D	ST (9/12)

where:

a. APC, DCS, OSC, or PM is the introducer character that starts the control string and determines its type. In 7-bit environments, the introducer control characters are encoded as 1/11 5/15 (ESC _), 1/11 5/0 (ESC P), 1/11 5/13 (ESC]), and 1/11 5/14 (ESC ^), respectively

b. D..D is a command string which has a unique format for each control string type. However, the command string must be made up of characters in the range 0/8 to 0/13 inclusive and 2/0 to 7/14 inclusive. The inclusion of C0 Controls 0/8 to 0/13 in control string data is intended for convenience in formating and storing control strings. It is recommended that these characters not affect the interpretation of the control string.

c. ST is the String Terminator control character (in 7-bit environments ST is coded as ESC \ (1/11 5/12)).

3.5.4.1 Device Control Strings

The internal format of a Device Control String is:

DCS P..P I..I F D..D ST

where:

a. DCS is the Device Control String introducer.

b. P..P is an optional parameter string, identical in syntax to the parameter string of a control sequence.

c. I..I is zero or more intermediate characters in the range 2/0 to 2/15 inclusive.

d. F is a final, function defining character, in the range 7/0 to 7/14 inclusive (the same as for private Control Sequences).

e. D..D is the command string, whose format is determined by the combination of parameters, intermediate characters, and final character that precede it.

f. ST is the String Terminator control character.

3.5.4.2 Other Control Strings

Other control string types (Application Program Command, Operating System Command, and Privacy Message) are defined to have the following internal format.

```
APC  I..I  F  D..D  ST
OSC  I..I  F  D..D  ST
PM   I..I  F  D..D  ST
```

where:

a. APC, OSC, or PM is the Control String introducer.

b. I..I is zero or more intermediate characters in the range 2/0 to 2/15 inclusive.

c. F is a final, function defining character, in the range 3/0 to 3/15 inclusive (the same as for private Escape Sequences).

d. D..D is the command string, whose format is determined by the combination of intermediate characters and final character which precede it.

e. ST is the String Terminator control character.

3.5.4.3 Character Strings

Future device implementations should treat the SOS C1 control as the start of an unimplemented control string, although this is not mandatory.

A syntax is not yet specified for the interpretation of character strings; the use of SOS and character strings within Digital is reserved for future standardization. Note that ISO may define the inclusion of escape and control sequences inside Character Strings.

3.5.4.4 Termination Conditions

Control strings will normally terminate upon receipt of a String Terminator (9/12) character. The receipt of the control characters 0/8 through 0/13 is valid within control strings. For backward compatibility, and to minimize the effect of a lost String Terminator, the control codes CAN (1/8), SUB (1/10), ESC (1/11), or any C1 control will also terminate control strings.

Conforming software should not depend on this practice however, and shall only use the String Terminator control function to terminate a control string. Control characters other than those recognized within the command string syntax may be defined to have other meanings in the future. It is recommended that all other control codes not used within a command string be ignored by hardware.

3.5.4.5 GR Graphic Characters Within Control Strings

GR (8-bit) graphic characters in APC, OSC, and PM control strings will be treated as their 7-bit equivalent (the eighth bit will be ignored).

GR (8-bit) graphic characters are permitted within Device Control Strings, and the graphic character's interpretation will be dependent on the internal control string format. When they occur in the introducer sequence to a Device Control String, the eighth bit will be ignored, and they will be treated as their 7-bit equivalent. (Note that this is the same way 8-bit graphic characters are handled within control sequences.)

3.5.4.6 Unimplemented Control Strings

The contents of received control strings that are not implemented by the receiving party will be ignored. Therefore, all data (control and graphic characters) from the control string introducer to the string terminator (inclusive) will be discarded without being displayed.

3.5.5 PARSING ALGORITHMS

This section describes the rules required to correctly parse a 7-bit or 8-bit character stream, separating the control functions from data contained in the stream.

Implementation Notes:

This coding is intended to serve as a guideline to implementors and a clear specification of the parsing rules against which product certification can be performed. The code is typical of actual implementations, but is optimized for architectural clarity. It is recognized that specific implementations may alter the algorithms or program structure to optimize memory or execution speed. Such implementations must not violate the external functionality or rules provided by the parsing algorithms.

VAX-11 C was selected as the implementation language because of its simplicity, portability, and wide use both inside and outside of Digital.

Module Overview:

The parser is broken into modules as follows.

gparse.h	global include file (must be included by external routines that call the parser)
lparse.h	local include file (contains definitions used internally by the parser)
vtparse.c	parse routines
ptexec.c	parser tester executive

Other modules not shown here include system-dependent character I/O routines and interrupt handling.

3.5.5.1 C Language Source Code

```
/****************** module gparse.h ********************
*
* Global #include file for callable re-entrant ANSI parser.
*
* Environment:   VAX/VMS
* Author:  Peter Sichel    24-Feb-1984
*
* Modification history:
*     13-Nov-1984   P. Sichel
*         added external events for control
*         string introducers APC, OSC, and PM
*     16-May-1988   P. Sichel
*         added parse limits
*         changed default stack size to 32
*
*********************************************************/

/* parse limits */
#define MAX_NUM_INTERMEDIATES  3
#define MAX_NUM_PARAMETERS     16
#define MAX_PARAMETER_VALUE    16383

/* parse stack */
#define STACK_SIZE 32

struct parse_stack
    {
    char index;      /* stack pointer */
    unsigned char      class[STACK_SIZE];
    unsigned char      value[STACK_SIZE];
    unsigned short int data[STACK_SIZE];
    };

/* common parse states */
#define SEQ_START 0
#define CON_START 1

/* external parse events */
#define R_PARSE_ERROR   0
#define R_CONTINUE      1
#define R_GRAPHIC       2
#define R_CONTROL       3
#define R_ESC_SEQ       4
#define R_CSI_SEQ       5
#define R_DCS_SEQ       6
#define R_APC_SEQ       7
#define R_OSC_SEQ       8
#define R_PM_SEQ        9
```

```
/*   e s c a p e   s e q u e n c e   parser definitions  */
/*   (includes control character parser also)   */

/* parse states */
/*      SEQ_START 0 */    /* already defined above */
/*      CON_START 1 */    /* already defined above */
#define ES_IN_SEQ 2
#define ES_IGNORE 3


/*  c o n t r o l   s e q u e n c e   parser definitions  */

/* parse states */
/*      SEQ_START  0 */     /* already defined above */
#define CS_PRIVATE 1
#define CS_PARAM   2        /* parameter value */
#define CS_INTER   3
#define CS_IGNORE  4        /* ignore sequence,
                               but parse until final */
#define CS_IGNORE_P 5       /* ignore any further parameters,
                               but accept sequence if otherwise
                               valid */
```

```
/************************ module lparse.h ************************
*
*  Local #include file for callable re-entrant ANSI parser
*
*  Environment:  VAX/VMS   VAX-11 C  V1.2
*  Author:  Peter Sichel   24-Feb-1984
*
*  Modification history:
*      2-May-1984    T. Lasko
*          Fixed overlapping #define's in parser states
*      29-May-1984   P. Sichel
*          fixed pseq_parse_table to correcly
*          ignore invalid sequences
*      1-Jun-1984    P. Sichel
*          added state CS_OMIT to indicate
*          omitted parameters explicitly
*      16-May-1988   P. Sichel
*          removed CS_OMIT
*          Added CS_IGNORE_P to ignore parameters
*          beyond MAX_NUM_PARAMETERS.
*          Added ES_IGNORE to ignore escape sequences
*          with more than MAX_NUM_INTERMEDIATES.
*          Fixed bug in parse table that would accept
*          intermediates from CS_IGNORE state.
*       5-Feb-1989   P. Sichel
*          converted parse table data declaration to use
*          symbolic constants, and moved to module vtparse.c
*
************************************************************/
```

```
/* state classes - used by parse executive to determine which
                   parser to invoke */

#define ESEQ        0       /* escape sequence   */
#define CSI         1       /* control sequence  */
#define DCS         2       /* control sequence  */
#define LAST_SEQ    2       /* "in sequence" marker */
#define CON         3       /* control and graphic characters */


/* internal parse events returned by parse_ansi() */

#define PARSE_ERROR   0x00
#define CONTINUE      0x10
#define GRAPHIC       0x20
#define CONTROL       0x30
#define ESC_SEQ       0x40
#define PAR_SEQ       0X50

#define CANCEL        0xD0
#define PARAM         0xE0    /* reduce a numeric parameter */
#define IGNORE        0xF0


/*  e s c a p e  s e q u e n c e  parser definitions  */
/*   (includes control character parser also)  */

    /* parse inputs (count by number of states) */

#define ES_CONTROL     0
#define ES_INTER       4
#define ES_FINAL       8
#define ES_GRAPHIC     12



/*  c o n t r o l  s e q u e n c e  parser definitions  */

    /* parse inputs (count by number of states) */

#define PI_CONTROL     0
#define PI_PRIVATE     6
#define PI_NUMERAL     12
#define PI_SEMICOL     18
#define PI_INTER       24
#define PI_FINAL       30
#define PI_OTHER       36
```

```
/****************** module vtparse.c ***************
*
* Callable re-entrant ANSI parser.  This module contains
* the entire parser, and the action routines that alter
* the parse state.
*
* Environment:  VAX/VMS     VAX-11 C  V1.2
*
* Author:  Peter Sichel     24-Feb-1984
*
* Modified by:
*     PAS 30-May-84  added missing "break" to switch case "IGNORE".
*     PAS 12-Nov-84  modified eseq_class and pseq_class to
*                    recognize 0xFF as a control character
*     PAS 13-Nov-84  added code to recognize OSC, PM, and APC
*                    control strings
*     PAS 16-May-88  Added code to test for MAX_NUM_INTERMEDIATES.
*                    Added code to ignore parameters beyond
*                    MAX_NUM_PARAMETERS.
*                    Added code to handle parameter values
*                    larger than MAX_PARAMETER_VALUE.
*                    Removed distinction between omitted
*                    and zero parameters.
*                    Included basic documentation in this file.
*     PAS  5-Feb-89  Included parse tables in this module.
*/
```

```
/*
Introduction To VTPARSE

VTPARSE is a table-driven, character re-entrant, callable parser
designed to recognize control functions encoded according to the
ANSI syntax for information interchange.  The rules for encoding
control functions are defined in several standards including
ANSI 3.41, DEC STD 138-0 and DEC STD 070-3.

VTPARSE accepts one input character at a time (per call), and
recognizes the following catagories of control functions when
they occur in the input stream.

      o  Graphic Characters

      o  C0 and C1 Control Characters

      o  Escape Sequences

      o  Control Sequences

      o  Control String Introducer Sequences including:
         DCS, OSC, PM, and APC

Interpreting Return Information From parse_ansi

Parse_ansi returns with "event" set to one of the following.

    R_PARSE_ERROR     parse stack overflow, or software bug
    R_CONTINUE        no event, continue parsing
    R_GRAPHIC         recognized a graphic character
    R_CONTROL         recognized a control character
    R_ESC_SEQ         recognized an escape sequence
    R_CSI_SEQ         recognized a control sequence
    R_DCS_SEQ         recognized a DCS introducer sequence
    R_OSC_SEQ         recognized a OSC introducer sequence
    R_PM_SEQ          recognized a PM  introducer sequence
    R_APC_SEQ         recognized a APC introducer sequence

The character or character sequence recognized is on the
"state" stack.  Parse_ansi returns two integers pointing to
the first and final characters of the sequence on the stack.

Each stack entry has three fields: a "class" field, a "value"
field, and a "data" field.  The possible values for these fields
are defined in the include files "gparse.h" and "lparse.h".

The "class" field is used internally by the parser to determine
which subparser to invoke (that is, the parse table to use). To
keep the tables reasonably small, a separate table is used for
parsing parameter sequences.
```

The "value" field is used by the parser to maintain its current state. The parser is a table driven finite state automata, where the next state and action are a function of the current state and input. The "value" field of the parse stack provides a history of what states the parser has been in and is useful for interpreting the contents of the parse stack. The "value" field may be used to identify the contents of the "data" field except for the final character of a sequence. The final "value" is usually meaningless because when a sequence is recognized, it is removed from the stack eliminating the need for a parse state. A single graphic or control character is considered both the start and final character of a sequence.

The "data" field contains the actual character data input to the parser or a number of characters in reduced form. Examples of reduced forms are a single C1 control replacing an Fe sequence or a complete parameter value replacing a number of individual parameter characters. Reduced forms will always be identified by the contents of the "value" field.

Examples of typical stack data are:

```
                value          data
              ----------     --------
final ->   CON_START         41 'A'     event = R_GRAPHIC
                                        (the letter "A")

final ->   CON_START         0D         event = R_CONTROL
                                        (carriage return)

final ->   CON_START         41 'A'     event = R_ESC_SEQ
           ES_IN_SEQ         28 '('     (designate UK into G0)
start ->   SEQ_START         1B ESC

final ->   CS_IGNORE         68 'h'     event = R_CSI_SEQ
           CS_PARAM          3          parameter value 3
           CS_PRIVATE        3F '?'     DEC private char
start ->   SEQ_START         9B CSI     (set 132 col mode)

final ->   CS_IGNORE         72 'r'     event = R_CSI_SEQ
           CS_PARAM          14         parameter value 14(hex)
           CS_PARAM          0          zero or omitted parameter
start ->   SEQ_START         9B CSI     (set scroll region from
                                         line 1 to line 18)

final ->   CS_IGNORE         72 'p'     event = R_DCS_SEQ
start ->   SEQ_START         90 DCS     (begin REGIS DCS)

final ->   CON_START         9C ST      event = R_CONTROL
                                        (string terminator)
```

Errors And Exceptions

The parser ignores any invalid sequences it receives. No control function is recognized, and the parser resynchronizes the input stream according to the rules for error recovery in DEC STD 138-0.

To summarize:

- ESC restarts any escape, control, or control string introducer sequence in progress.

- CAN and SUB immediately cancel any escape, control, or control string introducer sequence in progress. SUB is recognized as a C0 control if no sequence is in progress.

- All other C0 controls are immediately recognized within escape, control, or control string introducer sequences. The escape, control, or control string introducer sequence continues where it left off.

- C1 controls immediately cancel any escape, control, or control string introducer sequence in progress. The C1 control is then recognized normally. This is for 7-bit compatibility where C1 controls are represented by two character Fe escape sequences.

  Two character Fe escape sequences are converted to C1 controls by the parser.

- Within escape, control, and control string introducer sequences, eight-bit graphic codes (GR set) are truncated to seven-bit codes.

- If a private parameter character (<, =, >, ?) occurs after the first parameter, the entire sequence will be ignored up to and including its final character.

- If a parameter character occurs after the first intermediate character, the entire sequence will be ignored up to and including its final character.

- The parser recognizes parameter values up to 16383 (14 bits) as specified by the constant MAX_PARAMETER_VALUE in "gparse.h". If a parameter exceeds the maximum value, the parameter will be set to the maximum value.

- The parser recognizes up to 16 parameters in a single parameter sequence as specified by the constant MAX_NUM_PARAMETERS in "gparse.h". If more than MAX_NUM_PARAMETERS are received, parameters beyond the maximum number are ignored and the "value" field for the last parameter will be set to "CS_IGNORE_P" (this may be used to detect that parametershave been ignored). The sequence will still be recognized if it is otherwise valid.

- The parser will recognize up to three intermediate characters in a single escape, control, or introducer sequence specified by the constant MAX_NUM_INTERMEDIATES in "gparse.h". If more than MAX_NUM_INTERMEDIATES are received, the entire sequence will be ignored up to and including its final character.

- DCS introducer sequences are parsed according to the rules for control sequences. APC, OSC, and PM introducer sequences are parsed according to the rules for escape sequences. The parser recognizes the introducer sequence upon receiving a valid final character. The parser does not keep track of whether a control string is in progress. Therefore it is possible to receive control codes, escape sequences, and control sequences within a control string. Handling of control strings must be performed at a higher level than the parser.

Each intermediate character requires one stack entry. Each parameter value also requires one stack entry. The default stack size of 32 is sufficient to hold a sequence starting character, a private parameter character, 16 parameters, 3 intermediates, and still recognize nested C0 control characters. If the maximum number of parameters or intermediates are increased, the stack size should be increased accordingly. If the stack overflows, the parser will re-initialize the stack, and return with event PARSE_ERROR.
*/

```
#include "gparse.h"    /* global parse definitions */
#include "lparse.h"    /* local parse definitions */


/* escape sequence parse table */
/*  eseq_parse_table[input+from_state]  ==>  action,to_state  */

char eseq_parse_table[16] =
    {

    /* input: ES_CONTROL */
    /* from_state     action    to_state  */
    /* ---------------------------------*/
    /* SEQ_START */   CONTROL | ES_IN_SEQ,
    /* CON_START */   CONTROL | CON_START,
    /* ES_IN_SEQ */   CONTROL | ES_IN_SEQ,
    /* ES_IGNORE */   CONTROL | ES_IGNORE,

    /* input: ES_INTER   */
```

```
    /* from_state      action     to_state  */
    /* ------------------------------------*/
    /* SEQ_START */    CONTINUE | ES_IN_SEQ,
    /* CON_START */    GRAPHIC  | CON_START,
    /* ES_IN_SEQ */    CONTINUE | ES_IN_SEQ,
    /* ES_IGNORE */    IGNORE   | ES_IGNORE,

    /* input: ES_FINAL    */
    /* from_state      action     to_state  */
    /* ------------------------------------*/
    /* SEQ_START */    ESC_SEQ  | CON_START,
    /* CON_START */    GRAPHIC  | CON_START,
    /* ES_IN_SEQ */    ESC_SEQ  | CON_START,
    /* ES_IGNORE */    CANCEL   | ES_IGNORE,

    /* input: ES_GRPAHIC */
    /* from_state      action     to_state  */
    /* ------------------------------------*/
    /* SEQ_START */    PARSE_ERROR | ES_IN_SEQ,
    /* CON_START */    GRAPHIC     | CON_START,
    /* ES_IN_SEQ */    PARSE_ERROR | ES_IN_SEQ,
    /* ES_IGNORE */    PARSE_ERROR | ES_IGNORE

    };


/* control sequense parse table */
/*  pseq_parse_table[input+from_state]  ==>  action,to_state  */

char pseq_parse_table[42] =
    {
    /* input: PI_CONTROL */
    /* from_state         action     to_state  */
    /* --------------------------------------*/
    /* SEQ_START    */    CONTROL  | CS_IGNORE,
    /* CS_PRIVATE   */    CONTROL  | CS_IGNORE,
    /* CS_PARAM     */    CONTROL  | CS_IGNORE,
    /* CS_INTER     */    CONTROL  | CS_IGNORE,
    /* CS_IGNORE    */    CONTROL  | CS_IGNORE,
    /* CS_IGNORE_P  */    CONTROL  | CS_IGNORE_P,

    /* input: PI_PRIVATE */
    /* from_state         action     to_state  */
    /* --------------------------------------*/
    /* SEQ_START    */    CONTINUE | CS_PRIVATE,
    /* CS_PRIVATE   */    CONTINUE | CS_IGNORE,
    /* CS_PARAM     */    CONTINUE | CS_IGNORE,
    /* CS_INTER     */    CONTINUE | CS_IGNORE,
    /* CS_IGNORE    */    IGNORE   | CS_IGNORE,
    /* CS_IGNORE_P  */    IGNORE   | CS_IGNORE,

    /* input: PI_NUMERAL */
```

```
    /* from_state          action      to_state  */
    /* ------------------------------------------*/
    /* SEQ_START    */     PARAM     | CS_PARAM,
    /* CS_PRIVATE   */     PARAM     | CS_PARAM,
    /* CS_PARAM     */     PARAM     | CS_PARAM,
    /* CS_INTER     */     CONTINUE  | CS_IGNORE,
    /* CS_IGNORE    */     IGNORE    | CS_IGNORE,
    /* CS_IGNORE_P  */     IGNORE    | CS_IGNORE_P,

    /* input: PI_SEMICOL */
    /* from_state          action      to_state  */
    /* ------------------------------------------*/
    /* SEQ_START    */     PARAM     | CS_PARAM,
    /* CS_PRIVATE   */     PARAM     | CS_PARAM,
    /* CS_PARAM     */     PARAM     | CS_PARAM,
    /* CS_INTER     */     CONTINUE  | CS_IGNORE,
    /* CS_IGNORE    */     IGNORE    | CS_IGNORE,
    /* CS_IGNORE_P  */     IGNORE    | CS_IGNORE_P,



    /* input: PI_INTER */
    /* from_state          action      to_state  */
    /* ------------------------------------------*/
    /* SEQ_START    */     CONTINUE  | CS_INTER,
    /* CS_PRIVATE   */     CONTINUE  | CS_INTER,
    /* CS_PARAM     */     CONTINUE  | CS_INTER,
    /* CS_INTER     */     CONTINUE  | CS_INTER,
    /* CS_IGNORE    */     IGNORE    | CS_IGNORE,
    /* CS_IGNORE_P  */     CONTINUE  | CS_INTER,

    /* input: PI_FINAL */
    /* from_state          action      to_state  */
    /* ------------------------------------------*/
    /* SEQ_START    */     PAR_SEQ   | CS_IGNORE,
    /* CS_PRIVATE   */     PAR_SEQ   | CS_IGNORE,
    /* CS_PARAM     */     PAR_SEQ   | CS_IGNORE,
    /* CS_INTER     */     PAR_SEQ   | CS_IGNORE,
    /* CS_IGNORE    */     CANCEL    | CS_IGNORE,
    /* CS_IGNORE_P  */     PAR_SEQ   | CS_IGNORE,

    /* input: PI_OTHER */
    /* from_state          action      to_state  */
    /* ------------------------------------------*/
    /* SEQ_START    */     CONTINUE  | CS_IGNORE,
    /* CS_PRIVATE   */     CONTINUE  | CS_IGNORE,
    /* CS_PARAM     */     CONTINUE  | CS_IGNORE,
    /* CS_INTER     */     CONTINUE  | CS_IGNORE,
    /* CS_IGNORE    */     IGNORE    | CS_IGNORE,
    /* CS_IGNORE_P  */     CONTINUE  | CS_IGNORE

    };
```

```
/* shorthand for "Stack Pointer" of parse stack */

#define SP state->index

/************************************
*
* parse_init - initialize parse state
*
************************************/

parse_init(state)

struct parse_stack *state;
{
    state->index = 0;
    state->class[0] = CON;
    state->value[0] = CON_START;
    return(0);
}



/*****************************************************
*
* parse_ansi - callable re-entrant ansi parser executive
*
*****************************************************/

parse_ansi(c, state, event, start, final)

struct parse_stack *state;  /* address of parse stack structure */

int c;       /* input character */
int *event;  /* event recognized (by ref) */
int *start;  /* stack index of first char of event (by ref) */
int *final;  /* stack index of final char of event (by ref) */

{
int action, i;

/*.........................................*/

/* if in sequence, convert 8 bit graphics to 7 bits */
if (state->class[SP] <= LAST_SEQ)
    if (c >= 0xA0) c &= 0x7F;

/* invoke appropriate parser based on state class */
switch (state->class[SP])
    {
    case CON:
        /* parse graphic and CONtrol characters */
        action = parse(state, c, eseq_class(c), eseq_parse_table);
```

```
        break;

    case ESEQ:
        /* parse escape sequences */
        action = parse(state, c, eseq_class(c), eseq_parse_table);

        /* check for too many intermediates */
        i = 0;
        while (state->value[SP-i] == ES_IN_SEQ)
            {
            i++;
            if (i > MAX_NUM_INTERMEDIATES)
                {
                /* ignore sequence (but parse until final) */
                state->value[SP] = ES_IGNORE;
                }
            }
        break;

    case CSI:
    case DCS:
        /* parse parameter sequence */
        action = parse(state, c, pseq_class(c), pseq_parse_table);




        /* check for too many intermediates */
        i = 0;
        while (state->value[SP-i] == CS_INTER)
            {
            i++;
            if (i > MAX_NUM_INTERMEDIATES)
                {
                /* ignore sequence (but parse until final) */
                state->value[SP] = CS_IGNORE;
                }
            }
        break;
    }

/* process recognized events */
switch (action)
    {
    case CONTINUE:
        *event = R_CONTINUE;
        break;
    case GRAPHIC:
        *event = R_GRAPHIC;
        *start = SP;
        *final = SP;
        SP -= 1;
        break;
    case CONTROL:
        pcontrol(state, event, start, final);
```

```
        break;
    case ESC_SEQ:
        pescseq(state, event, start, final);
        break;
    case PAR_SEQ:
        *final = SP;
        /* find start of sequence */
        *start = SP;
        while (state->value[*start] != SEQ_START) *start -= 1;
        switch (state->data[*start])
            {
            case 0x9B: *event = R_CSI_SEQ; break;
            case 0x90: *event = R_DCS_SEQ; break;
            default:   *event = R_PARSE_ERROR; break;
            }
        /* remove seq from parse stack,
           and return to previous parser */
        SP = *start - 1;
        break;
    case PARAM:
        /* input was either a numeral, or a semicolon,
           reduce parameter chars to parameter value
           if current char is a numeral then
               if prev is a value, mul by 10, add current,
                  range check.
               else store current as a value.
           else if current is a semicolon



               store value of zero
               if prev is not a value,
               advance stack and store a value of zero.
           finally, check for too many parameters.  */
        *event = R_CONTINUE;
        i = SP;    /* remember stack pointer */
        if (state->data[SP] != 0x3B)
            {    /* numeral */
            if (state->value[SP-1] == CS_PARAM)
                {
                SP -= 1;
                if (state->data[SP] > MAX_PARAMETER_VALUE/10)
                    state->data[SP] = MAX_PARAMETER_VALUE;
                else
                    {
                    state->data[SP] *= 10;
                    state->data[SP] += state->data[SP+1] - 0x30;
                    if (state->data[SP] > MAX_PARAMETER_VALUE)
                        state->data[SP] = MAX_PARAMETER_VALUE;
                    }
                }
            else
                state->data[SP] -= 0x30;
            }
        else
```

```
            {    /* semicolon */
            state->data[SP] = 0;
            if (state->value[SP-1] != CS_PARAM)
                {
                SP += 1;
                state->class[SP] = state->class[SP-1];
                state->value[SP] = CS_PARAM;
                state->data[SP] = 0;
                }
            }
        /* check for too many parameters */
        if (SP >= i)    /* did we add a parameter? */
            {
            i = 0;
            while (state->value[SP-i] == CS_PARAM)
                {
                i++;
                if (i > MAX_NUM_PARAMETERS)
                    {
                    /* ignore last parameter */
                    SP -= 1;
                    /* ignore any further parameters */
                    state->value[SP] = CS_IGNORE_P;
                    }
                }
            }
        break;
    case CANCEL:
        if (state->class[SP] <= LAST_SEQ)
            {



            while (state->value[SP] != SEQ_START) SP -= 1;
            SP -= 1;
            }
        *event = R_CONTINUE;
        break;
    case IGNORE:
        *event = R_CONTINUE;
        SP -= 1;
        break;
    default:
        *event = R_PARSE_ERROR;
        parse_init(state);    /* re-initialize */
        break;

    }   /* end switch event */

return(*event);
}   /* end routine parse_ansi */
```

```
/*************************************************************
*
* eseq_class - return escape sequence parser input class
*              of given character.
*
*************************************************************/

eseq_class(c)
int c;
{
int c_class;
/*....................*/

/* determine character class */
if (0x20 <= c && c <= 0x2F)
    c_class = ES_INTER;
else if ( (0x00 <= c && c <= 0x1F) ||
          (0x7F <= c && c <= 0x9F) ||
          (c == 0xFF) )
    c_class = ES_CONTROL;
else if (0x30 <= c && c <= 0x7E)
    c_class = ES_FINAL;
else
    c_class = ES_GRAPHIC;

return(c_class);
}



/*************************************************************
*
* pseq_class - return parameter sequence parser input class of
* given character.
*
*************************************************************/
pseq_class(c)
int c;
{
int pi_class;
/*....................*/

/* determine character class */
if (0x30 <= c && c <= 0x39)
    pi_class = PI_NUMERAL;
else if (c == 0x3B)
    pi_class = PI_SEMICOL;
else if (0x20 <= c && c <= 0x2F)
    pi_class = PI_INTER;
else if (0x3C <= c && c <= 0x3F)
    pi_class = PI_PRIVATE;
```

```
else if (0x40 <= c && c <= 0x7E)
    pi_class = PI_FINAL;
else if ( (0x00 <= c && c <= 0x1F) ||
          (0x7F <= c && c <= 0x9F) ||
          (c == 0xFF) )
    pi_class = PI_CONTROL;
else
    pi_class = PI_OTHER;

return(pi_class);
}



/*******************************************
*
* parse - table driven parse routine
*
* returns the following events:
*   PARSE_ERROR
*   CONTINUE
*   GRAPHIC
*   CONTROL
*   ESC_SEQ
*   CON_SEQ
*   PARAM
*   CANCEL
*   IGNORE
*
* The parser is a table driven finite state automata,
* where the next state and action are a function of
* the current state and input.
*
*******************************************/
parse(state, c, c_class, parse_table)

struct parse_stack *state;      /*  parse stack         */
int c;                          /*  input character     */
int c_class;                    /*  input class         */
char parse_table[];             /*  parse table         */

{

int next_action;
int current_state;
int next_state;

/*.................................*/

/* get next state */
current_state = state->value[SP];
next_state = parse_table[c_class + current_state] & 0x0F;

/* get action */
next_action = parse_table[c_class + current_state] & 0xF0;
```

```
/* default action */
if (SP >= STACK_SIZE) return(PARSE_ERROR);
SP += 1;
state->class[SP] = state->class[SP-1];
state->value[SP] = next_state;
state->data[SP] = c;

/* return action */
return(next_action);

}   /* end routine parse */


/*****************************************
*
* pcontrol - process controls that alter
*            the parse state.
*
*****************************************/
pcontrol(state, event, start, final)

struct parse_stack *state;
int *event;
int *start;
int *final;

{
int sp, code;
/*................*/

sp = state->index;
code = state->data[sp];

/* ESC */
if (code == 0x1B)
    {
    /* if already in escape or control sequence, restart it */
    if (state->class[sp] <= LAST_SEQ)
        while (state->value[sp] != SEQ_START) sp -= 1;
    state->class[sp] = ESEQ;    /* switch to escape seq parser */
    state->value[sp] = SEQ_START;
    state->data[sp] = 0x1B;
    state->index = sp;
    *event = R_CONTINUE;
    return(0);
    }

/* CAN */
if (code == 0x18)
    {
    /* if already in escape or control sequence, cancel it */
    if (state->class[sp] <= LAST_SEQ)
        {
        while (state->value[sp] != SEQ_START) sp -= 1;
```

```
        }
    state->index = sp - 1;
    *event = R_CONTINUE;
    return(0);
    }

/* SUB */
if (code == 0x1A)
    {
    /* if already in escape or control sequence, cancel it */
    if (state->class[sp] <= LAST_SEQ)
        {
        while (state->value[sp] != SEQ_START) sp -= 1;
        state->index = sp-1;



        *event = R_CONTINUE;
        }
    else
        {    /* not in sequence, recognize SUB */
        *event = R_CONTROL;
        *start = sp;
        *final = sp;
        state->index = sp;
        }
    return(0);
    }


/* is it a C1 control ? */
if (code > 0x7F)
    {

    /* if already in escape or control sequence, cancel it */
    if (state->class[sp] <= LAST_SEQ)
        while (state->value[sp] != SEQ_START) sp -= 1;

    /* CSI */
    if (code == 0x9B)
        {
        /* switch to parameter seq parser */
        state->class[sp] = CSI;
        state->value[sp] = SEQ_START;
        state->data[sp] = code;
        state->index = sp;
        *event = R_CONTINUE;
        return(0);
        }

    /* DCS */
    if (code == 0x90)
        {
        /* switch to parameter seq parser */
        state->class[sp] = DCS;
```

```
        state->value[sp] = SEQ_START;
        state->data[sp] = code;
        state->index = sp;
        *event = R_CONTINUE;
        return(0);
        }


/* OSC, PM, APC */
    if ( (code == 0x9D) || (code == 0x9E) || (code == 0x9F) )
        {
        /* switch to escape sequence parser */
        state->class[sp] = ESEQ;
        state->value[sp] = SEQ_START;
        state->data[sp] = code;
        state->index = sp;
        *event = R_CONTINUE;
        return(0);
        }

    }   /* end C1 control */


/* recognize other controls */
*event = R_CONTROL;
state->data[sp] = code;
*start = sp;
*final = sp;
/* remove control character from parse stack */
state->index = sp - 1;
return(0);
}
```

```
/*************************************
*
* pescseq - process escape sequences
*           that alter the parse state.
*
*************************************/
pescseq(state, event, start, final)

struct parse_stack *state;
int *event;
int *start;
int *final;

{
int sp;
/*.....................*/

sp = state->index;

/* two character escape sequence ? */
if (state->data[sp-1] == 0x1B)
    {

    /* convert 7-bit Fe sequences to C1 controls */
    if ((0x40 <= state->data[sp]) &&  (state->data[sp] <= 0x5F))
        {
        /* store C1 control */
        state->data[sp-1] = state->data[sp] + 0x40;
        state->index = sp-1;
        /* steal code from pcontrol */
        pcontrol(state, event, start, final);
        return(0);
        }
    }

/* recognize other escape sequences */
/* find start of sequence */
while (state->value[sp] != SEQ_START) sp -= 1;
*start = sp;
*final = state->index;
switch (state->data[*start])
    {
    case 0x1B:  *event = R_ESC_SEQ; break;
    case 0x9D:  *event = R_OSC_SEQ; break;
    case 0x9E:  *event = R_PM_SEQ;  break;
    case 0x9F:  *event = R_APC_SEQ; break;
    default:    *event = R_PARSE_ERROR; break;
    }
state->index = *start - 1;   /* remove seq from parse stack */
return(0);

}   /* end routine pescseq */
```

```
/********************* module ptexec.c **************
*
* This module contains the parser test executive.
* It reads characters from the keyboard and dumps the
* parse stack when a control function is recognized.
*
* Environment:  VAX/VMS
*
* Author:  Peter Sichel    24-Feb-1984
*
* Modification history
*    8-May-1984    T. Lasko
*        Made ^Y (power off) cleaner.
*    9-MAY-1984    T. Lasko
*        Fixed several protocols.
*    13-Nov-1984   P. Sichel
*        Added OSC, PM, and APC control strings
*    9-May-1988    P. Sichel
*        Cosmetic changes for inclusion in VSRM
*/
```

```
#include <stdio.h>
#include "gparse.h"   /* global parse definitions */
#include "lparse.h"   /* local parse definitions */

/* global storage */
extern int poweron;

/* module wide storage */
struct parse_stack kbd_state;

static char *eseq_state[3] =
    {
    "SEQ_START",
    "CON_START",
    "ES_IN_SEQ"
    };
static char *pseq_state[6] =
    {
    "SEQ_START",
    "CS_PRIVATE",
    "CS_PARAM",
    "CS_INTER",
    "CS_IGNORE",
    "CS_IGNORE_P"
    };
```

```
/*******************************************
*  ptstart() -
*
*  This is the startup routine for the parser tester.
*  It simply initializes the internal state and the terminal
*  screen, then calls the executive loop, which returns when
*  we give it a ^Y (power turns off).
*******************************************/
ptstart()
{
unsigned short n;

/* erase screen, move cursor to home */
printf("\33[2J\33[H");
printf("A N S I   P a r s e r   T e s t e r\n");
printf("    Press ^Y to exit \n\n");

parse_init(&kbd_state);
poweron = 1;              /* ready....*/
set_passall();

execloop();               /* go! */

set_nopassall();
return(0);                /* when power off...return */
}
```

```
/*****************************************
*
* execloop - parser test executive
*
*****************************************/
execloop()
{

int c;
int event, start, final, i;

/*.........................................*/

while (poweron)
    {

    /* get character from KEYBOARD and parse it */
    c = readkbd();

    parse_ansi(c, &kbd_state, &event, &start, &final);

    /* process recognized events */
    switch (event)
        {
        case R_CONTINUE:
            break;
        case R_GRAPHIC:
            printf("\nR_GRAPHIC:");
            ptdump(&kbd_state, start, final);
            break;
        case R_CONTROL:
            printf("\nR_CONTROL:");
            ptdump(&kbd_state, start, final);
            break;
        case R_ESC_SEQ:
            printf("\nR_ESC_SEQ:");
            ptdump(&kbd_state, start, final);
            break;
        case R_CSI_SEQ:
            printf("\nR_CSI_SEQ:");
            ptdump(&kbd_state, start, final);
            break;
        case R_DCS_SEQ:
            printf("\nR_DCS_SEQ:");
            ptdump(&kbd_state, start, final);
            break;
        case R_OSC_SEQ:
            printf("\nR_OSC_SEQ:");
            ptdump(&kbd_state, start, final);
            break;
        case R_PM_SEQ:
            printf("\nR_PM_SEQ:");
            ptdump(&kbd_state, start, final);
            break;
```

```
        case R_APC_SEQ:
            printf("\nR_APC_SEQ:");
            ptdump(&kbd_state, start, final);
            break;
        default:
            printf("\nR_PARSE_ERROR:");
            ptdump(&kbd_state, start, final);
            break;
        }   /* end switch case */

    }   /* while (poweron) */

return(0);
}   /* end routine execloop */
```

```
/**********************************
*
* ptdump - parser test stack dump
*
**********************************/
ptdump( state, start, final )

struct parse_stack *state;
int start, final;

{
int i;

for (i=start; i<=final; i++)
    {
    printf("\n    ");
    switch (state->class[i])
        {
        case ESEQ:
            printf("ESEQ %-11s", eseq_state[state->value[i]]);
            break;
        case CSI:
            printf("CSI  %-11s", pseq_state[state->value[i]]);
            break;
        case DCS:
            printf("DCS  %-11s", pseq_state[state->value[i]]);
            break;
        case CON:
            printf("CON  %-11s", eseq_state[state->value[i]]);
            break;
        default:
            printf("???      %02x    ", state->value[i]);
        }

    if (state->data[i] < 16) printf("  %02x  ", state->data[i]);
        else printf("%4x  ", state->data[i]);
    if ((0x1F < state->data[i]) && (state->data[i] < 0x7F))
        putchar((int)state->data[i]);
    }

return(0);
};
```

3.6 GRAPHIC CODE EXTENSION TECHNIQUES

In a 7-bit environment, the basic graphics' portion of the code table consists of a single 94- or 96-character set. In an 8-bit environment, this is supplemented with an additional 94- or 96-character set, and the two sets are referred to as the GL set (left half graphics, corresponding to the single set in 7-bits), and the GR set (right half). In both environments, the graphic codes in the table can be further extended by the designation and invocation of a potentially unlimited repertory of graphics sets. This is implemented by the provision of four sets of 94 or 96 graphic characters, referred to as the G0, G1, G2, and G3 sets.

Character sets may be designated to any of the four G-sets, which may be currently or subsequently invoked into either the GL or GR positions in the code table.

3.6.1 DESCRIPTION OF SHIFT FUNCTIONS

Shift functions are used to invoke one of the G-sets into the left or right half side of the code table. There are two kinds of shift functions: locking shifts and single shifts. Locking shifts cause a G-set to be permanently invoked into the code table, where it will remain until another shift function occurs to replace it. Single shifts cause a G-set to be invoked only for the occurrence of the single character code following the single shift control in the code stream, after which the table reverts to its previous state before the shift occurred.

For example, if on power-up the ASCII character set is designated into G0 and G1, and G0 is invoked into GL, then any graphic codes in the GL portion of the table (2/1 through 7/14) will cause an ASCII character to be generated. Subsequently, the Line Drawing character set may be designated into G1 to replace ASCII in that G-set. No visible effect will occur until G1 is invoked into GL using a shift function. Thereafter, any GL graphic codes received will cause Line Drawing characters to be displayed.

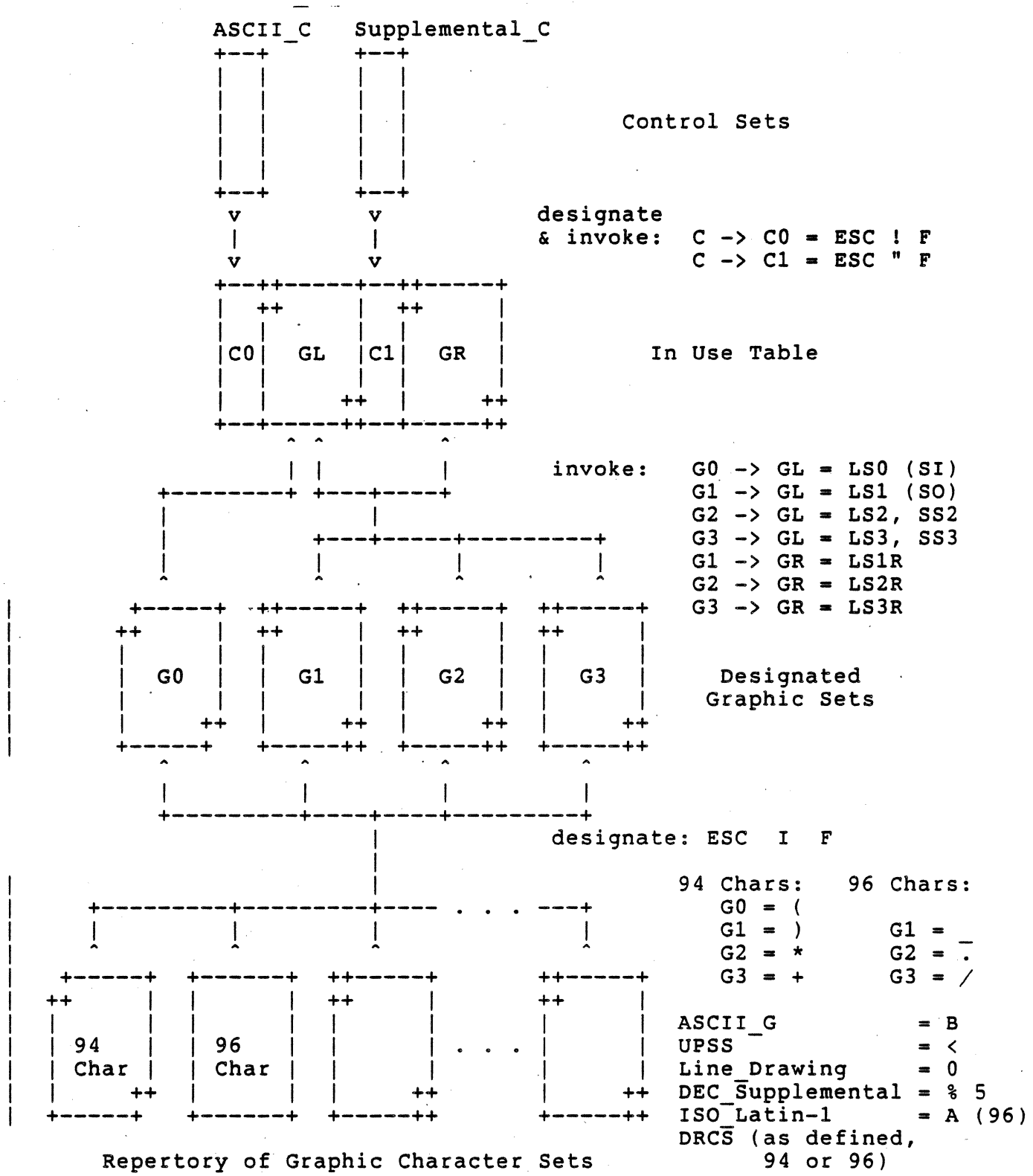


Figure 5. Designating and Invoking Control and Graphic Character Sets into the Eight-bit In Use Table

3.6.1.1 Locking Shifts

Conforming terminal products will provide the following locking shift functions in Level 1 operation.

Code	Name	G-set invoked	Table position
LS0 *	Locking Shift Zero	G0	GL
LS1 *	Locking Shift One	G1	GL
LS2	Locking Shift Two	G2	GL
LS3	Locking Shift Three	G3	GL

* Note - The functions Locking Shift Zero and Locking Shift One have been previously referred to as Shift In (SI) and Shift Out (SO). The effect of executing these controls has not been changed.

Conforming terminals will provide the following additional locking shift functions in Level 2 operation.

Code	Name	G-set invoked	Table position
LS1R	Locking Shift One Right	G1	GR
LS2R	Locking Shift Two Right	G2	GR
LS3R	Locking Shift Three Right	G3	GR

The following bit combinations are not affected by the use of the locking shifts.

a. Those corresponding to control characters in columns 00, 01, 08, and 09

b. Those corresponding to the character Space (SP) in position 2/0, and Delete (DEL) in position 7/15 when a 94-character set is invoked into GL

c. Those corresponding to positions 10/0 and 15/15 when a 94-character set is invoked into GR

d. Those included in any escape or control sequence (see notes to subheadings 3.5.2 Escape Sequences and 3.5.3 Control Sequences for a full explanation of the use of bit combinations within escape and control sequences)

e. The bit combination following a single shift function (SS2 or SS3)

3.6.1.2 Single Shift Functions

Conforming terminal products will provide the following single shift functions in both 7-bit and 8-bit environments.

Code	Name	G-set invoked	Table position
SS2	Single Shift Two	G2	GL
SS3	Single Shift Three	G3	GL

3.6.2 DESIGNATING CHARACTER SETS

Each terminal shall maintain a repertory of available character sets, which may be designated to any of the four G-sets (note there is no direct way to designate a 96 character set into G0). Once a character set is designated to a G-set, it remains in that set until explicitly replaced by another character set designator. If the G-set is invoked into the code table at the time the character set is designated, it immediately replaces the existing set in the code table.

Note: For a complete description of conforming software use of the designating and invoking controls, see DEC STD 070-1. DEC STD 70 addresses conformance of software when communicating directly with a presentation device, or when writing data to a file that is intended only for subsequent transmission to a presentation device. It does not address conformance of software when processing terminal input or when writing data to a file that is intended for subsequent processing by application software.

DEC STD 169-0 DEC Standard Coded Graphic Character Sets for Hardware and Software requires that when data is intended for subsequent processing conforming software shall not use either locking shift or single shift functions. This requirement is in fact a subset of the conformance requirements defined in this document, therefore the two documents do conflict.

3.6.3 THE USER PREFERENCE SUPPLEMENTAL SET (UPSS)

The User Preference Supplemental Set is a logical character set name similar in concept to a G-set (G0-G3) but at the next higher level. The UPSS allows one level of software indirection for designating supplemental character sets. The UPSS is designated and invoked like any other character set, but the character set used can be selected by the user as a SETUP option or by host control function in response to a user request.

The UPSS was introduced to facilitate migration from DEC-MCS to ISO Latin-1. Software applications that do not need to be concerned with which supplemental character set is used can defer

the choice of a specific supplemental set to the user, thus not imposing a choice that might interfere with the customers migration plan. The designating sequence previously used for DEC Supplemental was changed to designate the UPSS, providing automatic transparent migration from DEC MCS to ISO Latin-1 for existing applications as soon as terminal devices can support ISO Latin-1.

The concept of a User Preference Supplemental Set also accomodates existing CSS Specials that used the old DEC Supplemental designating sequence to designate a local country specific character set.

Support for the UPSS is part of the 8-bit Interface Architecture Extension to Level 2, and required for conformance to Level 3. As a minimum, the UPSS must be selectable between DEC Supplemental and ISO Latin-1 supplemental. The UPSS also establishes the initial supplemental set for display (defaults to G2 on power-up), and determines the keyboard supplemental character set when 8-bit multinational characters are enabled. See DEC STD 070-6.

```
typedef enum    /* upss_type */
    {
    UPSS_DEC_SUPPLEMENTAL_G, UPSS_ISO_LATIN1_SUPPLEMENTAL_G
    } upss_type;
upss_type upss;
```

3.6.4 DEFAULT DESIGNATION AND INVOCATION

The following assignments will apply to the power-on and soft reset states of the terminal.

3.6.4.1 Level 1

In Level 1 operation, the default character set designations will be ASCII in G0, G1, G2, and G3. Furthermore, the G0 set (ASCII) will be invoked into the GL position of the In Use Table. Level 1 operation is 7-bits only.

3.6.4.2 Level 2 (without 8-bit Interface Architecture Extension)

In Level 2 operation, the default character set designations will be ASCII in G0 and G1, and the DEC Supplemental graphics set in G2 and G3. Furthermore, the G0 set (ASCII) will be invoked into the GL position of the In Use Table, and the G2 set (DEC Supplemental graphics) will be invoked into the GR position of the In Use Table.

3.6.4.3 Level 3 (or Level 2 with 8-bit Interface Architecture Extension)

In Level 3 operation, the default character set designations will be ASCII in G0 and G1, and the User Preference Supplemental Set (UPSS) in G2 and G3. Furthermore, the G0 set (ASCII) will be invoked into the GL position of the In Use Table, and the G2 set (UPSS, typically DEC Supplemental or ISO Latin-1 supplemental) will be invoked into the GR position of the In Use Table.

Level 3 (or Level 2 with the 8-bit Interface Architecture Extension) also recognizes the "Announce Subset Of Code Extension Facilities" defined in ISO 4873 which re-initialize the designated and invoked character sets as described later in this chapter.

3.6.4.4 NRCS Extension

When the NRCS Extension is present, ASCII may be replaced with a National Replacement Character set as a Set-Up option. In this case, the default character set designations will be the NRCS in G0, G1, G2, and G3, with G0 in GL. National Replacement Character Sets are intended for backward compatibility and may be used as above in Level 1, Level 2, and Level 3 by selecting "7-bit NRCS Characters" in Set-Up or setting DECNRCM (National Replacement Character set Mode) from the host. Setting this mode has the side effect of resetting the display character sets to their default state as described above. Note that the NRC Mode is 7-bits only, GR is not available. The actual NRC set used is determined by the "Keyboard Dialect", and the selection of "Typewriter" or "Data Processing" keys. See DEC STD 070-6 for more information.

3.7 STATE DESCRIPTIONS

3.7.1 CONTROL SETS

At the present time the C0 and C1 control sets are fixed as the ANSI control sets defined in ANSI X3.4 - 1986 and ISO 6429: 1988.

```
typedef enum    /* c0_control_set_type */
    {
    ASCII_C
    } c0_control_set_type;

typedef enum    /* c1_control_set_type */
    {
    SUPPLEMENTAL_C
    } c1_control_set_type;
```

3.7.2 G-SETS

Graphic character sets are defined for each service class. (See DEC STD 070-5 Video Systems Reference Manual - Character Cell Display for a complete description of the available character sets and designating sequences.) The actual character sets available will vary for different conformance levels and extensions (NRCS, 8-bit IA, TCS)). Four character sets may be designated at any time, each into one of the four G-sets, which may be subsequently invoked into the In Use Table.

```
typedef enum    /* graphic_character_set_type */
    {
    ASCII_G,
    LINE_DRAWING_G,
    DEC_SUPPLEMENTAL_G,
    ISO_LATIN1_SUPPLEMENTAL_G,    /* 8-bit IA */
    UPSS_G                        /* 8-bit IA */
    } graphic_character_set_type;

graphic_character_set_type designated_graphic_sets[4];
```

3.7.3 IN USE TABLE

Mapping of received control functions and graphic character data is provided by the 8-bit code table, as described above. Since the 8-bit table is a superset of the 7-bit table, all references will be to the 8-bit table in both environments.

```
typedef enum    /* gset_name_type */
    {
    G0, G1, G2, G3
    } gset_name_type;

typedef struct    /* in_use_table_type */
    {
    c0_control_set_type          c0;
```

```
    graphic_character_set_type   gl;
    gset_name_type               invoked_gl;

    c1_control_set_type          c1;
    graphic_character_set_type   gr;
    gset_name_type               invoked_gr;
    } in_use_table_type;

in_use_table_type in_use_table;
```

3.7.4 SINGLE SHIFT FUNCTIONS

Since single shift controls affect the processing of the next graphic character in the received data stream, it is necessary to retain their occurrence until subsequent processing occurs.

```
typedef enum    /* single_shift_type */
    {
    NONE, SS2, SS3
    } single_shift_type;

single_shift_type single_shift;
```

3.7.5 ENVIRONMENTS

The environment of the host port can be selected for either a seven or eight bit data path in set-up. The environment of the printer port can be selected independently of the host environment as a set-up function. See DEC STD 070-7 Video Systems Reference Manual - Printer Port Extension.

Note that the logical environment for which each port is set is independent of the actual physical hardware setting of the ports, and may in fact be different. For example, it may be possible to set the host port for an 8-bit environment, even though the physical channel is limited by the hardware to 7-bits. The architecture assumes that the physical and logical settings are compatible, and does not define the results when this is not the case. The terminal characteristics are dependent on both the environment and the conformance level of operation. However, there are only three possible states for the terminal to be in that this architecture specifies:

* Level 1 operation

Host Port	transmit -	7-bit controls	7-bit graphics
	receive -	7-bit controls	7-bit graphics
(Printer Port	transmit -	7-bit controls	7-bit graphics)

* Level 2 or Level 3 operation, 7-bit C1 controls

Host Port	transmit -	7-bit controls	8-bit graphics
	receive -	8-bit controls	8-bit graphics

* Level 2 or Level 3 operation, 8-bit C1 controls

Host Port	transmit -	8-bit controls	8-bit graphics
	receive -	8-bit controls	8-bit graphics

3.7.5.1 Level 1 - Host Port

Level 1 operation provides only for 7-bit communications. Thus, in Level 1 all control and graphic characters can only be received and transmitted in seven bits. The high order bit of all 8-bit characters received will be stripped by the terminal before any processing occurs. All C1 controls will be transmitted by the terminal as ESC Fe sequences. Only 7-bit characters will be generated from the keyboard (that is, attempts to generate 8-bit characters will fail).

3.7.5.2 Level 2 or Level 3 - Host Port

In Level 2 or Level 3 operation both 7-bit and 8-bit communications are supported. The terminal will accept C1 control characters and GR graphic characters in both 7-bit and 8-bit C1 Transmission Modes.

When 7-bit communications is selected (7-bit host port environment), C1 controls will only be transmitted as ESC Fe sequences, and only 7-bit characters will be generated from the keyboard (that is, attempts to generate 8-bit characters on line will fail).

When 8-bit communications is selected (8-bit host port environment), the terminal can be selected to transmit C1 control characters either as 7-bit ESC Fe sequences, or as 8-bit control characters. The terminal will generate GR characters from the keyboard in eight bits.

```
typedef enum    /* environment_type */
    {
    SEVEN_BIT, EIGHT_BIT
    } environment_type;

environment_type
    host_port_environment,
    printer_port_environment,
    c1_transmission_mode;
```

3.8 CONTROL OPERATIONS

3.8.1 ANNOUNCE SUBSET OF CODE EXTENSION FACILITIES

ANNOUNCE SUBSET OF CODE EXTENSION FACILITIES

Levels: 2X, 3 (8-bit Interface Architecture Extension)

Purpose: To indicate which subset of code extension facilities defined in ISO 4873 conforming interchange will use.

Format:

ESC	SP	<final>
1/11	2/0	

<final>		Level of conformance
L	4/12	Level 1 of ISO 4873
M	4/13	Level 2 of ISO 4873
N	4/14	Level 3 of ISO 4873

Description: This sequence announces the level of conformance that the subsequent interchange will conform to. Conformance to ISO 4873 requires some default designation and invocations on the part of the terminal.

Upon receipt of a Level 1, 2, or 3 announcer sequence, the terminal will make the following designations and invocations:

Level 1
- ASCII is designated into G0
- ISO Latin Alphabet Nr 1 supplemental is designated into G1
- G0 is invoked into GL
- G1 is invoked into GR

Level 2
- ASCII is designated into G0
- ISO Latin Alphabet Nr 1 supplemental is designated into G1
- G0 is invoked into GL
- G1 is invoked into GR

Level 3
- ASCII is designated into G0
- G0 is invoked into GL

NOTES:

Conformance to a subset of ISO 4873 code extension and interchange is NOT equivalent to the Digital conformance levels for Character Cell Display terminals. These announcer sequences and DECSCL both specify conformance levels, but to a different standard (ISO 4873 in this case).

Any hard or soft terminal reset (receipt of RIS,

DECSTR, or DECSCL, or issuing the RECALL function from Set-Up), will cause the Terminal to reset the "Default Designations And Invocations" as described previously to conform with DEC STD 169-0.

Interchange conforming to ISO 4873 which issues a RIS, DECSTR, or DECSCL, must subsequently send an announcer sequence to regain the default ISO designations and invocations.

State Affected:

```
graphic_character_set_type  designated_graphic_sets[4];
graphic_character_set_type  gl;
graphic_character_set_type  gr;
gset_name_type              invoked_gl;
gset_name_type              invoked_gr;
```

Algorithm:

```
void announce_level_1 ()
{
if ( (conformance_level > 2) ||
     (level_2_extensions[8bit_ia]) )
    {
    designated_graphic_sets[0] = ASCII_G;
    designated_graphic_sets[1] = ISO_LATIN1_SUPPLEMENTAL_G;
    gl = designated_graphic_sets[0];
    invoked_gl = G0;
    gr = designated_graphic_sets[1];
    invoked_gr = G1;
    }
}


void announce_level_2 ()
{
if ( (conformance_level > 2) ||
     (level_2_extensions[8bit_ia]) )
    {
    designated_graphic_sets[0] = ASCII_G;
    designated_graphic_sets[1] = ISO_LATIN1_SUPPLEMENTAL_G;
    gl = designated_graphic_sets[0];
    invoked_gl = G0;
    gr = designated_graphic_sets[1];
    invoked_gr = G1;
    }
}

void announce_level_3 ()
{
if ( (conformance_level > 2) ||
     (level_2_extensions[8bit_ia]) )
```

```
    {
    designated_graphic_sets[0] = ASCII_G;
    gl = designated_graphic_sets[0];
    invoked_gl = G0;
    }
}
```

Known Deviations: None

3.8.2 COMMUNICATIONS ENVIRONMENT

SELECT 7-BIT C1 TRANSMISSION **S7C1T**

Levels: 2, 3

Purpose: Cause the device to generate C1 controls in 7-bits as two character ESC Fe sequences.

Format:	ESC	SP	F
	1/11	2/0	4/6

Description: The S7C1T control causes the terminal to use the 7-bit encoding for all C1 control characters transmitted. Therefore, all C1 characters will be represented as two character ESC Fe sequences.

Notes:

1. The coding of this control is defined in ISO and ANSI standards, but is not assigned a name in those standards.

State Affected:

```
environment_type c1_transmission_mode;
```

Algorithm:

```
void select_7_bit_c1_transmission ()
{
if (conformance_level != LEVEL_1)
    c1_transmission_mode = SEVEN_BIT;
}
```

Known Deviations: None

SELECT 8-BIT C1 TRANSMISSION S8C1T

Levels: 2, 3

Purpose: Cause the device to generate C1 controls in 8-bits as single character control codes.

Format:	ESC	SP	G
	1/11	2/0	4/7

Description: The S8C1T control causes the terminal to use the 8-bit encoding for all C1 control characters transmitted. Therefore, all C1 characters will be represented as single character control codes.

Notes:

1. The coding of this control is defined in ISO and ANSI standards, but is not assigned a name in those standards.
2. If the Host_Port_environment is set to Seven_Bit, this control will be ignored by the terminal.

State Affected:

```
environment_type c1_transmission_mode;
```

Algorithm:

```
void select_8_bit_c1_transmission ()
{
if ( (conformance_level != LEVEL_1) &&
     (host_port_environment == EIGHT_BIT) )
    c1_transmission_mode = EIGHT_BIT;
}
```

Known Deviations: None

3.8.3 SHIFT FUNCTIONS

3.8.3.1 Locking Shifts

SHIFT IN	SI
LOCKING SHIFT ZERO	LS0

Levels: 1, 2, 3

Purpose: Invoke the G0 graphic character set into the GL position of the code table.

Format: LS0 (SI)

1/15

Description: The LS0 or SI control invokes the set of graphic characters currently designated as a G0 set into the left-hand graphics portion of the Code Table. Receipt of this control causes subsequent data in the range 2/1 through 7/14 that is entered into the display to be rendered according to the current contents of the G0 character set.

NOTE

G0 cannot normally be a 96-character set since there is no control function to designate a 96-character set as G0. A 96 character UPSS could be designated as G0 in which case subsequent data in the range 2/0 to 7/15 would be affected. Conforming software shall not designate a 96 character UPSS as G0.

State Affected:

```
in_use_table_type in_use_table;
```

Algorithm:

```
void shift_in ()
{
in_use_table.gl = designated_graphic_sets[0];
in_use_table.invoked_gl = G0;
}
```

Known Deviations: None

SHIFT OUT SO
LOCKING SHIFT ONE LS1

Levels: 1, 2, 3

Purpose: Invoke the G1 graphic character set into the GL position of the code table.

Format: LS1 (SO)

1/14

Description: The LS1 or SO control invokes the set of graphic characters currently designated as a G1 set into the left-hand graphics portion of the Code Table. Receipt of this control causes subsequent data in the range 2/1 through 7/14 (or 2/0 through 7/15 for 96 character sets) that is entered into the display to be rendered according to the current contents of the G1 character set.

State Affected:

```
in_use_table_type in_use_table;
```

Algorithm:

```
void shift_out ()
{
in_use_table.gl = designated_graphic_sets[1];
in_use_table.invoked_gl = G1;
}
```

Known Deviations: None

LOCKING SHIFT TWO LS2

Levels: 2, 3

Purpose: Invoke the G2 graphic character set into the GL position of the code table.

Format:

	ESC	n
	1/11	6/14

Description: The LS2 control invokes the set of graphic characters currently designated as a G2 set into the left-hand graphics portion of the Code Table. Receipt of this control causes subsequent data in the range 2/1 through 7/14 (or 2/0 through 7/15 for 96 character sets) that is entered into the display to be rendered according to the current contents of the G2 character set.

Notes:

1. This control is included in the terminal interface for interchange compatibility with ISO and ANSI standards. It is not to be used by conforming software.

State Affected:

```
in_use_table_type in_use_table;
```

Algorithm:

```
void locking_shift_two ()
{
if (conformance_level == LEVEL_2)
    {
    in_use_table.gl = designated_graphic_sets[2];
    in_use_table.invoked_gl = G2;
    }
}
```

Known Deviations: None

LOCKING SHIFT THREE LS3

Levels: 2, 3

Purpose: Invoke the G3 graphic character set into the GL position of the code table.

Format:

ESC	o
1/11	6/15

Description: The LS3 control invokes the set of graphic characters currently designated as a G3 set into the left-hand graphics portion of the Code Table. Receipt of this control causes subsequent data in the range 2/1 through 7/14 (or 2/0 through 7/15 for 96 character sets) that is entered into the display to be rendered according to the current contents of the G3 character set.

Notes:

1. This control is included in the terminal interface for interchange compatibility with ISO and ANSI standards. It is not to be used by conforming software. See DEC STD 070-1.

State Affected:

```
in_use_table_type in_use_table;
```

Algorithm:

```
void locking_shift_three ()
{
if (conformance_level == LEVEL_2)
    {
    in_use_table.gl = designated_graphic_sets[3];
    in_use_table.invoked_gl = G3;
    }
}
```

Known Deviations: None

LOCKING SHIFT ONE RIGHT LS1R

Levels: 2

Purpose: Invoke the G1 graphic character set into the GR position of the code table.

Format: ESC ~

1/11 7/14

Description: The LS1R control invokes the set of graphic characters currently designated as a G1 set into the right-hand graphics portion of the Code Table. Receipt of this control causes subsequent data in the range 10/1 through 15/14 (or 10/0 through 15/15 for 96 character sets) that is entered into the display to be rendered according to the current contents of the G1 character set.

Notes:

1. This control is included in the terminal interface for interchange compatibility with ISO and ANSI standards. It is not to be used by conforming software. (See DEC STD 070-1.)

2. Conformance to Level 3 of ISO 4873 requires the use of LS1R to invoke the right-hand part of the eight-bit code table.

State Affected:

```
in_use_table_type in_use_table;
```

Algorithm:

```
void locking_shift_one_right ()
{
if (conformance_level == LEVEL_2)
    {
    in_use_table.gr = designated_graphic_sets[1];
    in_use_table.invoked_gr = G1;
    }
}
```

Known Deviations: None

LOCKING SHIFT TWO RIGHT LS2R

Levels: 2, 3

Purpose: Invoke the G2 graphic character set into the GR position of the code table.

Format:

ESC	}
1/11	7/13

Description: The LS2R control invokes the set of graphic characters currently designated as a G2 set into the right-hand graphics portion of the Code Table. Receipt of this control causes subsequent data in the range 10/1 through 15/14 (or 10/0 through 15/15 for 96 character sets), that is entered into the display to be rendered according to the current contents of the G2 character set.

State Affected:

```
in_use_table_type in_use_table;
```

Algorithm:

```
void locking_shift_two_right ()
{
if (conformance_level == LEVEL_2)
    {
    in_use_table.gr = designated_graphic_sets[2];
    in_use_table.invoked_gr = G2;
    }
}
```

Known Deviations: None

LOCKING SHIFT THREE RIGHT LS3R

Levels: 2, 3

Purpose: Invoke the G3 graphic character set into the GR position of the code table.

Format:

ESC	
1/11	7/12

Description: The LS3R control invokes the set of graphic characters currently designated as a G3 set into the right-hand graphics portion of the Code Table. Receipt of this control causes subsequent data in the range 10/1 through 15/14 (or 10/0 through 15/15 for 96 character sets) that is entered into the display to be rendered according to the current contents of the G3 character set.

State Affected:

```
in_use_table_type in_use_table;
```

Algorithm:

```
void locking_shift_three_right ()
{
if (conformance_level == LEVEL_2)
    {
    in_use_table.gr = designated_graphic_sets[3];
    in_use_table.invoked_gr = G3;
    }
}
```

Known Deviations: None

3.8.3.2 Single Shifts

SINGLE SHIFT TWO SS2

Levels: 1, 2, 3

Purpose: Temporarily invoke a single character from the designated G2 set.

Format: SS2

8/14

Description: The SS2 control causes the next character in the received data stream, which must be in the range 2/0 through 7/15 or 10/0 through 15/15, to be rendered according to the current contents of the G2 set. If the next character in the received data stream is not a graphic character in this range, the single shift function is ignored and the received character is processed as if the single shift had not been received.

The only exception to this rule is that a single Shift Out (0/14) or Shift In (0/15) control character may occur between the Single Shift and the subsequent graphic character. The Shift Out or Shift In character will be interpreted normally, but does not cancel the Single Shift. The following bit combination shall be interpreted as a character from the G2 set. This is done for conformance to ISO 2022, and is intended to facilitate the design of 8-bit to 7-bit transformation facilities.

State Affected:

```
single_shift_type single_shift;
boolean_type cancel_single_shift;
```

Algorithm:

(See "Single Shift Three" below.)

SINGLE SHIFT THREE SS3

Levels: 1, 2, 3

Purpose: Temporarily invoke a single character from the designated G3 set.

Format: SS3

8/15

Description: The SS3 control causes the next character in the received data stream, which must be in the range 2/0 through 7/15 or 10/0 through 15/15, to be rendered according to the current contents of the G3 set. If the next character in the received data stream is not a graphic character in this range, the single shift function is ignored and the received character is processed as if the single shift had not been received.

The only exception to this rule is that a single Shift Out (0/14) or Shift In (0/15) control character may occur between the Single Shift and the subsequent graphic character. The Shift Out or Shift In character will be interpreted normally, but does not cancel the Single Shift. The following bit combination shall be interpreted as a character from the G3 set. This is done for conformance to ISO 2022, and is intended to facilitate the design of 8-bit to 7-bit transformation facilities.

State Affected:

```
single_shift_type single_shift;
boolean_type cancel_single_shift;
```

Algorithm:

```
/* . . . main loop */
c = readcom();

parse_ansi(c, &com_state, &event, &start, &final);

if (single_shift != NONE)
    {
    if (cancel_single_shift) single_shift = NONE;
    cancel_single_shift = IS_TRUE;
    }

/* process recognized events . . . */
if (event == R_CONTROL)
    {
    code = com_state->data[final];

    if (code == 0x0E)    /* SO (LS1) */
        {
        shift_out();
```

```
            cancel_single_shift = IS_FALSE;
            }
        else if (code == 0x0F)    /* SI (LS0) */
            {
            shift_in();
            cancel_single_shift = IS_FALSE;
            }

        else if (code == 0x8E)
            {
            single_shift = SS2;
            cancel_single_shift = IS_FALSE;
            }
        else if (code == 0x8F)
            {
            single_shift = SS3;
            cancel_single_shift = IS_FALSE;
            }
        /* . . . */
        }
    /* continue main loop . . . */
```

Known Deviations:

Previous versions of this standard incorrectly stated that an SO or SI between a Single Shift and its operand would be ignored. Since we do not normally translate between 7-bit and 8-bit environments within Digital, this has not been a problem. Future implementations should conform to ISO 2022 as described above.

3.9 CHANGE HISTORY

3.9.1 REVISION 0.2 to 0.3

1. The standard's title was changed from "Communications and Coding Interchange" to "Code Extension Layer".

2. The names of the Reference Standards were corrected.

3. The conformance section was removed, and will later be integrated into the section "Concepts and Conformance Criteria".

4. The terminology section was moved from the back of the document to the front.

5. A sentence was added to the section on graphic character codes to clarify the handling of bit combinations in the range 2/1 through 7/14 and 10/1 through 15/14 when received within a control function.

6. Qualifications were added to the section on rate limiting in terminals.

7. XON was removed from the definition of the Universal Terminator.

8. Sections were added to the notes on Escape Sequences and Control Sequences to clarify the processing of 7/15 and 15/15 and C1 control characters within control functions.

9. The section on processing of control sequence parameter values was substantially rewritten.

10. The section on Control Strings was moved ahead of the parser design.

11. A note was added on the non-conformance of the VT125 in processing CAN within control strings.

12. Global changes were made to the parser design to incorporate the processing of DCS introducer sequences.

13. A figure was added showing the logical relationship of the C sets, G sets, In Use Table, and character sets.

14. A note was added referencing DEC STD 169-0 and clarifying the distinction between conforming Presentation interchange and conforming Application interchange.

15. The 8-bit locking shifts table was corrected for LS3R.

16. All state descriptions were rewritten, and the names changed for consistency with other parts of the SRM.

17. The environments section was substantially rewritten.

18. The default character set designations were changed to distinguish between Level 1 operation and Level 2 operation.

19. The names of Select 7-bit C1 and Select 8-bit C1 were changed to include "Transmission". These controls are now called S7C1T and S8C1T.

20. GR Transmission Mode and Send/Receive Mode were removed from the architecture.

21. The coding was changed for LS0, LS1, LS1R, LS2R, LS3R, SS2 and SS3. Notes were added on conforming software use of these controls.

22. The character set designators were removed from this section and will later be incorporated into the section "Character Cell Display".

3.9.2 REVISION 0.3 to AX10

1. Removed reference to use of Delete as a pad character, and made wording stronger that conforming software must use NUL only.

2. Added note to description of the processing of the 10/0 character to indicate that it is treated as Space (2/0) within control functions.

3. Made minor corrections and changed a few variable names in the parser algorithms to be consistent with other sections of the SRM.

4. Added a note to the section on environments indicating that the "logical" environment might in fact be different from the "physical" environment of the hardware, and if this is the case the architecture does not define the result.

5. Changed descriptions of Single_Shift_Two and Single_Shift_Three as well as the algorithms to indicate that they will be ignored if the next character in the received data stream is not a graphic character. (With the exception that a single Shift Out or Shift In character may occur.)

6. Removed the notes which restricted conforming software from using Single Shift Two and Single Shift Three.

7. Added a note that rate limiting should be the factory default in devices which have a capability of defeating this feature.

8. Changed the description of control characters occurring within control strings to permit, but deprecate the use of, any function other than String Terminator to terminate a string.

9. Added a note that GR (8-bit) graphic characters may occur within control strings.

3.9.3 REVISION AX10 to AX11

1. Removed Rev AX10 change bars. Added change bars to any change that could affect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

2. Changed GET__NEXT__CODE__FROM__INPUT__BUFFER to strip parity bit in Level 1.

3. Added statement that the inclusion of C0 Controls 0/8 to 0/13 in control string data is intended for convenience in formating and storing control strings. It is recommended that these characters not affect the interpretation of the control string.

3.9.4 REVISION AX11 to AX12

1. Removed Rev AX11 change bars. Added change bars to any change that could affect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

2. Added description of 8-bit Interface Architecture Extension including Support for 96-character graphic character sets, ISO Latin-1 Supplemental, and the User Preference Supplemental Set (UPSS). The 8-bit Interface Architecture Extension is required for Level 3 conformance, and strongly recommended for Level 2. Modified rest of chapter to assume the 8-bit Interface Architecture Extension is present. Graphic characters can now be in the range 2/0-7/15 and 10/0-15/15 when 96-character graphic sets are used. Expanded description of default designations and invocations for 8-bit Interface Architecture.

3. Updated referenced standards section. Added dpANS X3.134.1-1985, X3.134.2-1985, and ISO 8859-1:1987.

4. Removed one page section on communication controls (XON/XOFF, padding, and rate limiting) because they are not really part of Code Extension. These will included in chapter 12 on Terminal Synchronization.

5. Changed recommendation for minimum size of parameters that should be supported. Was 255 (8-bits), changed to 16384 (14-bits).

6. Added recommendation for handling parameters beyond the device maximum. Parameters beyond the maximum number supported should be ignored.

7. Extended description of multiple selective parameters to conform with current usage. Not all sequences containing more than one selective parameter have the same effect as a corresponding number of separate control sequences, each with a single parameter.

8. Clarified range of final characters for DCS sequences 7/0 to 7/14 (same as for private control sequences).

9. Added description of Digital extended syntax for other Control Strings (APC, OSC, PM). Syntax is patterned after Private Escape Sequences.

10. Added description of Character Strings (SOS) and anticipatory guidelines compatible with ISO 6429.

11. Clarified rules for 8-bit graphic characters within control strings including exception for DCS strings. 8-bit graphic characters are treated as their 7-bit

equivalents (8th bit ignored) in APC, OSC, PM strings, and DCS introducer sequences. Interpretation of 8-bit characters in data portion of DCS strings is dependent on the internal control string format.

12. Added description of NRCS Extension allowing ASCII to be replaced with NRC set (including default designations and invocations). Removed previous description of British NRCS as a special case. Included reference to section 6 where DECNRCM is documented.

13. Removed description of Printer Port Environment (7-bits/8-bits) within Host Port Environment. Added reference to "Printer Port Extension" chapter.

14. Added description of "Announce Subset of Code Extension Facilities" per ISO 4873 including algorithm and escape sequences.

15. Added note about the possibility of a 96-character UPSS being designated as G0. Normally there is no way to directly designate a 96-character set as G0. Conforming software shall not designate a 96-character UPSS as G0.

16. Changed algorithms to use character re-entrant parser coded in VAX-11 C. This table driven ANSI parser is easier to follow, and is being used in actual products.

17. Changed standard to show that SI or SO occurring between a Single Shift and its operand does not cancel the Single Shift (SS2 or SS3) but may still affect subsequent data.

3.10 REFERENCED DOCUMENTS

DIGITAL STANDARDS

EL-00070-01	DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria
EL-00070-05	DEC STD 070-5 Video Systems Reference Manual - Character Cell Display
EL-00070-06	DEC STD 070-6 Video Systems Reference Manual - Keyboard Processing
EL-00070-07	DEC STD 070-7 Video Systems Reference Manual - Printer Port Extension
EL-00138-00	DEC STD 138-0 Registry of Control Functions for Character Imaging Devices
EL-00169-00	DEC STD 169-0 DEC Standard Coded Graphic Character Sets for Hardware and Software

Copies of Digital Standards Can be obtained from Standards and Methods Control, $ VTX SMC, JOKUR::SMC, DTN 287-3724, or CTS1-2/D4.

Please provide your name, badge number, cost center, mailstop, and ENET node when ordering.

ANSI AND ISO STANDARDS

ANSI X3.4 - 1986	American National Standard Code for Information Interchange (ASCII character set)
ANSI X3.41 - 1974	American National Standard Code Extension Techniques for use with the 7-Bit Coded Character Set of the American National Standard Code for Information Interchange
ANSI X3.64 - 1979	Additional Controls for use with American National Standard Code for Information Interchange
ANSI X4.23 - 1982	Keyboard Arrangement for Alphanumeric Machines

dpANS X3.134.1-1985	8-bit ASCII Structure and Rules
dpANS X3.134.2-1985	7-bit and 8-bit ASCII Supplemental Multinational Graphic Character Set
ISO 646	Information Processing / 7-Bit Coded Character Set for Information Interchange
ISO 2022:1986	Information Processing / ISO 7-Bit and 8-Bit Coded Character Sets - Code Extension Techniques
ISO 6429:1988	Information Processing / Control Functions for Coded Character Sets
ISO 8859-1:1987	Information Processing / 8-Bit Single-Byte Coded Character Sets - Part 1 : ISO Latin Alphabet Nr 1;

Copies of ANSI and ISO Standards can be obtained from local Digital Libraries.

This page deliberately left blank.

Section Index

READER COMMENTS

Your comments and suggestions will help Standards and Methods Control improve their services and documents.

Did you request this document? _____ If so, did it arrive within a satisfactory period of time? _____ Please comment.

What are your impressions of this document? Consider format, organization, completeness, readability, and illustrations.

- -- -- -- -- -- -- -- -- -- -- FOLD ON THIS LINE -- -- -- -- -- -- -- --

Did you find technical or clerical errors in this document? If so, please specify the page number(s) and the error(s).

Are the instructions for the update package clear? _____
Do you have other suggestions for improving this document?

The following information is optional:

Name _____ Mailstop _____

Department _____ Node _____

Send your comments to JOKUR::PROJECTS, or fold, staple, and send this page through interoffice mail to:

READERS' COMMENTS
STANDARDS AND METHODS CONTROL
CTS1-2/D4

DEC STD 070-4 Video Systems Reference Manual - Terminal Management

DOCUMENT IDENTIFIER: A-DS-EL00070-04-0000 Rev A, 04-Oct-1990

ABSTRACT: This section describes device and service-class independent protocols for performing identification and status functions, and for selecting syntax for interchange compatibility.

APPLICABILITY: Mandatory for engineers designing hardware for terminal products and software engineers designing programs using terminal interfaces. Additional requirements are defined in DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria.

STATUS: APPROVED 04-Oct-1990; use VTX SMC for current status.

The material contained within this document is assumed to define mandatory standards unless it is clearly marked as: a. not mandatory; or b. guidelines.

Material that is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques that are considered to be good practice, but should not be considered as requirements.

DEC STD 070-4 Video Systems Reference Manual - Terminal Management

DOCUMENT IDENTIFIER: A-DS-EL00070-04-0000 Rev A, 04-Oct-1990

REVISION HISTORY: Rev A 04-Oct-1990

Document Management Category:	Terminal Interface Architecture (STI)
Responsible Department:	VIPS Terminals Architecture
Responsible Person:	Peter Sichel
SMC Writer:	Don Mehaffey

APPROVAL: This document has been reviewed and recommended for approval by the General Review group for its category.

Peter A. Sichel

Peter Sichel - VIPS Terminals Architecture

Eric A Williams 22Feb91

Eric Williams - Standards Process Manager

Direct requests for further information to:

Peter Sichel
Use $ VTX ELF for the latest location information.

Use VTX SMC to order copies of this document from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

CONTENTS

FIGURES

TABLES

1 INTRODUCTION

This standard defines the interface to perform terminal management functions in all terminal devices. Terminal management functions include device control operations that are independent of any display service class and that are not associated with the presentation of data.

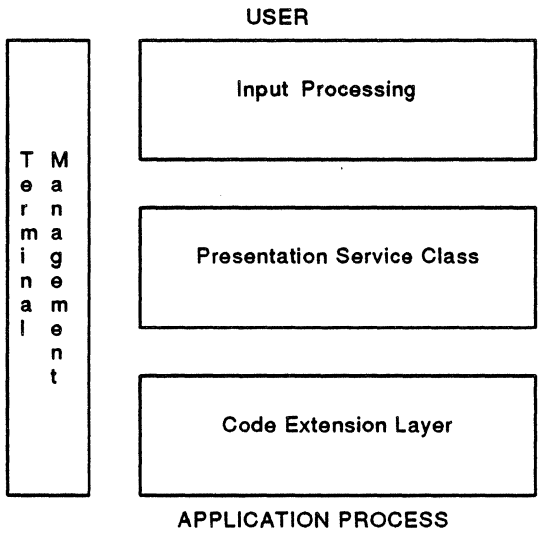
1.1 SCOPE

This standard applies to all Digital products that implement terminal management interfaces and are to be certified as conforming to the Terminal Interface Architecture (TIA). It also applies to software that is intended to use these interfaces in a conforming manner.

1.2 RELATIONSHIP TO TERMINAL INTERFACE ARCHITECTURE (TIA)

The terminal management functions are a unique class of device controls that operate across layers of the TIA.

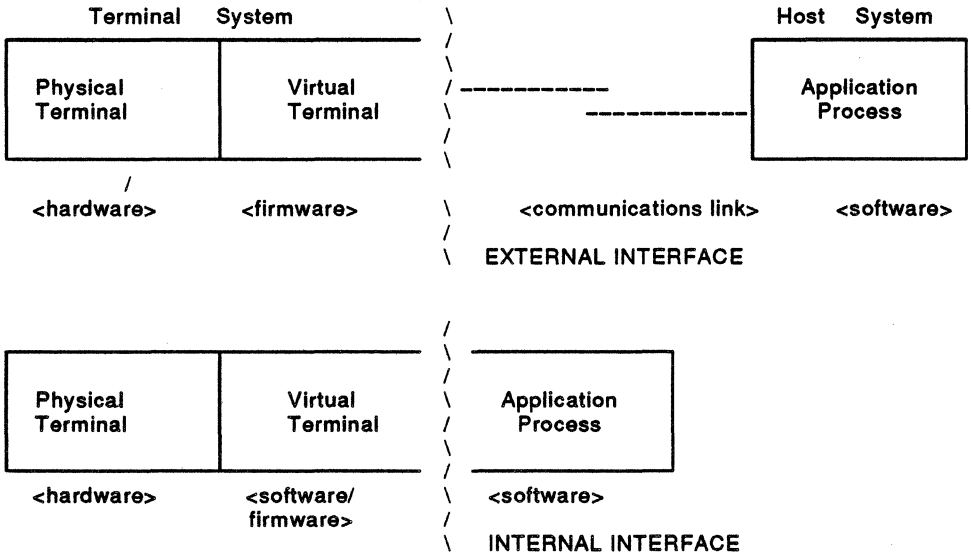
Figure 1: Structuring of the Terminal Interface Architecture



To aid in understanding, functions described in this document are identified as intended for either application, terminal management, or communications use when an external communications interface is present (guideline). These different uses may not always be separated. For example, applications may require their own terminal management. If not otherwise indicated, functions described in this document are intended for terminal management use.

The interfaces defined within this document apply to both internal and external product interfaces. External interfaces are interfaces between a terminal, personal computer, or workstation and a remote system. Internal interfaces are interfaces between a terminal sub-system and software processes running within a terminal, personal computer, or workstation.

Figure 2: External and Internal Terminal Interfaces



2 TERMINOLOGY

Conformance Level - An architectural agreement that defines a number of functions within a class of operations that must be adhered to in all products claiming to implement that level. Conformance levels are defined to provide interface compatibility with other devices at the same level. Selecting a particular conformance level automatically places the terminal into a known state (the Soft Terminal Reset state on video terminals), without affecting the data currently contained in the display.

Device Self-Test - A control provided as a means of invoking device specific diagnostic tests, and subsequently determining the device status by means of the Device Status Report (DSR) control.

Device Status Report - A control used to inquire as to the current state of the terminal device. Note that some Device Status Reports (DSRs) are properly part of terminal management, while others pertain to presentation state or input processing.

Factory Default State - Digital specified default values for all state information. Factory Defaults are also the initial configuration of all setable state when the unit is shipped from the factory (manufacturing site).

Power-Up State - Factory defaults plus any non-volatile settings, such as Non-Volatile Memory (NVM), customization switches. The initial configuration of all setable state immediately following power-up.

Primary Device Attributes Control - The control used to request or report the implemented architectural conformance level and extensions.

Secondary Device Attributes Control - The control used to request or report the identification code of the specific product in which the interface is implemented.

Tertiary Device Attributes Control - The control used to request the identification code of the specific unit in which the interface is implemented.

Terminal Management Functions - A special class of device controls that operate across all layers of the Terminal Interface Architecture (TIA) and are not related to the presentation of data.

3 STATE DESCRIPTIONS

The algorithms in this standard provide a description of terminal state and how it is affected by various functions. They have been compiled for accuracy, but have not been formally certified. They should be viewed the same as other text in the standard.

3.1 DEVICE IDENTIFICATION

A conforming device shall provide an identification mechanism that returns the following information upon request.

a. The general characteristics (display service class) of the device, indicating the architectural level to which the device conforms

b. The architectural extensions that the device implements

c. A product specific code registered by the architecture

d. A revision level code, indicating the release level of the communicating firmware or software implementing the terminal management functions

e. A list, defined by each product implementation, of product specific options installed in the device at the time the identification request is received

f. For Level 4 and higher devices, a unique serial number to identify the specific unit

The following example illustrates the constants used for VT420 terminal identification.

```
/************** Module terminal_identification.h *************/
/*
 * Constants used for terminal identification
 * VT420 example
 */

/*
 * Extensions to the terminal interface architecture
 * Level 1 through Level 4              1-Feb-1990
 */
#define ONE_THIRTY_TWO_COLUMN     1
#define PRINTER_PORT              2
#define REGIS_GRAPHICS            3
#define SIXEL_GRAPHICS            4
#define KATAKANA_EXT              5
#define SELECTIVE_ERASE           6
#define DRCS_EXT                  7
#define UDK_EXT                   8
#define NRCS_EXT                  9
#define KANJI_EXT                10
#define STATUS_DISPLAY_EXT       11
#define SERBO_CROATION           12
#define BLOCK_MODE               13
#define EIGHT_BIT_IA             14
#define TCS                      15    /* Technical Char Set */
#define LOCATOR_PORT             16
#define TSI                      17    /* State Interrogation */
#define WINDOWING                18
#define MULTIPLE_SESSIONS        19    /* via TD/SMP */
#define APL                      20
#define HORIZONTAL_SCROLL        21
#define COLOR_TEXT               22
#define GREEK                    23
#define TURKISH                  24
#define ARABIC_BILINGUAL_M1      25
#define ARABIC_BILINGUAL_M2      26
#define ARABIC_BILINGUAL_M3      27
#define RECTANGULAR_EDITING      28
#define TEXT_LOCATOR             29
#define HANZI                    30
                               /* 31 not assigned */
#define TEXT_MACROS              32
#define HANGUL_HANJA             33
#define ICELANDIC                34
#define ARABIC_BILINGUAL_TC      35
#define ARABIC_BILINGUAL_NOTC    36
#define THAI                     37
#define CHARACTER_OUTLINING      38

#define NUMBER_OF_EXTENSIONS     39

/* device identification */
#define PRODUCT_ID     41     /* VT420 */
#define REVISION_LEVEL 10     /* Version 1.0 times 10 */

/* hardware options */
#define NUMBER_OF_OPTIONS 0   /* VT420 has no options */
```

3.2 TERMINAL STATE DECLARATIONS

The code example below illustrates type declarations for the terminal state. Type declarations in C do not allocate any storage. All these types can be defined regardless of the conformance level or extensions actually implemented.

```
/*************** Module terminal_state_types.h  *************/
/*
 * type declarations for terminal state       PAS 1-Feb-1990
 * VT420 example
 */

/*
 * terminal management
 */
#include "terminal_identification.h"    /* DA definitions */
typedef short int conformance_level_t;
#define LEVEL_1 1
#define LEVEL_2 2
#define LEVEL_3 3
#define LEVEL_4 4

typedef enum
    {           /* device_status_t */
    READY,
    MALFUNCTION
    } device_status_t;

typedef enum { LIMITED, UNLIMITED } transmit_rate_limit_mode_t;
typedef enum
    {           /* vt_configuration_t */
    SINGLE_SESSION,
    SESSIONS_ON_COMM,
    S1_HOST_S2_PRINTER,
    S2_HOST_S1_PRINTER
    /* VT330 with 2nd Host Port
    SESSIONS_ON_COMM2,
    S1_COMM1_S2_COMM2,
    S2_COMM1_S1_COMM2  */
    } vt_configuration_t;

typedef enum
    {          /* control_representation_mode_t */
    INTERPRET_CONTROLS,
    DISPLAY_CONTROLS
    } control_representation_mode_t;

/*
 * Code Extension Layer
 */
/* type declarations */
#include "gparse.h"    /* ANSI parser definitions */
typedef enum { ASCII_C }         c0_control_set_t;
typedef enum { SUPPLEMENTAL_C }  c1_control_set_t;
typedef enum
    {           /* graphic_character_set_t */
    ASCII_G,
    LINE_DRAWING_G,
    DEC_SUPPLEMENTAL_G,
    ISO_LATIN1_SUPPLEMENTAL_G,
    DRCS_G,
    NRCS_G,
    UPSS_G,
    TCS_G
```

```
    } graphic_character_set_t;
typedef enum { G0, G1, G2, G3 } gset_name_t;
typedef struct
    {           /* in_use_table_t */
    c0_control_set_t            c0;
    graphic_character_set_t     gl;
    gset_name_t                 invoked_gl;

    c0_control_set_t            c1;
    graphic_character_set_t     gr;
    gset_name_t                 invoked_gr;
    } in_use_table_t;
typedef enum { NO_SINGLE_SHIFT, SS2, SS3 } single_shift_t;
typedef enum
    {       /* in_control_string_t */
    NOT_IN_CONTROL_STRING,
    REGIS,
    SIXEL,
    DRCS_LOAD,
    UDK_LOAD,
    MACRO_LOAD,
    TERMINAL_STATE_LOAD,
    PRESENTATION_STATE_LOAD,
    SET_TERMINAL_UNIT_ID
    } in_control_string_t;
typedef enum { EIGHT_BIT, SEVEN_BIT } environment_t;
typedef enum { EIGHT_BIT_MULTINATIONAL, SEVEN_BIT_NATIONAL }
    character_set_mode_t;

#include "char_cell_types.h"    /* character cell display */

/*
 * Status Display Extension
 */
typedef enum { MAIN_DISPLAY, STATUS_DISPLAY } active_display_t;
typedef enum { NO_STATUS_DISPLAY, INDICATOR, HOST } status_display_t;

/*
 * keyboard processing
 */
typedef enum { UNLOCKED, LOCKED } keyboard_action_mode_t;
typedef enum { REPEAT_ON, REPEAT_OFF } auto_repeat_mode_t;
typedef enum { CURSOR, CK_APPLICATION } cursor_key_mode_t;
typedef enum { NUMERIC, KP_APPLICATION } keypad_mode_t;
typedef enum { TYPEWRITER, DATA_PROCESSING }
    keyboard_usage_mode_t;
typedef enum { CAPS_LOCK, SHIFT_LOCK } caps_shift_lock_mode_t;
typedef enum { HOLD_OFF, HOLD_ON } hold_screen_mode_t;
typedef enum { LOCK_OFF, LOCK_ON } lock_mode_t;
typedef enum { KEYCLICK_OFF, KEYCLICK_ON } keyclick_mode_t;
typedef enum { ECHO_ON, ECHO_OFF } send_receive_mode_t;
typedef enum { DELETE, BACKSPACE } backarrow_key_mode_t;
typedef enum { LOCAL, REPORT, IGNORE } local_key_mode_t;
typedef enum { LOCAL_F5, SHIFTED_ONLY, REPORT_F5, IGNORE_F5 }
    F5_key_mode_t;
typedef enum { CHARACTER, POSITION } keyboard_encoding_mode_t;
typedef short int keyboard_dialect_t;
#define KEYBOARD_UNDEFINED 0
#define NORTH_AMERICAN     1
#define BRITISH            2
#define FLEMISH            3
#define CANADIAN_FRENCH    4
#define DANISH             5
#define FINNISH            6
#define GERMAN             7
#define DUTCH              8
```

```
#define ITALIAN             9
#define SWISS_FRENCH       10
#define SWISS_GERMAN       11
#define SWEDISH            12
#define NORWEGIAN          13
#define BELGIAN_FRENCH     14
#define SPANISH            15
#define PORTUGUESE         16
#define CANADIAN_ENGLISH   17
#define MAX_NUM_DIALECTS   18

/*
 * printer port
 */
typedef enum { PRINTER_CONTROLLER_OFF, PRINTER_CONTROLLER_ON }
    printer_controller_mode_t;
typedef enum { AUTO_PRINT_OFF, AUTO_PRINT_ON } auto_print_mode_t;
typedef enum { PRINT_FF_OFF, PRINT_FF_ON } print_form_feed_mode_t;
typedef enum { PRINT_SCROLLING_REGION, PRINT_DISPLAY }
    print_extent_mode_t;
typedef enum { NO_PRINTER_TO_HOST, PRINTER_TO_HOST }
    printer_to_host_mode_t;
typedef enum { SHARED, SESSION1, SESSION2 }
    printer_assignment_t;
typedef enum { NATIONAL_ONLY, NATIONAL_PLUS_LINE_DRAWING,
               MULTINATIONAL_ONLY, ALL_CHARACTERS } printer_style_t;

/*
 * Graphics
 */
typedef enum { GRAPHICS_CURSOR_ON, GRAPHICS_CURSOR_OFF }
    graphics_cursor_enable_mode_t;    /* ReGIS */



/**************  module char_cell_types.h  *************/
/*
 * type declarations for character cell display state
 * PAS 28-Jul-1989
 */

typedef enum { JUMP, SLOW } scrolling_mode_t;
typedef enum { NORMAL_SCREEN, REVERSE_SCREEN } screen_mode_t;
typedef enum { WRAP_OFF, WRAP_ON } auto_wrap_mode_t;
typedef enum { TEXT_CURSOR_OFF, TEXT_CURSOR_ON }
    text_cursor_enable_mode_t;
typedef enum { INSERT, REPLACE } insert_replace_mode_t;
typedef enum { NEW_LINE_OFF, NEW_LINE_ON } new_line_mode_t;
typedef short int line_t;
typedef short int column_t;
typedef short int page_t;
typedef struct
    {         /* character_position_t */
    line_t line;
    column_t column;
    page_t page; /* L4 or Windowing */
    } character_position_t;
typedef enum { TRUE, FALSE } boolean_t;
typedef struct
    {         /* character_rendition_t */
    boolean_t bold;
    boolean_t blink;
    boolean_t underscore;
    boolean_t reverse;
    } character_rendition_t;
typedef struct
```

```
    {       /* character_attribute_t */
    boolean_t selective_erase;
    } character_attribute_t;
typedef enum
    {       /* line_rendition_t */
    DOUBLE_HEIGHT_TOP,
    DOUBLE_HEIGHT_BOTTOM,
    SINGLE_WIDTH,
    DOUBLE_WIDTH
    } line_rendition_t;
typedef struct
    {       /* character_t */
    short int code;
    character_rendition_t rendition;
    character_attribute_t attribute;
    graphic_character_set_t character_set;
    } character_t;
typedef struct
    {       /* char_set_designator_t */
    graphic_character_set_t name;
    short int first_intermediate;    /* 0-47, 0=no intermediate */
    short int second_intermediate;   /* 0-47 */
    short int final;                 /* 0 indicates null entry */
    } char_set_designator_t;
typedef enum { ABSOLUTE, DISPLACED } origin_mode_t;
typedef struct
    {      /* save_buffer_t */
    character_position_t position;
    character_rendition_t rendition;
    origin_mode_t origin_mode;
    graphic_character_set_t left;
    graphic_character_set_t right;
    graphic_character_set_t g0;
    graphic_character_set_t g1;
    graphic_character_set_t g2;
    graphic_character_set_t g3;
    character_attribute_t attribute;
    } save_buffer_t;
typedef enum { EIGHTY, ONE_THIRTY_TWO } column_mode_t;
typedef enum { UNCOUPLED, COUPLED } cursor_coupling_mode_t;
typedef enum { FIXED, SETABLE } left_right_margins_mode_t;
typedef enum { STREAM, RECTANGULAR } area_extent_t;
typedef short int lines_per_page_t;    /* 24, 25, 36, 72, 144 */
typedef short int columns_per_page_t;  /* 80, 132 */
typedef short int number_of_pages_t;   /* 1, 2, 3, 4, 5, 6 */
/************ end of char_cell_types.h  ****************/
```

The following example illustrates storage declarations for the terminal state. Only those state variables required for the desired conformance level and extensions need be present.

```
/**************  Module terminal_state_extern.h  *************/
/*
 * external storage declarations for terminal state
 * VT420 example                    PAS 28-Feb-1989
 */

/*
 * This code can also used to allocate storage
 * by redefining "ext" to be null.
 */
#define ext extern

#include "terminal_state_types.h"

/*
 * terminal management
 */
ext conformance_level_t conformance_level;
ext boolean_t level_1_extensions[NUMBER_OF_EXTENSIONS];
ext boolean_t level_2_extensions[NUMBER_OF_EXTENSIONS];
ext boolean_t level_3_extensions[NUMBER_OF_EXTENSIONS];
ext boolean_t level_4_extensions[NUMBER_OF_EXTENSIONS];
/* VT420 has no options --
ext boolean_t product_options[NUMBER_OF_OPTIONS];
--*/
ext long int terminal_unit_id;    /* Level 4, VT420 example */
ext device_status_t device_status;
ext transmit_rate_limit_mode_t transmit_rate_limit_mode;
ext boolean_t power_up_detected_flag;    /* Level 4 */
ext boolean_t error_detected_flag;       /* Level 4 */
ext vt_configuration_t vt_configuration; /* session mgmt ext */
ext control_representation_mode_t
        control_representation_mode;     /* Set-Up */
ext boolean_t line_local_mode;           /* Set-Up */

/*
 * Code Extension Layer
 */
ext struct parse_stack comm_state;      /* ANSI parser state */
ext graphic_character_set_t  designated_graphic_sets[4];
ext in_use_table_t           in_use_table;
ext single_shift_t           single_shift;
ext in_control_string_t      in_control_string;
ext graphic_character_set_t  upss;    /* Level 3 or 8-bit IA */
ext character_set_mode_t     character_set_mode;
ext character_set_mode_t     r_character_set_mode;
ext environment_t
    host_port_environment,
    c1_transmission_mode;

#include "char_cell_extern.h"    /* character cell display */
```

```
/*
 * keyboard processing
 */
ext keyboard_action_mode_t  keyboard_action_mode;
ext auto_repeat_mode_t      auto_repeat_mode;
ext cursor_key_mode_t       cursor_key_mode;
ext keypad_mode_t           keypad_mode;
ext keyboard_usage_mode_t   keyboard_usage_mode;     /* Level 3 */
ext caps_shift_lock_mode_t  caps_shift_lock_mode;
ext hold_screen_mode_t      hold_screen_mode;
ext lock_mode_t             lock_mode;
ext keyclick_mode_t         keyclick_mode;
ext send_receive_mode_t     send_receive_mode;
ext send_receive_mode_t     r_send_receive_mode;
ext backarrow_key_mode_t    backarrow_key_mode;      /* Level 3 */
ext local_key_mode_t                                 /* Level 4 */
        compose_key_mode,
        alt_key_mode,
        F1_key,
        F2_key,
        F3_key,
        F4_key;
ext F5_key_mode_t           F5_key_mode;             /* Level 4 */
ext keyboard_encoding_mode_t keyboard_encoding_mode; /* Level 4 */
ext keyboard_dialect_t      keyboard_dialect;           /* NRCS */
ext graphic_character_set_t nrcs_list[MAX_NUM_DIALECTS];/* NRCS */

/*
 * printer port extension
 */
ext printer_controller_mode_t  printer_controller_mode;
ext auto_print_mode_t          auto_print_mode;
ext print_form_feed_mode_t     print_form_feed_mode;
ext print_extent_mode_t        print_extent_mode;
ext printer_to_host_mode_t     printer_to_host_mode; /* Level 3 */
ext printer_assignment_t       printer_assignment;  /* sessions */
ext environment_t              printer_port_environment;
ext printer_style_t            printer_style;

/*
 * Graphics
 */
ext graphics_cursor_enable_mode_t
        graphics_cursor_enable_mode;   /* ReGIS */

/*
 * communications layer
 */
ext int host_port;     /* logical unit number for comm port */
ext int printer_port;

/**************  module char_cell_extern.h  *************/
/*
 * external storage declarations for character cell display state
 * PAS 28-Jul-1989
 */
```

```
#define MAX_NUM_LINES 73
#define MAX_NUM_COLUMNS 133
#define EMPTY_CHARACTER 0
#define MAX_NUM_CHAR_SETS 8
/* character set table entries */
#define TABLE_ASCII 0
#define TABLE_LINE_DRAWING 1
#define TABLE_DEC_SUPP 2
#define TABLE_NRCS 3
#define TABLE_LATIN1 4
#define TABLE_UPSS 5
#define TABLE_TCS 6
#define TABLE_DRCS 7
/* Note DRCS must be last in search order
   to replace others if identical */

ext character_t             display[MAX_NUM_LINES][MAX_NUM_COLUMNS];
ext line_rendition_t        line_rendition[MAX_NUM_LINES];
ext character_position_t    active_position;
ext line_t                  top_margin;
ext line_t                  bottom_margin;
ext column_t                left_margin;    /* Horiz scroll ext */
ext column_t                right_margin;   /* Horiz scroll ext */
ext boolean_t               last_column_flag;
ext active_display_t        active_display; /* L3 or Status Dsply*/
ext status_display_t        status_display; /* L3 or Status Dsply*/
ext char_set_designator_t   char_set_table[MAX_NUM_CHAR_SETS];
ext save_buffer_t           cursor_save_buffer;
ext character_rendition_t   current_rendition;
ext character_attribute_t   current_attribute; /* Selective Erase*/
ext boolean_t               horizontal_tab_stops[MAX_NUM_COLUMNS];
ext column_mode_t           column_mode;      /* 132 Column Ext */
ext lines_per_page_t        lines_per_page;   /* Windowing */
ext columns_per_page_t      columns_per_page; /* Windowing */
ext number_of_pages_t       number_of_pages;  /* Windowing */
ext scrolling_mode_t        scrolling_mode;
ext screen_mode_t           screen_mode;
ext origin_mode_t           origin_mode;
ext auto_wrap_mode_t        auto_wrap_mode;
ext text_cursor_enable_mode_t  text_cursor_enable_mode;
ext insert_replace_mode_t   insert_replacement_mode;
ext new_line_mode_t         new_line_mode;
ext cursor_coupling_mode_t
        horizontal_cursor_coupling,      /* Windowing */
        vertical_cursor_coupling,        /* Windowing */
        page_cursor_coupling;            /* Windowing */
ext left_right_margins_mode_t
        left_right_margins_mode; /* Horiz Scroll */
ext area_extent_t
        attribute_change_extent; /* L4 or rect editing */
/********** end of char_cell_extern.h  *************/
```

3.3 STATUS AND TEST

Upon request, a conforming device shall provide at the terminal interface a means of returning device status information. Status information consists of either "Device Ready" or "Device Malfunction". (See subhead 5, DEVICE STATUS REPORT.) The device status shall be set upon power-up initialization or receipt of RESET TO INITIAL STATE (RIS). See *Video Systems Reference Manual - Documented Exceptions.*

The architecture does not define the nature or duration of specific device test sequences. The architecture provides a registry of test invocation control functions (Refer to *DEC STD 138-0 Registry of Control Functions for Character-Imaging Devices*) that may have different effects on different products. However, failure of any applicable test sequences shall set the device status to the "Device Malfunction" state.

3.3.1 Virtual Terminal Configuration

If the Multiple Sessions extension is supported, a conforming device shall provide a means at the terminal interface of returning information on whether the terminal is configured to support virtual terminals. Status information returned shall be in the form of a Device Status Report (DSR) (see subhead 5, Device Status Report) and consist of: "Virtual Terminals Not Enabled", "Virtual Terminals Enabled Over Physical Lines", "Virtual Terminals Available Through TD/SMP".

This feature is intended for terminal management use. Specific information on the state of the Terminal Device/Session Management Protocol (TD/SMP) or other session management protocol should be ascertained through the protocol handler.

3.3.2 Data Integrity

At Conformance Level 4, a conforming device shall provide a means at the terminal interface of returning the status of a data integrity flag. Status information returned shall be in the form of a Device Status Report (DSR) (see subhead 5 Device Status Report) and consist of: "OK, no comm error detected since last report", "Error, a comm error has been detected since last report", or "Power-up, No request since last power-up or reset".

This is an application feature for defensive programming, and not intended for terminal management or communications use.

4 DEVICE INITIALIZATION

On power up, all state information (state) in the terminal shall be set to a known value, or value previously selected by the user and stored through non-volatile means. This state is referred to as the Power-Up State.

Since the architecture allows the Power-Up State of certain features to be set and stored under local control (Set-Up and non-volatile memory for example), software cannot rely on the values of this state after an appropriate initialization sequence is executed or after the device is powered up. The values given in the algorithm below apply only to those implementations in which the state cannot be set and stored locally by the terminal user. The state defined by the architecture to which this indeterminate condition may apply is limited to the following:

```
conformance_level

host_port_environment
vt_configuration
c1_transmission_mode
transmit_rate_limit_mode
terminal_unit_id
character_set_mode
upss
columns_per_page
lines_per_page
number_of_pages
status_display

column_mode
scrolling_mode
screen_mode
auto_wrap_mode
text_cursor_enable_mode
new_line_mode
horizontal_tab_stops

vertical_cursor_coupling
horizontal_cursor_coupling
page_cursor_coupling

auto_repeat_mode
cursor_key_mode
caps_shift_lock_mode

keyboard_usage_mode
backarrow_key_mode
alt_key_mode
compose_key_mode
F1_key_mode
F2_key_mode
F3_key_mode
F4_key_mode

auto_print_mode

printer_controller_mode

print_form_feed_mode

print_extent_mode
printer_to_host_mode
printer_style

graphics_cursor_enable_mode
UDK definitions (see DEC-STD-070-11, UDK Extension)
```

State Affected:

All terminal state. See subhead 3.2, Terminal State Declarations.

The following code example illustrates terminal management routines.

Algorithm:

```
/************** Module terminal_management.c *************/
/*
* terminal management routines
* VT420 example
*/

/* get access to global terminal state */
#include "terminal_state_extern.h"
```

```
/* function declarations */
void hardware_init();
void comm_init();
void setup_init();
void host_state_init();
void soft_terminal_reset();
int send_char();
int send_string();
int send_int();
int send_hex();

void power_up_initialization()
{
hardware_init();      /* actual hardware as needed */
comm_init();          /* communications layer */
setup_init();         /* features that are not host setable */
host_state_init();    /* all host setable state */
}   /* end procedure power_up_initialization */

void setup_init()
/* initialize local features that are not host setable */
{
host_port_environment = EIGHT_BIT;      /* NVM */
vt_configuration = SINGLE_SESSION;      /* NVM */
control_representation_mode =
INTERPRET_CONTROLS;
printer_port_environment = EIGHT_BIT;   /* NVM */
printer_style = NATIONAL_ONLY;          /* NVM */
}   /* end procedure setup_init */

void host_state_init()
{
line_t   x;
column_t y;
/*.................*/

/* do initialization */

/* Terminal Management */
conformance_level = LEVEL_4;            /* NVM */
transmit_rate_limit_mode = LIMITED;     /* NVM */
power_up_detected_flag = TRUE;
error_detected_flag = FALSE;
vt_configuration = SINGLE_SESSION;      /* NVM */
/* VT420 terminal unit ID example (different for each unit) */
terminal_unit_id = 0x01000001;    /* HKO unit 1,  NVM */

/* Code Extension */
upss = DEC_SUPPLEMENTAL_G;              /* NVM */

in_use_table.c0 = ASCII_C;
in_use_table.gl = ASCII_G;
in_use_table.invoked_gl = G0;

in_use_table.c1 = SUPPLEMENTAL_C;
in_use_table.gr = upss;
in_use_table.invoked_gr = G2;

designated_graphic_sets[0] = ASCII_G;
designated_graphic_sets[1] = ASCII_G;
```

```
switch (conformance_level)
    {
     case LEVEL_1:
      in_use_table.gr = ASCII_G;
      designated_graphic_sets[2] = ASCII_G;
      designated_graphic_sets[3] = ASCII_G;
      c1_transmission_mode = SEVEN_BIT;
      character_set_mode = SEVEN_BIT_NATIONAL;
      break;
     case LEVEL_2:
     case LEVEL_3:
     case LEVEL_4:
      designated_graphic_sets[2] = upss;
      designated_graphic_sets[3] = upss;
      c1_transmission_mode = EIGHT_BIT;              /* NVM */
      character_set_mode = EIGHT_BIT_MULTINATIONAL;  /* NVM */
    };

parse_init(comm_state);  /* initialize ANSI parser */
single_shift = NO_SINGLE_SHIFT;
in_control_string = NOT_IN_CONTROL_STRING;

host_port_environment = EIGHT_BIT;              /* NVM */

/* character cell display */
for (y=1; y<MAX_NUM_LINES; y++)
    {
    for (x=1; x<MAX_NUM_COLUMNS; x++)
     {
     display[y][x].code = EMPTY_CHARACTER;
     }
    line_rendition[y] = SINGLE_WIDTH;
    }
column_mode = EIGHTY;                           /* NVM */
scrolling_mode = SLOW;                          /* NVM */
screen_mode = NORMAL_SCREEN;                    /* NVM */
origin_mode = ABSOLUTE;
auto_wrap_mode = WRAP_OFF;                      /* NVM */
text_cursor_enable_mode = TEXT_CURSOR_ON;       /* NVM */
insert_replacement_mode = REPLACE;              /* NVM */
new_line_mode = NEW_LINE_OFF;                   /* NVM */
active_display = MAIN_DISPLAY;
status_display = NO_STATUS_DISPLAY;             /* NVM */
horizontal_cursor_coupling = UNCOUPLED;         /* NVM */
vertical_cursor_coupling = COUPLED;             /* NVM */
page_cursor_coupling = COUPLED;                 /* NVM */
left_right_margins_mode = FIXED;
attribute_change_extent = STREAM;
lines_per_page = 24;                            /* NVM */
number_of_pages = MAX_NUM_LINES/lines_per_page;

active_position.line = 1;
active_position.column = 1;
active_page = 1;
top_margin = 1;
bottom_margin = lines_per_page;
left_margin = 1;
/* if Windowing Extension,
    columns_per_page in NVM will override column_mode */
if (column_mode == EIGHTY) columns_per_page = 80;  /* NVM */
    else columns_per_page = 132;
right_margin = columns_per_page;
```

```
current_rendition.bold       = FALSE;
current_rendition.blink      = FALSE;
current_rendition.underscore = FALSE;
current_rendition.reverse    = FALSE;
current_attribute.selective_erase = FALSE;

horizontal_tab_stops[1] = FALSE;                /* NVM */
for (x=2; x<MAX_NUM_COLUMNS; x++)
    {
    if (x%8 == 1) horizontal_tab_stops[x] = TRUE;
    else horizontal_tab_stops[x] = FALSE;
    }

save_cursor();  /* initialize cursor save buffer */
macro_init();   /* initialize text macros (Level 4) */

/* keyboard processing */
keyboard_action_mode = UNLOCKED;
auto_repeat_mode = REPEAT_ON;                   /* NVM */
cursor_key_mode = CURSOR;                       /* NVM */
keypad_mode = NUMERIC;
keyboard_usage_mode = TYPEWRITER;               /* NVM */
caps_shift_lock_mode = CAPS_LOCK;               /* NVM */
hold_screen_mode = HOLD_OFF;
lock_mode = LOCK_OFF;
keyclick_mode = KEYCLICK_ON;                    /* NVM */
send_receive_mode = ECHO_OFF;                   /* NVM */
backarrow_key_mode = DELETE;                    /* NVM */
compose_key_mode = LOCAL;                       /* NVM */
alt_key_mode = REPORT;                          /* NVM */
F1_key = LOCAL;                                 /* NVM */
F2_key = LOCAL;                                 /* NVM */
F3_key = LOCAL;                                 /* NVM */
F4_key = LOCAL;                                 /* NVM */
F5_key_mode = LOCAL_F5;                         /* NVM */
keyboard_encoding_mode = CHARACTER;

/* printer port */
printer_port_environment = EIGHT_BIT;           /* NVM */
printer_controller_mode =
      PRINTER_CONTROLLER_OFF;                    /* NVM */
auto_print_mode = AUTO_PRINT_OFF;               /* NVM */
print_form_feed_mode = PRINT_FF_OFF;            /* NVM */
printer_to_host_mode = NO_PRINTER_TO_HOST;      /* NVM */
print_extent_mode = PRINT_DISPLAY;              /* NVM */
printer_assignment = SHARED;

/* initialize other extensions */
graphics_cursor_enable_mode = GRAPHICS_CURSOR_ON; /* NVM */
udk_init();                                     /* NVM */
drcs_init();

/* terminal management */
device_status = READY;

};    /* end procedure host_state_init */
```

5 CONTROL FUNCTIONS

DEVICE ATTRIBUTES (Primary) DA

Levels: 1x, 2, 3, 4

Purpose: Request or report the device attributes.

Request Format:

CSI	Ps	c	default Ps: 0
9/11	Ps	6/3	

Response Format:

CSI	?	Ps	c
9/11	3/15	Ps	6/3

Description: Use the primary Device Attributes (DA) control to request or report the implemented architectural conformance level and extensions. An omitted parameter or a parameter value of zero (0) indicates a request for identification. The response to the primary DA request consists of request for a DA control sequence using a private parameter string with the introducer 3/15 (?) followed by a variable number of parameters.

The first parameter in the string indicates the basic service class of the device and the architectural level to which the device conforms. The rest of the parameters indicate the architectural extensions supported at the current operating level.

NOTE

Exceptions to the architecture that are documented as appendices to these standards are not reported in the primary DA response.

The value of the first parameter is encoded so that a simple range check can be performed to determine the basic service class and level of the device. Each service class is designated a range of decimal values from 1 to 9. For example, Character Cell Display devices are in the range 61 to 69. The architectural level to which the device conforms is indicated by the low order digit. Thus, a response with a first parameter of 61 indicates that the device conforms to Level 1 of the Character Cell Display architecture. Higher levels, by definition, support the functionality of lower levels of the same service class.

Therefore, if a device code of 63 is returned, the device may be supported as a Level 1, Level 2, or Level 3 Character Cell Display device. This does not mean higher conformance levels are strict supersets of lower conformance levels, but that a device that supports a higher conformance level must support the lower levels as well (using DECSCL).

In general, higher conformance levels are supersets of lower levels, but specific control functions can be interpreted differently to accommodate evolving requirements. Conforming software shall select a known conformance level to guarantee compatibility. See the description of Select Conformance Level (DECSCL) in this document for further information.

Note that the device code in the response represents the highest level that the device can support, and does not indicate the level at which the device is currently operating. For example, a Level 3 conforming device operating at Level 2, as selected by the Select Conformance Level control function, would still respond with a device code of 63.

Table 1 lists extensions to the character cell display service class registered within Digital that may be transmitted by a device in response to an appropriate request. Levels 1 - 4 (DA parameters 61 - 64) have been defined for the character cell display service class.

Table 1: Registered Extensions to the Character Cell Display Service Class

Extension Parameter	Levels	Extension Name
1	1 - 4	132 Column Display
2	1 - 4	Printer Port
3	1 - 4	ReGIS Display
4	1 - 4	Sixels Display
5	1 - 3	Katakana
6	2 - 4	Selectively Erasable Characters
7	2 - 4	Dynamically Redefinable Character Sets
8	2 - 4	User Defined Keys
9	1 - 4	National Replacement Character Sets
10	1 - 3	Kanji
11	2	Status Display
12	1 - 3	Serbo-Croation
13	1 - 3	Block Mode
14	2	8-bit Interface Architecture
15	2 - 4	Technical Character Set
16	1 - 4	Locator Port
17	2	Terminal State Interrogation
18	2 - 4	Windowing
19	2 - 4	Multiple Sessions (TD/SMP)
20	1 - 4	APL
21	2 - 4	Horizontal Scrolling
22	2 - 4	Color Text
23	1 - 3	Greek
24	1 - 3	Turkish
25	2, 3	Arabic Bilingual Mode 1
26	2, 3	Arabic Bilingual Mode 2
27	2, 3	Arabic Bilingual Mode 3
28	2, 3	Rectangular Editing
29	2 - 4	Text Locator
30	2, 3	Hanzi
31		
32	2 - 4	Text Macros
33	2, 3	Hangul and Hanja
34	2, 3	Icelandic
35	2, 3	Arabic Bilingual with Text Controls
36	2, 3	Arabic Bilingual with no Text Controls
37	2, 3	Thai
38	2, 3	Character Outlining

Implicit Extensions

Implicit Extensions are features originally defined at a higher conformance level that still operate when a lower conformance level is selected and are not required to be explicitly reported. The "8-bit Interface Architecture" extension, for example, was originally defined in Level 3 and as an extension to Level 2. A Level 3 device operating at Level 2 need not disable the 8-bit Interface Architecture Extension, nor report it explicitly as a parameter in the DA response. Level 3 or higher defines 8-bit Interface Architecture to be an implicit extension to Level 2.

Each conformance level may define a set of implicit extensions at lower levels. Higher conformance levels must support all the implicit extensions defined by lower levels. Implicit extensions must only be allowed if they are fully backward compatible with the lower conformance level.

Software can determine whether an implicit extension is present by seeing if the reported conformance level is greater than or equal to the conformance level at which the feature was originally defined.

If a higher conformance level subsumes extensions previously reported at a lower level, these extensions shall be reported at the lower level so that existing software can determine if the extension is present.

Defining implicit extensions allows features defined by a higher conformance level to operate at a lower conformance level if they are consistent with that level.

Level Reported	Operating Level	Implicit Extension
2 or higher	1	DECSCL
3 or higher	2	11 Status Display
	2	14 8-bit Interface Architecture
	2	17 Terminal State Interrogation
4 or higher	1-3	DECSR

State Affected: None

Algorithm:

```
void device_attributes_1()
{
int n;
send_char(0x9B, host_port);
send_string("?64", host_port);
if (conformance_level == LEVEL_1)
    for (n=1; n<NUMBER_OF_EXTENSIONS; n++)
     if (level_1_extensions[n] == TRUE)
      {
      send_char(';', host_port);
      send_int(n, host_port);
      };
if (conformance_level == LEVEL_2)
    for (n=1; n<NUMBER_OF_EXTENSIONS; n++)
     if (level_2_extensions[n] == TRUE)
      {
      send_char(';', host_port);
      send_int(n, host_port);
      };
if (conformance_level == LEVEL_3)
    for (n=1; n<NUMBER_OF_EXTENSIONS; n++)
     if (level_3_extensions[n] == TRUE)
      {
      send_char(';', host_port);
      send_int(n, host_port);
      };
if (conformance_level == LEVEL_4)
    for (n=1; n<NUMBER_OF_EXTENSIONS; n++)
     if (level_4_extensions[n] == TRUE)
      {
      send_char(';', host_port);
      send_int(n, host_port);
      };
send_char('c', host_port);
};
```

Known Deviations:

Terminals designed before this architecture was developed do not conform to the above encoding scheme. They will, however, respond to the Device Attributes request by returning a DA control with a variable length parameter string. The first parameter of this parameter string is a device-specific code in the range 1 to 59. Thus, these devices are easily distinguished from those that implement the architectural response, whose first parameter is always 60 or greater.

Some terminals provide an alias ID (identification) capability, selectable under local control, to emulate the DA response of older terminals. The factory default will always be to respond with the correct identification code for the family of devices in which the interface is implemented.

The Greek and Turkish special versions of the VT200 and VT300 series use extension parameters 10 and 11 instead of the registered parameter values 23 and 24, respectively.

SELECT CONFORMANCE LEVEL DECSCL

Purpose: Select a particular level of operation for interface compatibility.

Levels: 1x, 2, 3, 4

Format:

```
CSI         Ps          ;          Ps          "          p
9/11        Ps          ;          Ps          2/2        7/0
```

Description: The Select Conformance Level (DECSCL) control selects a terminal operating mode that provides interface compatibility defined by a conformance level of the architecture. The operations and state information available in each level are fully defined in the architectural specifications. The first parameter selects the conformance level of operation. The optional second parameter applies only to Level 2 (it will be ignored in Level 1 operation). The second parameter may be used to select the C1 control transmission for 7-bit or 8-bit controls, in accordance with the following table:

```
0 - default (8-bit C1 controls)
1 - 7-bit C1 controls
2 - 8-bit C1 controls
```

NOTES

1. **The DECSCL control function is required only in Level 2. However, in Level 2 (and higher) conforming devices, the DECSCL control must also be executed when the device is in Level 1 operation. This is an exception to the normal rule that Level 2 functions only work in Level 2 operation, and is necessary to enable switching between levels. DECSCL is an Implicit Extension to Level 1.**
2. **At the present time, four conformance levels are defined by the Terminal Interface Architecture (TIA):**
   61 - Level 1 Character Cell Display Terminals
   62 - Level 2 Character Cell Display Terminals
   63 - Level 3 Character Cell Display Terminals
   64 - Level 4 Character Cell Display Terminals

   Software should be designed to anticipate future higher-numbered levels that will be supersets of Level 1-4.
3. **Selecting a particular conformance level also causes a "soft reset" of the terminal to be performed. This automatically places the terminal into a known state, without affecting the data contained in the display. See Soft Terminal Reset, DECSTR, for more information.**
4. **Attempts to select conformance levels that are unsupported by the device will be ignored.**

State Affected:
conformance_level
c1_transmission_mode

Algorithm:

```
void select_conformance_level(service_class, c1_mode)
    int service_class;
    int c1_mode;
{
switch (service_class)
    {
    case 61:
     conformance_level = LEVEL_1;
     break;
    case 62:
     conformance_level = LEVEL_2;
     break;
    case 63:
     conformance_level = LEVEL_3;
     break;
    case 64:
     conformance_level = LEVEL_4;
     break;
    default: return;
    };
soft_terminal_reset();
switch (c1_mode)
    {
    case 0:
    case 2:
     c1_transmission_mode = EIGHT_BIT;
     break;
    case 1:
     c1_transmission_mode = SEVEN_BIT;
    };
   };    /* end procedure select_conformance_level() */
```

Known Deviations: None

Levels: 1X, 2, 3, 4

Purpose: Request or report the device identification.

Request Format:

CSI	>	Ps	c	Default Ps: 0
9/11	3/14	Ps	6/3	

Response Format:

CSI	>	Ps	c
9/11	3/14	Ps	6/3

Description: The DA2 control requests or reports the identification code of the specific product in which the interface is implemented. The request consists of a DA control sequence with the private parameter code 3/14 (>) and an omitted parameter or a parameter value of zero (0).

The response to the DA2 request consists of a DA control sequence with the private parameter code 3/14 (>) followed by a variable number of parameters. The first parameter indicates the product specific identification of the device. This parameter value shall be registered with the architecture.

The second parameter indicates the revision level of the firmware or software implementing the terminal management functions. Revision levels are generally noted by a real number with one significant decimal. This number should be (not mandatory) multiplied by 10 and used as the value of the second parameter.

Additional parameters may be present to indicate options installed in the device. These options are implementation defined, and the number and meaning of the additional parameter values will be product dependent, however, these values shall be registered with the architecture.

State Affected: None

Algorithm:

```
void device_attributes_2()
{
int n;
send_char(0x9B, host_port);    /* CSI */
send_int(PRODUCT_ID, host_port);
send_char(';', host_port);
send_int(REVISION_LEVEL, host_port);
/* Device IDs and options must be registered */
/* VT420 has no options --
for (n=1; n<NUMBER_OF_OPTIONS; n++)
    {
    if (product_options[n] == TRUE)
     {
      send_char(';', host_port);
      send_int(n, host_port);
     }
    };
--*/
send_char('c', host_port);
};    /* end procedure device_attributes_2() */
```

Known Deviations:

Terminals designed before the Terminal Interface Architecture (TIA) was developed do not implement this control. Terminals that return a first parameter value less than 60 to the primary Device Attributes (DA) will ignore the secondary Device Attributes (DA2) control.

DEVICE ATTRIBUTES (Tertiary) DA3
REPORT TERMINAL UNIT ID DECRPTUI

Level: 4

Purpose: Request terminal unit identification.

Request Format:

CSI	=	c
9/11	3/13	6/3

Description: The Tertiary Device Attributes Request (DA3) causes the terminal to report its unique terminal unit ID to the host using the DECRPTUI (Report Terminal Unit ID) sequence. The terminal unit ID consists of a string of up to 31 displayable characters that are factory set to a unique value for the specific product (DA2 registration) in which the interface is implemented.

Individual products may further restrict the length or format of the terminal unit ID as long as the identifier continues to provide a unique ID within that specific product. The format of the Terminal Unit ID shall be registered with the architecture.

Report Terminal Unit ID

Format:

DCS	!	|	D. . .D	ST
9/0	2/1	7/12	?/?	9/12

Implementation Guideline

The VT420 terminal unit ID consists of a 32-bit number reported as four hex pairs. The first hex pair (8 bits) contains the manufacturing site code, and the remaining three hex pairs (24 bits) contain a serial number for that site. The site code and serial number together provide a unique ID for each VT420. See DEC STD 138-0 for a listing of manufacturing site codes.

The terminal unit ID is factory set and does not change unless the terminal's internal circuitry is repaired or replaced. In the VT420, the unit ID is maintained in Non-Volatile Memory (NVM) and can be rewritten in the event of NVM failure using the DECSTUI (Set Terminal Unit ID) sequence. This sequence is only recognized when the NVM checksum is invalid, or when an internal jumper on the main board is set.

Set Terminal Unit ID (DECSTUI) Format:

DCS	!	{	D. . .D	ST
9/0	2/1	7/11	?/?	9/12

State Affected: None

Algorithm:

```
void device_attributes_3()     /* VT420 example */
{
    int n;
send_char(0x90, host_port);     /* DCS */
send_string("!|", host_port);
n = (terminal_unit_id >> 24) & 0xFF;
send_hex(n, host_port);
n = (terminal_unit_id >> 16) & 0xFF;
send_hex(n, host_port);
n = (terminal_unit_id >>  8) & 0xFF;
send_hex(n, host_port);
n = terminal_unit_id & 0xFF;
send_hex(n, host_port);
send_char(0x9C, host_port);     /* ST */
}   /* end procedure device_attributes_3() */
```

Known Deviations: None

Levels: 1, 2

Purpose: Request the device identification code.

Format:

ESC	Z
1/11	5/10

Description: The DECID control requests that the terminal report the implemented architectural conformance level and extensions. The device responds by transmitting an appropriate primary Device Attributes (DA) response.

NOTES

a. **This control is identical to a primary DA control with an omitted parameter or a parameter value of zero (0). THIS CONTROL IS RESERVED FOR FUTURE STANDARDIZATION BY ISO AND ANSI. Conforming software shall use the primary Device Attributes (DA) control for purposes of device identification.**

b. **DECID is not recognized at Level 3 or above.**

c. **Terminals that implement the VT52 emulation hardware option respond differently to this control when in VT52 mode. See *Video Systems Reference Manual - VT52 Emulation* for more information.**

State Affected: None

Algorithm:

```
/*
* DECID    ESC Z
* Ignored above Level 2
* execution is same as primary DA
*/
void id()
{
if ( (conformance_level == LEVEL_1) ||
    ( (conformance_level == LEVEL_2) )
     device_attributes_1();
};
```

Known Deviations:

The VT300 family and VT420 do not have a fully independent VT200 mode, and ignore DECID when set to conformance Level 2.

See additional deviations under Device Attributes (Primary).

SECURE RESET DECSR
SECURE RESET CONFIRMATION DECSRC

Levels: 4

Purpose: To ensure that the terminal is in a known state so that data sent to the terminal is displayed as expected, and keystrokes from the terminal transmit as expected, and to erase any host-writeable data in the terminal.

Format:

CSI	Pn	+	P
9/11	3/?	2/11	7/0

Description: Secure Reset initializes the terminal to its Power-Up state and erases all host setable state information and data. If a non-zero numeric parameter in the range 1 to 16383 is provided, the device responds with a Secure Reset Confirmation (DECSRC) control function with the same parameter upon successful completion. If Pn is omitted or zero, no response is generated. If Pn exceeds 16383, the entire sequence shall be ignored.

Secure Reset Confirmation

Format:

CSI	Pn	*	q
9/11	3/?	2/10	7/1

NOTES

1. **DECSR is recognized at all conformance levels (Level 1 or higher).**
2. **DECSR is recognized in Control Representation Mode (Display Controls Mode, CRM set) and causes the device to exit this mode. This is desirable because CRM is enabled under local control in video terminals and there is no other way to clear it from the host. If the terminal is in Display Controls Mode, DECSR shall be displayed before it is executed.**
3. **DECSR is not recognized in VT52 Mode (DECANM reset), or Tektronix 4010/4014 Emulation Mode (DECTEK). Host software must explicitly exit these modes. This is necessary because these modes do not use ANSI compatible syntaxes.**
4. **DECSR is not recognized (intercepted) in Printer Controller Mode (see Media Copy - MC). Host software must explicitly exit this mode. Secure Reset may be sent to an attached printer in Printer Controller Mode.**
5. **DECSR is buffered in the same way as other incoming data.**
6. **In a multiple-session device, DECSR only affects the session that receives the control function. At the time of this update to the standard, secure connections are only supported in single-session devices. Reconfiguring the device from a single-session to multiple-sessions causes a BREAK to be generated to alert host software.**

7. **Secure Reset need not erase data maintained in Non-Volatile Memory (NVM) as long as such data cannot be written from the host. If values for some state information can be stored under host control in NVM, then DECSR should erase this NVM (not mandatory).**
8. **DECSTUI, which sets the terminal unit ID, is an exception to the previous note since it is only recognized if the NVM checksum is invalid, or an internal jumper has been installed. In normal operation, the terminal unit ID cannot be written from the host.**

State Affected:

Secure Reset affects all host setable state similar to power-up initialization. See subhead 4, DEVICE INITIALIZATION.

The following state information is also explicitly reset:

UDKs
The UDKs are erased to be empty, or restored to the value saved in NVM if such NVM is present and not writable from the host. The UDK definitions are re-initialized even if the "UDKs Locked" state is in effect.
DRCS
Macro Definitions
clears the keyboard input silo
erases all local buffers that might contain data from host (such as paste_buffer, capture_buffer, etc...)

Secure Reset does not affect host port receive buffers, nor session management state (if present).

Algorithm:

```
void secure_reset(param)
    int param;
{
if (param <= 16383)     /* 0x3FFF */
    {
    /* exit CRM */
    control_representation_mode = INTERPRET_CONTROLS;

    /* initialize all host setable state */
    host_state_init();
    clear_udk();

    /* report reset was successful if non-zero parameter */
    if (param != 0)
     {
     send_char(0x9B, host_port);
     send_int(param, host_port);
     send_string("*q", host_port);
     };
    };
};    /* end procedure secure_reset() */
```

Known Deviations: None

SOFT TERMINAL RESET DECSTR

Levels: 2, 3, 4

Purpose: Reset soft terminal state to the default values.

Format:

```
CSI             !             P
9/11            2/1           7/0
```

Description: The DECSTR control sets the modes available to user application programs to known values. User Preference Features are not affected. DECSTR is intended for application use, whereas User Preference Feature control functions are intended for terminal management use. When the DECSTR control is received, the following state information in the terminal shall be reset as indicated below. This list defines the Soft Terminal Reset state.

State	Default
Top Margin	1
Bottom Margin	number of lines on page
Left Margin	1
Right Margin	80 (or 132 if column mode is set)
Current Rendition	normal
Current Attributes	not selective erasable
In Use Table (gl, gr)	set to current level defaults
Designated Sets	set to current level defaults
Cursor Save Buffer	set to current level defaults
Insert/Replacement Mode	reset (Replace)
Origin Mode	reset (Absolute)
Auto Wrap Mode	Auto Wrap Off (NVM if present)
Character Set Mode	8-bit Multinational (ignore NVM)
Cursor On/Off	on (ignore NVM)
Active Display	Main Display (status display is exited)
UPSS	DEC Supplemental (NVM if present)
Cursor Key Mode	reset (Cursor)
Keyboard Action Mode	reset (Unlocked)
Keypad Mode	reset (Numeric)
Keyboard Encoding Mode	character

The following User Preference Features are NOT affected by the soft reset.

Column Mode
Scroll Mode
Screen Mode
Auto Repeat Mode
Caps/Shift Lock Mode
Tab stops
Key click

The following are also NOT affected by the soft reset.

The data in the display
New Line Mode
Line Attributes
Printer Controller Mode
Auto Print Mode
Compose Mode
C1 Transmission Mode
Conformance Level
DRCS
UDK
Macro Definitions

NOTES

a. **Numeric is the default state for Keypad Mode. It is the power-up default state for the terminal. It is preferred that operating systems perform translation of keypad codes for applications software at the time data is passed to the application. This allows applications to clear the type-ahead buffer to ensure synchronization when changing keypad mode. Operating systems implementing this strategy should set Keypad Mode to Application after performing the soft reset and do the required translation.**

b. **The DECSTR control is ignored when operating at Level 1 and by Level 1 devices.**

c. **DECSTR has no affect on the Active Position.**

d. **When the terminal is in Printer Controller Mode, the DECSTR control is not executed by the terminal, but is transmitted instead to the printer port. Thus, DECSTR does not reset the terminal state unless Printer Controller Mode is explicitly reset before DECSTR is transmitted.**

e. **DECSTR is interpreted differently in printing devices to achieve the corresponding concept of resetting the device to known application state while preserving the data currently on the display (previously printed, or buffered in the printer).**

State Affected:

See subhead 4, DEVICE INITIALIZATION for type declarations.

```
/* VT420 example */
/* code extension layer state */
graphic_character_set_t  designated_graphic_sets[4];
in_use_table_t           in_use_table;
character_set_mode_t     character_set_mode;      /* NRCS */
graphic_character_set_t  upss           /* L3 or 8-bit IA */

/* character cell display state */
line_t                   top_margin;
line_t                   bottom_margin;
column_t                 left_margin;   /* Horizontal Scroll */
column_t                 right_margin;  /* Horizontal Scroll */
left_right_margins_mode_t  left_right_margins_mode; /* H Scroll */
active_display_t         active_display;  /* L3 or status display */
save_buffer_t            cursor_save_buffer;
character_rendition_t    current_rendition;
character_attribute_t    current_attribute;  /* Selective Erase */
origin_mode_t            origin_mode;
auto_wrap_mode_t         auto_wrap_mode;
text_cursor_enable_mode_t  text_cursor_enable_mode;
insert_replace_mode_t    insert_replacement_mode;
new_line_mode_t          new_line_mode;

/* keyboard processing state */
keyboard_action_mode_t   keyboard_action_mode;
cursor_key_mode_t        cursor_key_mode;
keypad_mode_t            keypad_mode;
keyboard_encoding_mode_t keyboard_encoding_mode;  /* L4 */
```

Algorithm:

```
void soft_terminal_reset()
{
if (conformance_level != LEVEL_1)
    {
    /* Code Extension */
    upss = DEC_SUPPLEMENTAL_G;                      /* NVM */
    in_use_table.c0 = ASCII_C;
    in_use_table.gl = ASCII_G;
    in_use_table.invoked_gl = G0;
    in_use_table.c1 = SUPPLEMENTAL_C;
    in_use_table.gr = upss;
    in_use_table.invoked_gr = G2;
    designated_graphic_sets[0] = ASCII_G;
    designated_graphic_sets[1] = ASCII_G;
    designated_graphic_sets[2] = upss;
    designated_graphic_sets[3] = upss;
    character_set_mode = EIGHT_BIT_MULTINATIONAL; /* NVM */

    /* character cell display */
    origin_mode = ABSOLUTE;
    auto_wrap_mode = WRAP_OFF;                      /* NVM */
    text_cursor_enable_mode = TEXT_CURSOR_ON;       /* NVM */
    insert_replacement_mode = REPLACE;              /* NVM */
    new_line_mode = NEW_LINE_OFF;                   /* NVM */
    active_display = MAIN_DISPLAY;
    left_right_margins_mode = FIXED;
    top_margin = 1;
    bottom_margin = lines_per_page;                 /* NVM */
    left_margin = 1;
    right_margin = columns_per_page;                /* NVM */
    current_rendition.bold       = FALSE;
    current_rendition.blink      = FALSE;
    current_rendition.underscore = FALSE;
    current_rendition.reverse    = FALSE;
    current_attribute.selective_erase = FALSE;
    save_cursor();  /* initialize cursor save buffer */
    cursor_save_buffer.position.line = 1;
    cursor_save_buffer.position.column = 1;
    cursor_save_buffer.position.page = 1;

    /* keyboard processing */
    keyboard_action_mode = UNLOCKED;
    cursor_key_mode = CURSOR;                       /* NVM */
    keypad_mode = NUMERIC;
    keyboard_encoding_mode = CHARACTER;

    };
};      /* end procedure soft_terminal_reset() */
```

Known Deviations:

VT200s, VT300s, and the VT420 clear auto wrap (ignore NVM). DECterm V1 does not change the current setting of auto wrap. This was necessary for ULTRIX systems that depend on auto wrap being enabled when a new terminal is initialized.

Levels: 1, 2, 3, 4

Purpose: To cause the terminal to report the current functional status.

Request Format:

```
CSI          Ps          n
9/11         Ps          6/14
```

Report Format:

```
CSI          Ps          n
9/11         Ps          6/14
```

Description: The DSR request, with selective parameter, causes the terminal to respond with a corresponding DSR control using a parameter from the table to indicate its current functional status.

	Levels 1 - 4
5	request terminal status (command from host)
0	device ready
3	device not functional
	Level 4
?75	report status of data integrity flag (command from host)
?70	ok, no comm errors or power-up since last report
?71	error, a comm error has been detected since last report
?73	power-up, data integrity flag has not been reported since last power-up or RIS
	Multiple Sessions Extension
?85	report virtual terminal configuration (command from host)
?81	virtual terminals available through TD/SMP
?83	not configured for virtual terminals
?87	virtual terminals operating via physical lines

NOTES

1. **Some Device Status Reports (DSRs) are part of terminal management, while others pertain to presentation state or input processing. See *DEC STD 070-5 Video Systems Reference Manual - Character Cell Display* and *DEC STD 070-7 Video Systems Reference Manual - Printer Port Extension* for additional DSRs. The report status of the data integrity flag DSR is intended for application use.**
2. **On hardcopy devices DSR has often been processed asynchronously ahead of other data in the input buffer (guideline).**

3. **TD/SMP is described in the "Terminal Device/Session Management Protocol" Version 1 Architecture Specification, 13-Feb-1986.**

State Affected:

```
error_detected_flag    /* if DSR with Ps = ?75 is requested */
power_up_detected_flag
```

Algorithm:

```
void report_device_status(request, private)
    int request;
    boolean_t private;
{
switch (request)
    {
    case 5:
     if (device_status == READY)
      {
      send_char(0x9B, host_port);
      send_string("0n", host_port);
      }
     else
      {
      send_char(0x9B, host_port);
      send_string("3n", host_port);
      }
     break;
    case 75:
     if (private)
     {
     if (!error_detected_flag && !power_up_detected_flag)
      {
      send_char(0x9B, host_port);
      send_string("?70n", host_port);
      }
     else if (power_up_detected_flag)
      {
      send_char(0x9B, host_port);
      send_string("?73n", host_port);
      power_up_detected_flag = FALSE;
      }
     else
      {
      send_char(0x9B, host_port);
      send_string("?71n", host_port);
      error_detected_flag = FALSE;
      };
     };
    break;
     case 85:
 if (private)
      {
      send_char(0x9B, host_port);
      if (vt_configuration == SINGLE_SESSION)
      send_string("?83n", host_port);
     else
      if (vt_configuration == SESSIONS_ON_COMM)
      send_string("?81n", host_port);
     else
      send_string("?87n", host_port);
      };
    break;
    }
};    /* end procedure report_device_status() */
```

Known Deviations:

The VT330 and VT340 terminals do not implement the Virtual Terminal Configuration Device Status Reports.

The VT420 can report "?80 virtual terminals operating through TD/SMP", and include a second parameter that indicates the number of sessions available (when the first parameter is ?80 or ?81). It was later agreed that protocol state information should be obtained through the protocol handler, so this has not been standardized.

6 CHANGE HISTORY

6.1 REVISION 0.0 TO 0.1

The following changes were made to accommodate integration of this specification with the Video Systems Reference Manual:

1. The name of this section was changed from "Terminal Management Level 1" to "Terminal Management".
2. All references to Level 1 operation only were removed, and explicit references to Level 1 and Level 2 operation were added for all functions.
3. Some of the state names were changed to be consistent with the rest of the Video SRM.
4. The sections on Warning Bell and Text Cursor Enable Mode were removed to the section "Character Cell Display".
5. All sections on printer functions were removed to the section "Printer Port Extension".
6. The section on Hard Copy (DECHCP) was removed to the section "Sixel Graphics Extension".
7. Sections from the "Character Cell Display" section on Reset to Initial State and Device Status Report were added to this section.
8. New sections on Soft Terminal Reset and Answerback were also added to this section.
9. The algorithm for Reset to Initial State was corrected.
10. The coding of the DECID sequence was corrected.
11. Coding of other controls (most of which were removed to other section) were also corrected.
12. The coding of the Device Not Ready status response to Device Status Report was corrected.

6.2 REVISION 0.1 TO AX10

1. Removed statement that the revision level code in the secondary Device Attributes response would be zero for unreleased products.
2. Removed Answerback to an appendix. Answerback messages will not be required for conforming hardware.
3. Added Selectively Erasable Character extension to the primary Device Attributes response with parameter 6.
4. Made code changes in the primary DA algorithm. Fixed a bug in the Select Conformance Level algorithm.
5. Added note that DRCS and UDK state is cleared on Reset to Initial State in Level 2 terminals.
6. Made changes in the tables of affected state in the Soft Terminal Reset function.
7. Removed DECTST (Test) from the SRM. It will not be a required function for conforming hardware. However, Device Status Report will be required.
8. Removed Control Representation Mode from RIS.

9. Clarified that the device code returned by Primary Device Attributes will not change when the operating level changes.
10. Made New Line Mode unaffected by Soft Terminal Reset.
11. Added a note to Select Conformance Level to indicate that it is a required function in Level 2 only, but is active in Level 1 operation of Level 2 devices. Removed deviation notes on VT100 family for this function.
12. Removed Reset To Initial State as a required function, and placed it in Appendix D *Documented Exceptions*. It is not intended for use by conforming software. The RIS algorithm is now included in this section under the heading "Device Initialization".
13. Made Primary Device Attributes, Select Conformance Level, and Secondary Device Attributes a "temporal" extension to Level 1 (required in all future Level 1 implementations).
14. Added initialization of the character set table of designating escape sequences to the Soft Terminal Reset control.
15. Added setting of the Cursor Save Buffer values to the level defaults in Soft Terminal Reset.

6.3 Rev AX10 to AX11

Added "9 - NRC" to the list of Level 2 extensions.

6.4 Rev AX11 to AX12

GENERAL:

1. Removed Rev AX11 change bars. Added change bars to any change that could affect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.
2. Updated Reference Standards section to include latest standards and revisions.
3. Added paragraph to "Relationship to TIA" subsection explaining distinction between functions intended for either application use, terminal management use, or communications use. Included entity functions are intended to be used by in individual function descriptions.
4. Added note to definition of Device Status Report in terminology section to distinguish DSRs used for Terminal Management versus others. Some Device Status Reports are properly part of Terminal Management, while others pertain to presentation state or input processing.
5. Added definitions for "Factory Default State", "Power-Up State", and "Tertiary Device Attributes" in terminology section.
6. Recoded all algorithms in VAX–11 C to match new Code Extension section.
7. Updated algorithms and state information throughout to include Level 3 and Level 4 Character Cell Display Service Class.

STATE DESCRIPTIONS:

8. Added note explaining the status of the algorithms at the beginning of the State Descriptions subsection. Algorithms should be viewed the same as other text.
9. Added "terminal unit ID" to device identification information available at Level 4.
10. Updated list of extensions.

INITIALIZATION:

11. Updated list of features that may be initialized from NVM to include VT300 and VT400 capabilities. Removed "keypad_mode" to reflect actual implementation in the VT100/VT200/VT300. Keypad mode is not stored in NVM.
12. Combined Control String extensions as a single enumerated type since it is not possible to be in more than one control string at a time.
13. Added character_set_mode for NRCS extension.
14. Included parser initialization from Code Extension section.
15. Added data integrity flags as defined for VT400.
16. Added virtual terminal configuration (vt_configuration) as used in VT420 to control multiple sessions.
17. Added page number, lines_per_page, columns_per_page, and cursor coupling to support Windowing Extension as implemented by VT330,VT340, and VT420.
18. Added state information for horizontal scrolling: left_margin, right_margin, left_right_margin_mode.
19. Added area extent mode (stream versus rectangular) for rectangular area operations.
20. Added state for selecting status display type and active display.
21. Expanded keyboard state to include: keyboard_usage_mode, send_receive_mode, backarrow_key_mode, compose_key_mode, alt_key_mode, keyboard_encoding_mode (character versus position), and local controls for F1-F5.
22. Added printer_to_host per VT240, and printer_assignment for multiple sessions.
23. Added code to initialize set-up features that are not setable from the host but that affect the description of other functions.

CONTROL FUNCTIONS:

24. Clarified that higher conformance levels do not have to be strict supersets of lower levels, and that conforming software must select a known conformance level.
25. Clarified in the text that the primary DA reports extensions supported at the current operating level. If higher conformance levels subsume extensions previously reported at lower levels, these extensions must still be reported at the lower level to ensure existing software can recognize them.
26. Added concept of "Implicit Extensions" so software can recognize features defined at a higher conformance level which are not disabled at lower levels. Listed implicit extensions created by Level 2 and Level 3. Removed note that DECSCL was the only exception to rule that features defined at higher levels must be disabled at lower levels.

Implicit extensions ensure software can still recognize which features are present, and corresponds to actual VT200 and VT300 implementations.

27. Noted that DECID (ESC Z) is not recognized at Level 3 or higher.
28. Added description of Secure Reset function.
29. Updated DECSTR (Soft Terminal Reset) to reflect newer implementations as follows: Bottom margin is set to number of lines on page; left and right margins are reset; auto wrap mode is recalled from NVR; current attribute is set to not selective erasable; 8-bit multinational characters are selected; the main display is selected (status display is exited); the User Preference Supplemental Set (UPSS) is recalled from NVM; keyboard encoding mode is reset to character (VT420); Added macro definitions to list of state not affected (VT420).
30. Added Device Status Reports for data integrity and Virtual Terminal State (VT420). Noted that other DSRs are not part of terminal management.
31. Added note that DSR has been processed asynchronously ahead of other data in the input buffer on hardcopy devices (guideline).

APPENDIX A REFERENCED DOCUMENTS

A.1 EL-Class Digital Documents

EL-Class Number	Document Title
EL-00070-05	*DEC STD 070-5 Video Systems Reference Manual - Character Cell Display*
EL-00070-07	*DEC STD 070-7 Video Systems Reference Manual - Printer Port Extension*
EL-00070-0A	*Video Systems Reference Manual - VT52 Emulation*
EL-00070-0D	*Video Systems Reference Manual - Documented Exceptions*
EL-00138-00	*DEC STD 138-0 Registry of Control Functions for Character Imaging Devices*

Use VTX SMC to order copies of EL-class documents from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

APPENDIX B RELATED DOCUMENTS

B.1 EL-Class Digital Documents

EL-Class Number	Document Title
EL-00070-01	*DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria*
EL-00169-00	*DEC STD 169-0 DEC Standard Coded Graphic Character Sets for Hardware and Software*

Use VTX SMC to order copies of EL-class documents from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

B.2 Other Documents

Document Number	Document Title
ANSI X3.4 - 1986	*American National Standard Code for Information Interchange (ASCII Character Set)*
ANSI X3.41 - 1974	*American National Standard Code Extension Techniques for use with the 7-Bit Coded Character Set of the American National Standard Code for Information Interchange*
ANSI X3.64 - 1979	*Additional Controls for use with American National Standard Code for Information Interchange*
DPANS X3.134.1-1985	*8-Bit ASCII Structure and Rules*
DPANS X3.134.2-1985	*7-Bit and 8-Bit ASCII Supplemental Multinational Graphic Character Set*
ISO 2022:1986	*Information Processing / ISO 7-Bit and 8-Bit Coded Character Sets - Code Extension Techniques*
ISO 6429:1988	*Information Processing / Additional Controls for use with Character Imaging Devices*
ISO 646	*7-Bit Coded Character Set for Information Interchange*
ISO 8859-1:1987	*Information Processing / 8-Bit Single-Byte Coded Character Sets - Part 1 : ISO Latin Alphabet Nr 1;*

B.3 Ordering Information

Following are the sources for documents not available from Standards and Methods Control.

- ANSI, IEC, and ISO Documents
  American National Standards Institute (ANSI),
  1430 Broadway, New York, NY 10018
  Telephone (212) 354-3300 x479

INDEX

T

```
+-------------------------------------------------------------------+
|                        READER COMMENTS                            |
|                                                                   |
| Your comments and suggestions will help Standards and Methods     |
| Control improve their services and documents.                     |
+-------------------------------------------------------------------+
```

Did you request this document? _____ If so, did it arrive within a satisfactory period of time? _____ Please comment.

What are your impressions of this document? Consider format, organization, completeness, readability, and illustrations.

FOLD ON THIS LINE

Did you find technical or clerical errors in this document? If so, please specify the page number(s) and the error(s).

Are the instructions for the update package clear? _____
Was an index available? _____ If not, is one needed? _____
Do you have other suggestions for improving this document?

The following information is optional:

Name _____ Mailstop _____

Department _____ Node _____

Send your comments to JOKUR::PROJECTS, or fold, staple, and send this page through interoffice mail to:

```
+------------------------------+
|       READERS' COMMENTS      |
| STANDARDS AND METHODS CONTROL|
|           CTS1-2/D4          |
+------------------------------+
```

CHAPTER 5

CHARACTER CELL DISPLAY

VIDEO SYSTEMS REFERENCE MANUAL

Character Cell Display

Document Identifier: A-DS-EL00070-05-0 Rev. AX12, 29-Jun-1990

ABSTRACT: This section includes the definition of all state information required for the Character Cell Display service class, as well as all interface operations which may be performed to affect this state.

APPLICABILITY: Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria".

STATUS: FOR REVIEW ONLY

+--+
| The material contained within this document is assumed to |
| define mandatory standards unless it is clearly marked as: |
| (a) not mandatory; or (b) guidelines. Material which is |
| marked as "not mandatory" is considered to be of potential |
| benefit to the corporation and should be followed unless there |
| are good reasons for non-compliance. "Guidelines" define |
| approaches and techniques which are considered to be good |
| practice, but should not be considered as requirements. |
+--+

TITLE: VIDEO SYSTEMS - Character Cell Display

DOCUMENT IDENTIFIER: A-DS-EL00070-05-0 Rev. AX12, 29-Jun-1990

REVISION HISTORY:

Original Draft	15-Jul-1982
Revision 0.4	30-Sep-1982
Revision 0.5	25-Dec-1982
Revision AX01	28-Feb-1983
Revision AX10	15-May-1983
Revision AX11	18-Mar-1985

FILES: User Documentation EL070S5.mem

Internal Documentation EL070S5.rno
EL070S5.rnt
EL070S5.rnx

Document Management Group: Terminal Interface Architecture (STI)
Responsible Department: VIPS Terminals Architecture
Responsible Person: Peter Sichel

APPROVAL: This document prepared by the VIPS (Video Image & Printing Systems) Group has been reviewed and recommended for approval by the general review group for its category for use throughout Digital.

*** Status: For Review Only ***
_____ Peter Sichel,
VIPS Terminals Architecture

*** Status: For Review Only ***
_____ Eric Williams,
Standards Process Manager

Direct requests for further information to Peter Sichel, DSG1-2/C7, DTN 235-8374, HANNAH::TERMARCH

Copies of this document can be ordered from Standards and Methods Control, CTS1-2/D4, DTN 287-3724, JOKUR::SMC

CONTENTS

5.1 INTRODUCTION

5.1.1 Purpose

This section defines the interface to a Character Cell Display device. A Character Cell Display device is a device capable of character imaging output. In addition to the physical presentation of graphic characters, it must provide certain control functions for defining the format and rendition of the information presented.

The Character Cell service class consists of operations that image or format characters that are addressed on the imaging surface using a character cell addressing capability. All Character Cell operations are either graphic (printing) characters or control functions (control characters, escape sequences, control sequences, or control strings) according to the structure of ANSI X3.41, Code Extension Techniques for use with ASCII, and ANSI X3.64, Additional Control Functions for use with ASCII.

Character Cell Display devices are distinguished from Character Cell Printing devices by their ability to perform certain dynamic operations, such as inserting characters within text, erasing previously displayed characters, and blinking.

The Character Cell Display service class consists of operations that:

1. image characters on a two-dimensional display surface that represent text or picture drawing symbols,
2. alter their position after receipt
3. remove them from the display, and
4. cause them to be re-transmitted.

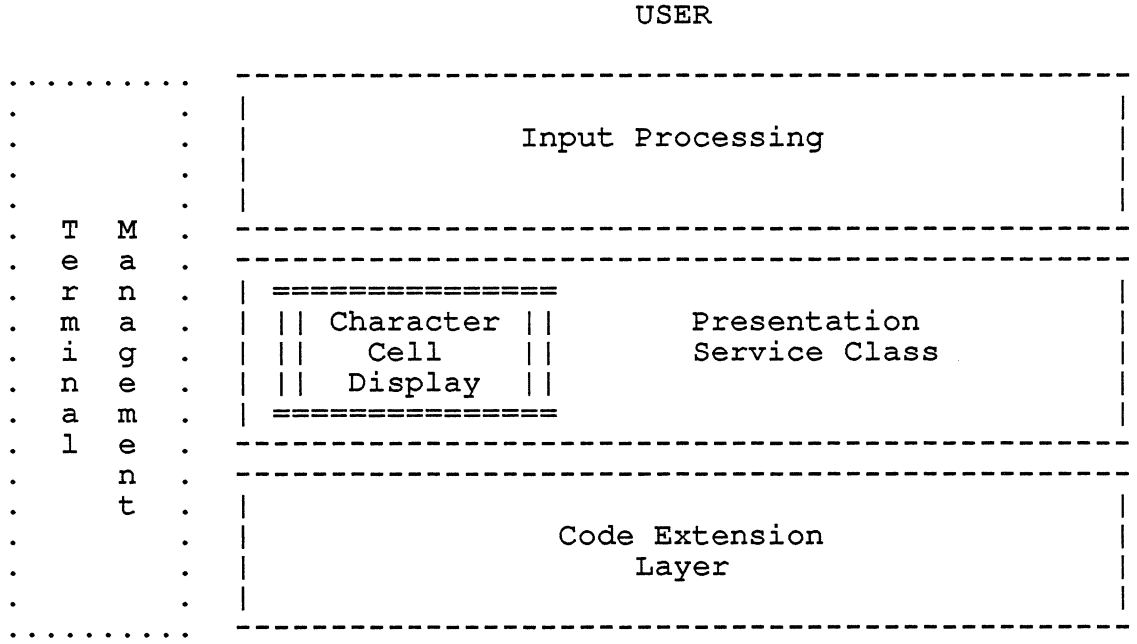
The intent of the Character Cell Display service class is to be implementable using a character generator or a minimum of a single plane image memory.

5.1.2 Scope

This section applies to all DEC products which implement these interfaces and are intended to be certified as conforming to the Terminal Interface Architecture. It also applies to software which is intended to use these interfaces in a conforming manner.

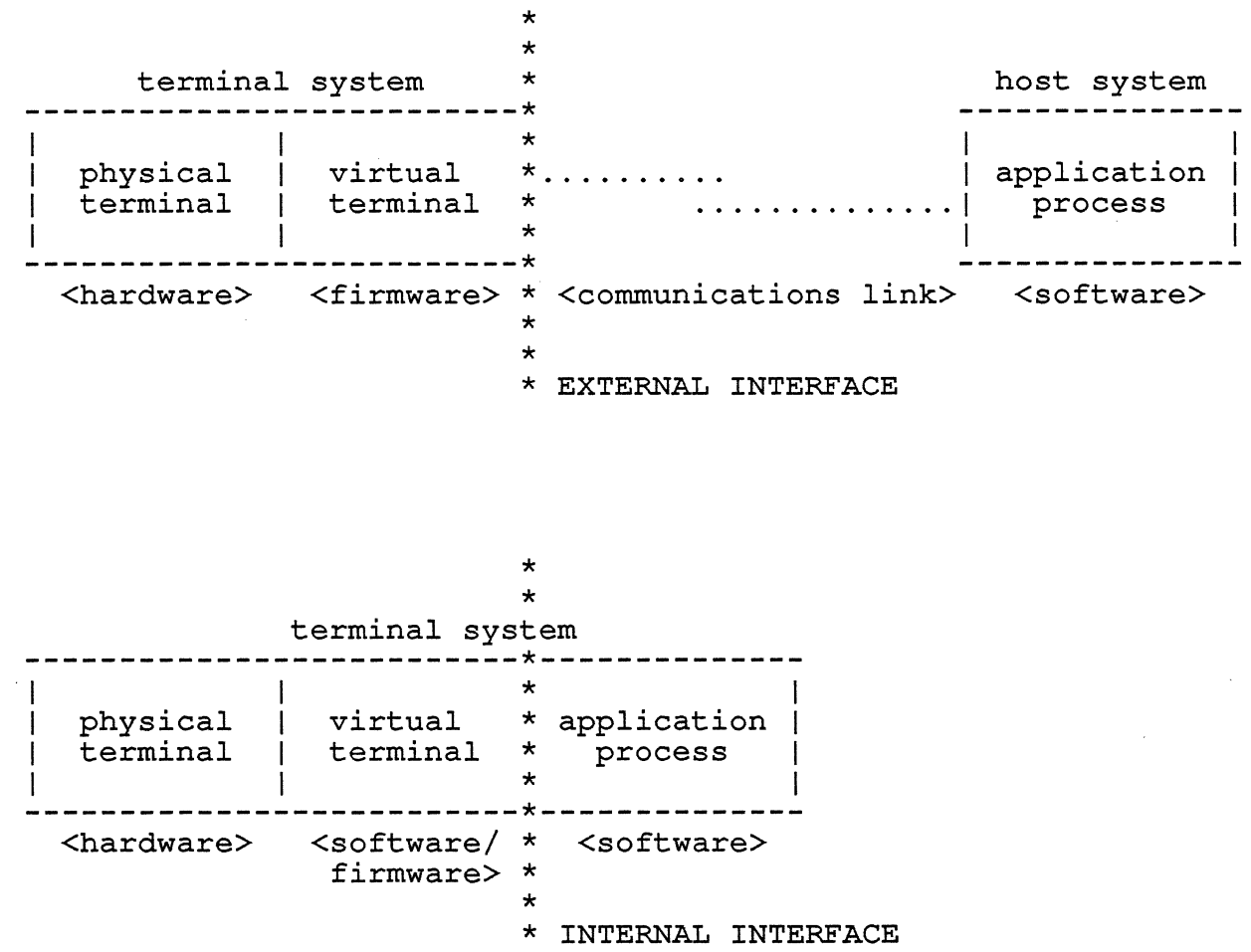
5.1.3 Relationship To TIA

The Character Cell interface forms a service class of the Presentation Service Class layer of the Terminal Interface Architecture. It is independent of the other layers of the architecture, and has no explicit relationship to other service classes within the Presentation Service Class layer.



Structuring of the Terminal Interface Architecture

The interfaces defined within this specification apply to both internal and external product interfaces. External interfaces are interfaces between a terminal, personal computer or workstation and a remote system. Internal interfaces are interfaces between a terminal sub-system and software processes running within a terminal, personal computer or workstation.



External and Internal Terminal Interfaces

5.1.4 Organization

To simplify understanding the Character Cell Display service class, this section is divided into a number of largely independent subsections which describe functionally related parts of the ANSI host interface. These subsections are:

- Display Coordinate System and Addressing
- Window Management
- Visual Attributes and Renditions
- Audible Indicator
- Graphic Character Sets
- Summary of Control Character Processing
- Mode States
- Editing Functions
- OLTP Features
- Saving and Restoring Terminal State

Other sections of the VSRM also describe parts of the ANSI host interface.

5.2 TERMINOLOGY

Active Position - (1) The character position in a visual display that is to image the graphic symbol representing the next graphic or control character for which a graphic representation is required (2) The character position in a logical display which contains the character code, graphic rendition, and character set of a graphic or control character for which a graphic representation is required.

Advance - To move the active position in the direction of increasing horizontal character position in a visual or logical display.

Application - A hardware or software implementation of a process or a device.

Application Program - A program that runs under control of an operating system.

Audible Indicator - A warning "bell", or similar indicator suitable to signal the operator's attention, primarily to warn of certain error conditions. There are four functional categories of bells in conforming terminals: Right Margin Bell, Warning Bell, Required Error Bells, and Selectable Error Bells.

Backward - In the direction of decreasing horizontal character position in a visual or logical display.

Blind Interchange - Information interchange in which no prior agreements between sender and recipient are necessary in order to achieve successful interpretation of the information, except agreed-upon standards.

C0 Set - A set of 32 control characters allocated to columns 0 and 1 of a code table.

C1 Set - A set of 32 control characters allocated to columns 8 and 9 of an 8-bit code table, or represented as ESCape Fe sequences in 7-bit environment with identical meaning.

Character Attribute - An attribute value which may be applied to a character position, which affects its subsequent processing, as in editing or retransmission.

Character Imaging Device - A device that gives a visual representation of data in the form of graphic symbols using any technology, such as a on a cathode ray tube. In this specification the term "character imaging device" does not apply to hard copy devices (printers) which are not capable of dynamic editing and reformatting of the displayed information.

Character Position - (1) The portion of a visual display that images or is capable of imaging a graphic symbol. (2) An addressable element in a logical display containing sufficient information to render a graphic symbol, including the character code, visual attributes, logical attributes, and character set designation.

Character Rendition - An attribute value which may be applied to a character position, which affects its visual representation (see Graphic Rendition).

Control - A control character, an escape sequence, or a control sequence that performs a control function.

| Control Sequence Introducer (CSI) - The C1 control (represented by
| a single character in 8-bits, or a two character ESC Fe sequence
| in 7-bits) which initiates a control sequence. CSI is a prefix
| affecting the interpretation of a limited number of contiguous bit
| combinations.

Cursor, Cursor Symbol - A visual (blinking or non-blinking)

symbol, usually a rectangle, diamond or underline that indicates the Active Position on the display.

Cursor Control - An editing function that moves the active position.

Default - A function-dependent value that is assumed when no explicit value, or a value of 0, is specified.

Delete - To remove displayed symbols and close up adjacent graphic symbols to fill the gap.

Designate - To identify a set of characters that are to be represented in a prescribed manner, in some cases immediately and in others on the occurrence of a further control function.

Display - The area for visual presentation of data on cathode ray tube and similar character imaging devices. In this specification the word "display" will always be qualified by the terms "physical" (or "visual") and "logical", to differentiate between the actual presentation surface and memory buffers containing information to be presented.

Editor Function - A control that affects the layout or positioning of previously entered or received information in a character imaging device and is intended to be interpreted without remaining in the data stream. See Format Effector.

Empty Character - A character position in the logical display in which no character code appears, and thus no character is rendered in the visual display for this position. In an empty character position the character rendition, character attribute, and character set values should be ignored.

Enter - To input information manually into a character imaging device or to read information from an auxiliary device into a character imaging device.

Erase - To remove displayed graphic symbols without closing up adjacent symbols to fill the gap.

Fixed Space - A character that normally has no graphic representation, occupies a character position in a visual display, and is usually encoded as bit combination 2/0.

Following - Lines or character positions in the visual display with larger numbered lines or larger numbered character positions than that of the active position.

Font - A complete assortment of displayable graphic symbols in one size or style.

Format Effector - A control that effects the layout or positioning

of information in a character imaging device and may remain in the data stream subsequent to interpretation and processing. See Editor Function.

Forward - In the direction of increasing horizontal character position in a visual display.

Graphic Character - A character, other than a control character, that has a visual representation normally handwritten, printed, or displayed.

Graphic Rendition - A visual style of displaying a set of graphic symbols.

Graphic Symbol - A visual representation of a graphic character or a control for which a graphic representation is required.

Invoke - To cause a designated set of characters to be represented by the prescribed bit combinations whenever those bit combinations occur until an appropriate code extension function occurs.

Line - A set of adjacent character positions in a visual display that have the same vertical position.

| Logical Display - the lines and columns available for storing and presenting graphic characters as seen by the application or host computer. A Logical Display may be organized as a single rectangular array of lines and columns (one page), or divided into a number of identically sized pages, each of which is a rectangular array of lines and columns.

| Margin - A line which marks the upper or lower boundary of the scrolling region (vertical scroll margin). A column which marks the left or right boundary of the scrolling region (horizontal scroll margin).

Mode - A state of a device, or other sender or recipient, that affects the interpretation of received information, the operation of the sender or recipient, or the format of the transmitted information.

Next - See Following.

Operating System - Software that controls the execution of computer programs and may provide scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management, and related services.

| Page - An addressable area of a Logical Display organized as a rectangular array of lines and columns. The number of lines and columns defines the Page Size.

Preceding - Lines or character positions in a visual display with

smaller numbered lines or smaller numbered character positions than that of the active position.

Private Use - A technique of encoding or representing information in a prescribed, but nonstandard, way.

Received Data Stream - The stream of bit combinations received by a character imaging device for purposes of information interchange.

Represent - (1) To use a prescribed bit combination with the meaning of a character in a set of characters that has been designated and invoked. (2) To use an escape sequence with the meaning of an additional control character.

Scroll - An action whereby all of the graphic symbols of a visual display are moved in a specified direction.

String Delimiter - A control that begins or ends a string of characters in a data stream.

Tabulation - A technique of identifying character positions in a visual display for the purpose of arranging information systematically.

Tabulation Stop - An indication that a character position is to be used for tabulation.

Transmit - To send data as a data stream for purposes of information interchange.

Transmitted Data Stream - The stream of bit combinations sent by a character imaging device when it is induced to transmit for purposes of information interchange.

5.2.1 Code Extension Terms

This chapter also uses the following terms which are defined in the "Code Extension Layer" section of the VSRM (DEC STD 70-3):

- Bit Combination
- Character
- Control Character
- Control Function
- Control Sequence
- Control String
- Escape Character (ESC)
- Escape Sequence
- Final Character
- Intermediate Character
- Numeric Parameter

| Parameter
| Parameter String
| Selective Parameter

5.3 CHARACTER CELL DISPLAY STATE DECLARATIONS

```
/**************  module char_cell_types.h  *************/
/*
 * type declarations for character cell display state
 * PAS 28-Jul-1989
 */

typedef enum { JUMP, SLOW } scrolling_mode_t;
typedef enum { NORMAL_SCREEN, REVERSE_SCREEN } screen_mode_t;
typedef enum { WRAP_OFF, WRAP_ON } auto_wrap_mode_t;
typedef enum { TEXT_CURSOR_OFF, TEXT_CURSOR_ON }
    text_cursor_enable_mode_t;
typedef enum { INSERT, REPLACE } insert_replace_mode_t;
typedef enum { NEW_LINE_OFF, NEW_LINE_ON } new_line_mode_t;
typedef short int line_t;
typedef short int column_t;
typedef short int page_t;
typedef struct
    {          /* character_position_t */
    line_t line;
    column_t column;
    page_t page;
    } character_position_t;
typedef enum { TRUE, FALSE } boolean_t;
typedef struct
    {          /* character_rendition_t */
    boolean_t bold;
    boolean_t blink;
    boolean_t underscore;
    boolean_t reverse;
    } character_rendition_t;
typedef struct
    {         /* character_attribute_t */
    boolean_t selective_erase;
    } character_attribute_t;
typedef enum
    {         /* line_rendition_t */
    DOUBLE_HEIGHT_TOP,
    DOUBLE_HEIGHT_BOTTOM,
    SINGLE_WIDTH,
    DOUBLE_WIDTH
    } line_rendition_t;
typedef struct
    {        /* character_t */
    short int code;
    character_rendition_t rendition;
    character_attribute_t attribute;
    graphic_character_set_t character_set;
    } character_t;
typedef struct
    {        /* char_set_designator_t */
    graphic_character_set_t name;
```

```
    short int first_intermediate;     /* 0-47, 0=no intermediate */
    short int second_intermediate;    /* 0-47 */
    short int final;                  /* 0 indicates null entry */
    } char_set_designator_t;
typedef enum { ABSOLUTE, DISPLACED } origin_mode_t;
typedef struct
    {      /* save_buffer_t */
    character_position_t position;
    character_rendition_t rendition;
    origin_mode_t origin_mode;
    graphic_character_set_t left;
    graphic_character_set_t right;
    graphic_character_set_t g0;
    graphic_character_set_t g1;
    graphic_character_set_t g2;
    graphic_character_set_t g3;
    character_attribute_t attribute;
    } save_buffer_t;
typedef enum { EIGHTY, ONE_THIRTY_TWO } column_mode_t;
typedef enum { UNCOUPLED, COUPLED } cursor_coupling_mode_t;
typedef enum { FIXED, SETABLE } left_right_margins_mode_t;
typedef enum { STREAM, RECTANGULAR } area_extent_t;
typedef short int lines_per_page_t;     /* 24, 25, 36, 72, 144 */
typedef short int columns_per_page_t;   /* 80, 132 */
typedef short int number_of_pages_t;    /* 1, 2, 3, 4, 5, 6 */
/************ end of char_cell_types.h  ****************/



/**************  module char_cell_extern.h  *************/
/*
 * external storage declarations for character cell display state
 * PAS 28-Jul-1989
 */

#define MAX_NUM_LINES 73
#define MAX_NUM_COLUMNS 133
#define EMPTY_CHARACTER 0
#define MAX_NUM_CHAR_SETS 8
/* character set table entries */
#define TABLE_ASCII 0
#define TABLE_LINE_DRAWING 1
#define TABLE_DEC_SUPP 2
#define TABLE_NRCS 3
#define TABLE_LATIN1 4
#define TABLE_UPSS 5
#define TABLE_TCS 6
#define TABLE_DRCS 7
/* Note DRCS must be last in search order
   to replace others if identical */

ext character_t
```

```
| display[MAX_NUM_LINES][MAX_NUM_COLUMNS];
| ext line_rendition_t        line_rendition[MAX_NUM_LINES];
| ext character_position_t    active_position;
| ext line_t                  top_margin;
| ext line_t                  bottom_margin;
| ext column_t                left_margin;    /* Horiz scroll ext */
| ext column_t                right_margin;   /* Horiz scroll ext */
| ext boolean_t               last_column_flag;
| ext active_display_t        active_display; /* L3 or Status Dsply*/
| ext status_display_t        status_display; /* L3 or Status Dsply*/
| ext char_set_designator_t   char_set_table[MAX_NUM_CHAR_SETS];
| ext save_buffer_t           cursor_save_buffer;
| ext character_rendition_t   current_rendition;
| ext character_attribute_t   current_attribute; /* Selective Erase*/
| ext boolean_t               horizontal_tab_stops[MAX_NUM_COLUMNS];
| ext column_mode_t           column_mode;      /* 132 Column Ext */
| ext lines_per_page_t        lines_per_page;   /* Windowing */
| ext columns_per_page_t      columns_per_page; /* Windowing */
| ext number_of_pages_t       number_of_pages;  /* Windowing */
| ext scrolling_mode_t        scrolling_mode;
| ext screen_mode_t           screen_mode;
| ext origin_mode_t           origin_mode;
| ext auto_wrap_mode_t        auto_wrap_mode;
| ext text_cursor_enable_mode_t  text_cursor_enable_mode;
| ext insert_replace_mode_t   insert_replacement_mode;
| ext new_line_mode_t         new_line_mode;
| ext cursor_coupling_mode_t
|         horizontal_cursor_coupling,      /* Windowing */
|         vertical_cursor_coupling,        /* Windowing */
|         page_cursor_coupling;            /* Windowing */
| ext left_right_margins_mode_t
|         left_right_margins_mode; /* Horiz Scroll */
| ext area_extent_t
|         attribute_change_extent; /* L4 or rect editing */
| /********** end of char_cell_extern.h  ************/
|
| /* function declarations */ short int end_of_line(); void
| scroll_up(); void scroll_down(); void scroll_left(); void
| scroll_right(); void insert_or_replace_character(); void
| selective_erase_in_line(); void selective_erase_in_display();
```

5.3.1 Notes On Character Cell Display State

a. An empty character is a character position in the Logical Display in which no character code appears, and thus no character is rendered in the visual display for this position. In this implementation this is handled by a null entry in the logical display structure, which should be rendered as a space with normal rendition in the visual display. In other implementations this may be

handled in a manner appropriate to the display list structure, but the affect on the visual display should be the same.

In an empty character position the character rendition, character attribute, and character set values should be ignored.

b. The number of character sets available depends on the conformance level and architecture extensions present. See "Character Set Repertoire" in this chapter (DEC STD 70-5).

See the "Terminal Management" Section (DEC STD 70-4) for a listing of all terminal state, and initialization procedures.

5.4 DISPLAY COORDINATE SYSTEM AND ADDRESSING

5.4.1 Logical Display

A conforming device shall provide the capability of storing and presenting single-width graphic characters in a rectangular array of lines and columns called a Logical Display Page. The number of lines and columns shall define the size of the Logical Display Page, or Page Size. The entire Logical Display may consist of a single page, or a number of pages of identical size.

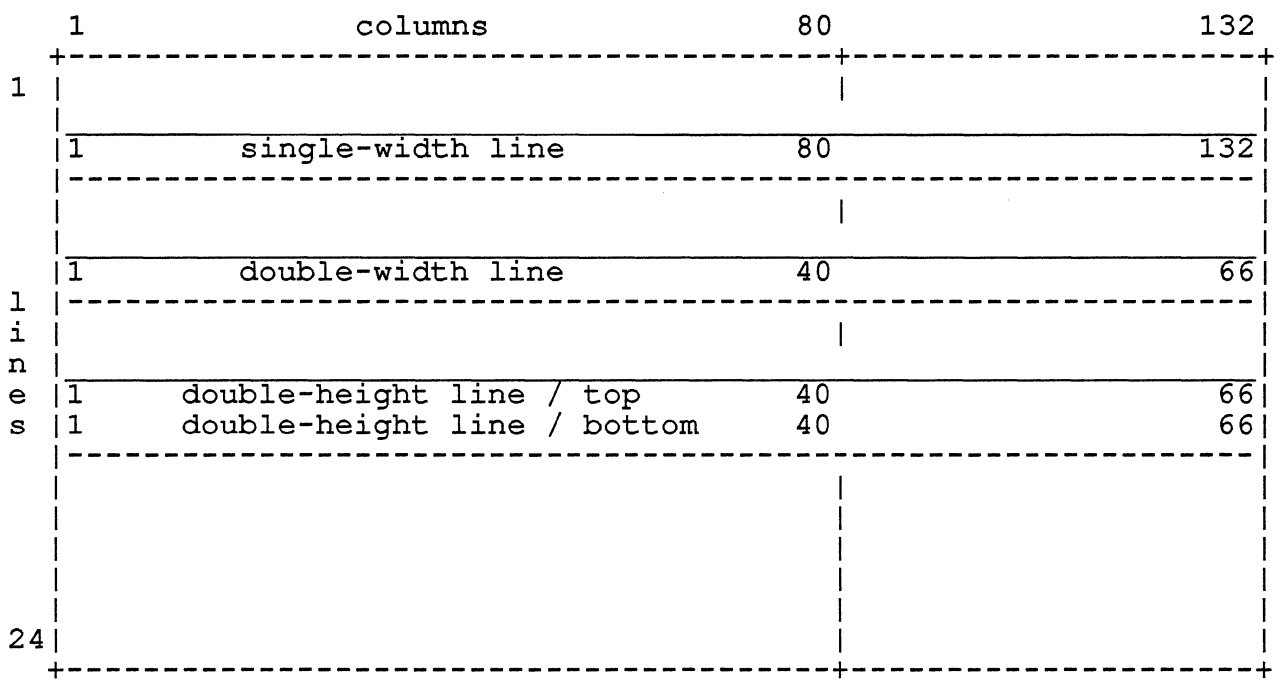
All conforming devices shall support a single Logical Display Page of 24 lines by 80 columns, extendable to 24 lines by 132 columns when the "132 Column Mode" extension is present (DECCOLM, see "Page Size and Arrangement").

Level 4 devices, and Level 3 devices with the "Windowing Extension" shall support commands to change the Logical Display Page Size (DECSLPP and DECSCPP, see "Page Size and Arrangement"). The actual Page Sizes supported shall be clearly specified in the product documentation, and means shall be provided for software to interrogate the size of the Logical Display Page. The Page Size and number of pages in the Logical Display may also be configurable by the user under local control.

Each character position in the Logical Display will consist of the following information: a character code, a character rendition, character attributes, and a character set.

The first position in a logical display page shall be at line 1, column 1. Frequently this will correspond to the upper left corner of the physical display with line numbers increasing downward, and column numbers increasing to the right, but this need not be true.

The graphic representation of each character on the physical display surface will occupy a rectangular cell whose lower left hand corner is at the logical screen coordinate corresponding to the position of the character in the the array.



Display Structure and Addressing

5.4.2 Active Position And Cursor

A conforming device shall maintain the line and column coordinates of a single character position, known as the "Active Position", which serves as a reference point for character insertion and replacement as well as various control functions. The Active Position indicates the character position at which the next operation is to begin.

The Active Position does not necessarily correspond to the visual indicator in the physical display known as the "Cursor Symbol". However, in this specification the terms "Active Position" and "Cursor" are used interchangeably in order not to conflict with previous usage.

SET/RESET TEXT CURSOR ENABLE MODE DECTCEM

Levels: 2-4

Purpose: Select whether the text cursor symbol in the display has a visual representation.

Set Format: CSI ? 25 h
Reset Format: CSI ? 25 l

Description: A conforming device shall provide a means of enabling or disabling the visible cursor symbol. This mode shall apply only to the text cursor symbol implemented by the Character Cell Display service class. In the set state (Text Cursor On), which is the default, the text cursor symbol will be displayed. In the reset state (Text Cursor Off), the text cursor symbol will not be displayed.

When a request is received to turn off the cursor, it will be done immediately (if not already off). If a request is received to enable the cursor, and the cursor is not already on, the cursor will remain off for a small "initial" cycle, after which it will become visible and begin its normal blinking cycle. This makes the cursor blink with a steady duty cycle regardless of how fast it is turned on and off.

Notes:

1. There is some concern that a program might inadvertently leave the text cursor turned off. It is suggested that entering SETUP mode will always turn on the text cursor while in SETUP mode, but that normally it be restored to its previous state upon exiting SETUP mode. Also, a means should be provided in SETUP mode to force the text cursor to be re-enabled.

State Affected:

text_cursor_enable_mode

Algorithm:

```
void set_text_cursor_enable_mode()
{
if (conformance_level >= LEVEL_2)
    text_cursor_enable_mode = TEXT_CURSOR_ON;
}


void reset_text_cursor_enable_mode()
{
if (conformance_level >= LEVEL_2)
    text_cursor_enable_mode = TEXT_CURSOR_OFF;
}
```

Known Deviations: None

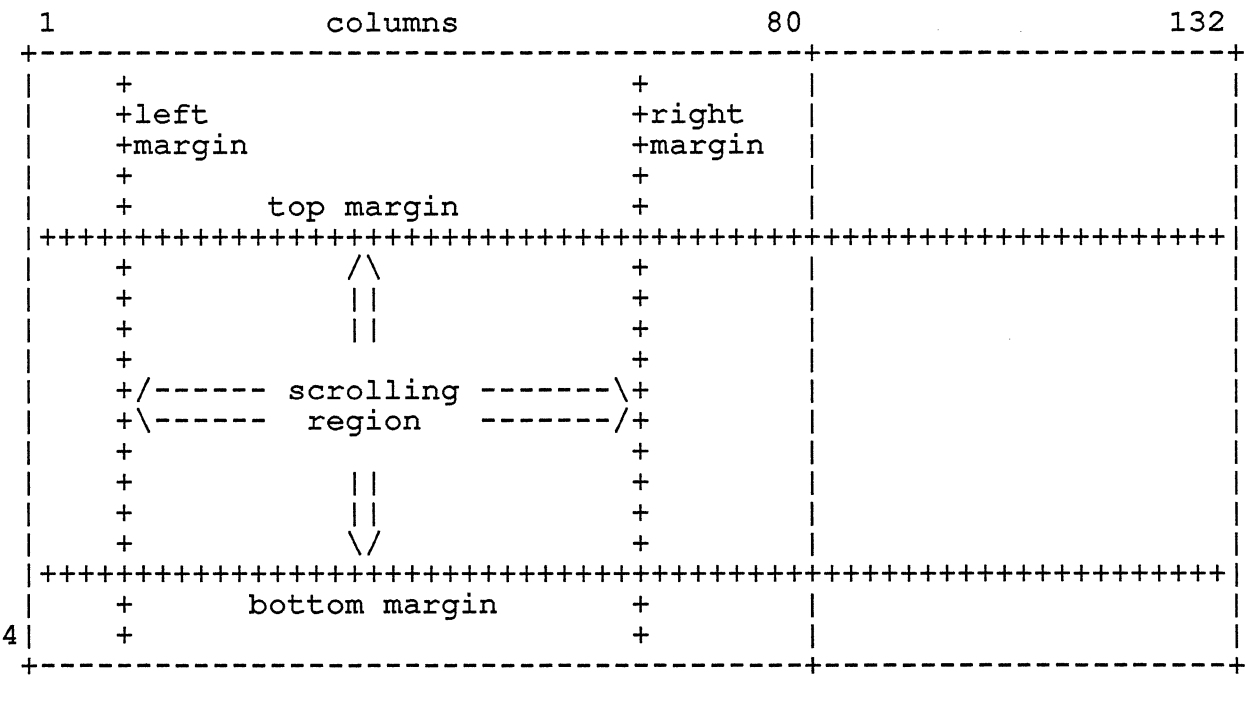
5.4.3 Margins And Scrolling

A conforming device shall provide the capability of defining top and bottom margins of a scrolling region. The top and bottom margins are two line numbers in a logical display page. The top margin must always have a lower line number than the bottom margin.

Devices with the Horizontal Scrolling Extension shall provide the ability to define left and right margins of a scrolling region. The left and right margins are two column numbers in a logical display page. The left margin must always have a lower column number than the right margin.

Numerous editing and cursor control functions are defined relative to the scrolling margins.

The default scrolling region at power-up or after reset is equal to the size of the logical display page.



Margins and Scrolling Region

A scrolling region is defined as the area bounded by the top, bottom, left, and right margins, which are set by DECSTBM and DECSLRM. All scrolling, both vertical and horizontal, is limited to this area.

Several controls are ignored or behave differently if the active

position is outside certain boundaries of the scrolling region. The following table lists these controls:

	margins T/B	margins L/R
DCH	O	X
ICH	O	X
DL	X	X
IL	X	X
DECDC	X	X
DECIC	X	X
IND	*	O
RI	*	O
DECBI	O	*
DECFI	O	*

X = Is affected.
Does not work outside these margins.

O = Not affected.
Continues to work outside these margins.

* = If active position is inside margins, affect is limited by margins.
If active position is outside margins, control still has an affect, but no scrolling can occur.

The following functions described in this subsection are used to control scrolling:

DECSTBM	Set Top and Bottom Margins
DECSLRM	Set Left and Right Margins
DECLRMM	Left Right Margin Mode
DECOM	Origin Mode
DECSCLM	Scrolling Mode
IND	Index
RI	Reverse Index
DECFI	Forward Index
DECBI	Back Index

SET TOP AND BOTTOM MARGINS DECSTBM

Levels: 1-4

Purpose: Change the top and bottom margin settings for the scrolling region.

Format:

	CSI	Pt	;	Pb	r	default Pt: 1
	9/11	Pt	;	Pb	7/2	default Pb: last line

Description: The DECSTBM control sets the values of the Top and Bottom Margins of the scrolling region. The new settings are determined by the parameter values. The first parameter sets the value of the Top Margin, and the second parameter sets the value of the Bottom Margin. The default settings if either or both parameters are omitted are the boundaries of the Logical Display Page: one (1) for the Top Margin and twenty-four (24) for the Bottom Margin when the Page Size is 24 lines.

Notes:

1. Execution of this control causes the Active Position to be set to the page origin obeying Origin Mode (DECOM): First column of first line if Origin Mode is in the reset (Absolute) state; Top and Left Margins if Origin Mode is in the set (Displaced) state.

2. If the value specified for the Top Margin is equal to or greater than the value specified for the Bottom Margin, this control will be ignored (not executed).

3. If the value specified for the Bottom Margin is greater than the number of lines in the Logical Display Page, this control will be ignored (not executed).

State Affected:

active_position
top_margin
bottom_margin

Algorithm:

```
void set_top_and_bottom_margins(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
/* use defaults for omitted parameters */
if (parv[0] == 0) parv[0] = 1;
if (parv[1] == 0) parv[1] = lines_per_page;

if ((parv[0] < parv[1]) && (parv[1] <= lines_per_page))
  {
  top_margin = parv[0];
  bottom_margin = parv[1];
  if (origin_mode == ABSOLUTE)
    {
    active_position.line = 1;
    active_position.column = 1;
    }
  else
    {
    active_position.line = top_margin;
    active_position.column = left_margin;
    }
  }
}
```

Known Deviations: None

SET LEFT AND RIGHT MARGINS DECSLRM

Levels: 4x (Horizontal Scrolling)

Purpose: Change the left and right margin settings for the scrolling region.

Format:

CSI	Pl	;	Pr	s	default Pl: 1
9/11	Pl	;	Pr	7/3	default Pr: Last Column

Description: The DECSLRM control sets the values of the Left and Right Margins of the scrolling region. The new settings are determined by the parameter values. The first parameter sets the value of the Left Margin, and the second parameter sets the value of the Right Margin. The default settings if either or both parameters are omitted are the boundaries of the Logical Display Page: one (1) for the Left Margin and eighty (80) for the Right Margin when the Page Size is 80 columns wide.

Notes:

1. Execution of this control causes the Active Position to be set to the page origin obeying Origin Mode (DECOM): First column of first line if Origin Mode is in the reset (Absolute) state; Top and Left Margins if Origin Mode is in the set (Displaced) state.

2. If the value specified for the Left Margin is equal to or greater than the value specified for the Right Margin, this control will be ignored (not executed).

3. If the value specified for the Right Margin is greater than the number of columns in the Logical Display Page, this control will be ignored (not executed).

4. This control function is only recognized when DECLRMM (Left Right Margin Mode) is set.

5. Setting the margins to other than their default values (the boundaries of the Logical Display Page) may disable smooth scrolling (DECSCLM).

State Affected:

active_position
left_margin
right_margin

Algorithm:

```
void set_left_and_right_margins(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
if ( (conformance_level == LEVEL_4) &&
     (level_4_extensions[HORIZONTAL_SCROLL] == TRUE) &&
     (left_right_margins_mode == SETABLE) )
  {
  /* use defaults for omitted parameters */
  if (parv[0] == 0) parv[0] = 1;
  if (parv[1] == 0) parv[1] = columns_per_page;

  if ((parv[0] < parv[1]) && (parv[1] <= columns_per_page))
    {
    left_margin = parv[0];
    right_margin = parv[1];
    if (origin_mode == ABSOLUTE)
      {
      active_position.line = 1;
      active_position.column = 1;
      }
    else
      {
      active_position.line = top_margin;
      active_position.column = left_margin;
      }
    }
  }
}
```

Known Deviations: None

SET/RESET LEFT RIGHT MARGIN MODE DECLRMM

Levels: 4x (Horizontal Scrolling)

Purpose: Change the state of Left Right Margin Mode between Not Available (reset) and Available (set).

To set:	CSI ? 69 h	(Available)
To reset:	CSI ? 69 l	(Not Available - default)

Description: DECLRMM is a parameter to the Set Mode and Reset Mode commands. In the reset state, the left and right scrolling margins in every page may not be moved through the DECSLRM command, they are set to be at the extremes of the page. In the set state, DECSLRM commands are recognized, however, no line attributes other than Single Wide may be used.

When DECLRMM is set, all line attributes currently in Page Memory for the Session will be reset to Single Wide, Single High, and DECDWL and DECDHL escape sequences, to change the line attribute to Double Wide or Double Wide and High, will be ignored.

State Affected:

- left_right_margins_mode
- line_rendition[MAX_NUM_LINES]
- left_margin
- right_margin

Algorithm:

```
void set_left_right_margins_mode()
{
int i;
if ( (conformance_level == LEVEL_4) &&
     (level_4_extensions[HORIZONTAL_SCROLL] == TRUE) )
  {
  for (i=0; i<MAX_NUM_LINES; i++)
    {
    line_rendition[i] = SINGLE_WIDTH;
    }
  left_right_margins_mode = SETABLE;
  }
}


void reset_left_right_margins_mode()
{
if ( (conformance_level == LEVEL_4) &&
     (level_4_extensions[HORIZONTAL_SCROLL] == TRUE) )
  {
  left_margin = 1;
  right_margin = columns_per_page;
  left_right_margins_mode = FIXED;
  }
}
```

Known Deviations: None

SET/RESET ORIGIN MODE DECOM

Levels: 1-4

Purpose: Change the state of Origin Mode between Absolute (reset) and Displaced (set) origin.

To Set: CSI ? 6 h (displaced)
To Reset: CSI ? 6 l (absolute - default)

Description: A conforming device shall allow addressing of the display area to be performed relative to either the display origin (coordinate 1,1) [reset state = absolute] or the origin of the current scrolling region (Top Margin, Left Margin) [set state = displaced]. When Origin Mode is in the set state (displaced), the Active Position cannot be moved above the Top Margin or left of the Left Margin of the scrolling region.

The DECOM controls are used the change the state of Origin Mode between Absolute and Displaced.

State Affected:

origin_mode
active_position

Algorithm:

```
void set_origin_mode()
{
origin_mode = DISPLACED;
active_position.line = top_margin;
active_position.column = left_margin;
}


void reset_origin_mode()
{
origin_mode = ABSOLUTE;
active_position.line = 1;
active_position.column = 1;
}
```

Known Deviations: None

SET/RESET SCROLLING MODE DECSCLM

Levels: 1-4

Purpose: Change the state of Scrolling Mode between Jump (reset) and Slow (set) scrolling.

To Set: CSI ? 4 h (smooth - default)
To Reset: CSI ? 4 l (jump)

Description: A conforming device shall provide the capability of selecting either jump scrolling [reset state = jump] and slow scrolling [set state = slow]. Jump scrolling is defined as scrolling of the display as fast as the device permits, to a maximum which will be implementation defined. Slow scrolling is defined as scrolling of the display at a rate of approximately 6 text lines per second. The intention of slow scrolling is to move text through the display at a rate which makes it "recognizable", but not necessarily readable. (i.e., it is not guaranteed that it will appear slowly enough for the user to scan read the entire text). The scrolling speed applies to vertical scrolling of the scrolling region of the display, which can be set as any combination of adjacent lines.

The DECSCLM controls are used to change the state of Scrolling Mode between Jump (reset state) and Slow (set scrolling.

Notes:

1. Some implementations may offer a choice of additional scrolling speeds in Set-Up. The reset state of DECSCLM will select the speed closest to 6 lines per second.

2. Setting the left or right margins to other than their default values (the boundaries of the Logical Display Page) may temporarily disable smooth scrolling (guideline), but will not affect the state of Scrolling Mode.

State Affected:

scrolling_mode

Algorithm:

```
void set_scrolling_mode()
{
scrolling_mode = SLOW;
}


void reset_scrolling_mode()
{
scrolling_mode = JUMP;
}
```

Known Deviations: Workstation terminal emulators which do not have scrolling hardware do not implement this control (DECterm, UIS).

INDEX IND

Levels: 1-4

Purpose: Move the Active Position downward one line, scrolling if necessary.

Format: IND
8/4 (ESC D)

Description: The IND control moves the Active Position downward in the display by one line. If the Active Position is already at the Bottom Margin the display will scroll upward by one line. If the display scrolls, a blank line with all attributes off will appear at the bottom margin.

Notes:

1. If the Active Position is above the Bottom Margin when this control is executed, the Active Position will not move beyond the Bottom Margin. If the Active Position is below the Bottom Margin (as the result of absolute cursor positioning) it will still move downward by one line and no scrolling will occur. In this case, the Active Position will not move beyond the bottom line of the display.

2. This control is identical to the Line Feed (LF) control, except that it is not affected by the setting of New Line Mode.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void index()
{
if (active_position.line == bottom_margin)
  scroll_up(top_margin, 1);
else
  if (active_position.line < lines_per_page)
    active_position.line += 1;

/* new line may have a different length - adjust column */
if (active_position.column > end_of_line(active_position.line))
  active_position.column = end_of_line(active_position.line);
}
```

Known Deviations:

On the VT100 and VT125, Index is affected by the setting of New Line Mode, in the same manner as Line Feed.

REVERSE INDEX RI

Levels: 1-4

Purpose: Move the Active Position upward one line, scrolling if necessary.

Format: RI
8/13 (ESC M)

Description: The RI control moves the Active Position upward in the display by one line. If the Active Position is already at the Top Margin the display will scroll downward by one line. If the display scrolls, a blank line with all attributes off will appear at the top margin.

Notes:

1. If the Active Position is below the Top Margin when this control is executed, the Active Position will not move beyond the Top Margin. If the Active Position is above the Top Margin (as the result of absolute cursor positioning) it will still move upward by one line and no scrolling will occur. In this case, the Active Position will not move beyond the first line of the display.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void reverse_index()
{
if (active_position.line == top_margin)
  scroll_down(top_margin, 1);
else
  if (active_position.line > 1)
    active_position.line -= 1;

/* new line may have a different length - adjust column */
if (active_position.column > end_of_line(active_position.line))
  active_position.column = end_of_line(active_position.line);
}
```

Known Deviations: None

FORWARD INDEX DECFI

Levels: 4x (Horizontal Scrolling)

Purpose: Move the Active Position forward one column, scrolling if necessary.

Format:

ESC	9
1/11	3/9

Description: The DECFI control causes the active position to move forward one column. If the active position was already at the right margin, the contents of the Logical Display Page within the right, left, top and bottom margins shifts left one column. The column shifting beyond the left margin is deleted. A new column is inserted at the right margin with all attributes turned off and the cursor appears in this column.

If the active position is outside the left or right margin when the command is received the active position moves forward one column. If the active position was at the right edge of the page, the command is ignored.

Notes on DECFI:

1. Lines of text are shifted one column regardless of their line attributes. This means double-wide and double-size lines will appear to shift twice as fast as single-width lines.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void decfi()
{
if ( (conformance_level == LEVEL_4) &&
     (level_4_extensions[HORIZONTAL_SCROLL] == TRUE) )
  {
  if (active_position.column == right_margin)
    scroll_left(left_margin, 1);
  else
    if (active_position.column < columns_per_page)
      active_position.column += 1;

  /* active line may not be columns_per_page long */
  if (active_position.column > end_of_line(active_position.line))
    active_position.column = end_of_line(active_position.line);
  }
}
```

Known Deviations: None

BACK INDEX DECBI

Levels: 4x (Horizontal Scrolling)

Purpose: Move the Active Position backward one column, scrolling if necessary.

Format:

ESC	6
1/11	3/6

Description: The DECBI control causes the active position to move backward one column. If the active position was already at the left margin, the contents of the Logical Display within the left, right, top and bottom margin shifts right one column. The column shifted beyond the right margin is deleted. A new column is inserted at the left margin with all attributes turned off and the cursor appears in this column.

If the active position is outside the left or right margin when the command is received the active position moves backwards one column. If the active position was at the left edge of the page, the command is ignored.

Notes on DECBI:

1. Lines of text are shifted one column regardless of their line attributes. This means double-wide and double-size lines will appear to shift twice as fast as single-width lines.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void decbi()
{
if ( (conformance_level == LEVEL_4) &&
     (level_4_extensions[HORIZONTAL_SCROLL] == TRUE) )
  {
  if (active_position.column == left_margin)
    scroll_right(left_margin, 1);
  else
    if (active_position.column > 1)
      active_position.column -= 1;
  }
}
```

Known Deviations: None

5.4.4 Cursor Movement

This section describes the following functions used to control the cursor position within the Logical Display Page:

CUU	Cursor Up
CUD	Cursor Down
CUF	Cursor Forward
CUB	Cursor Backward
CUP	Cursor Position
HVP	Horizontal/Vertical Position
CPR	Cursor Position Report
DECXCPR	Extended Cursor Report
LNM	Line Feed/New Line Mode
CR	Carriage Return
LF	Line Feed
VT	Vertical Tab
FF	Form Feed
BS	Backspace
NEL	Next Line

CURSOR UP CUU

Levels: 1-4

Purpose: Move the Active Position upward by n lines.

Format:	CSI	Pn	A	default Pn: 1
	9/11	3/?	4/1	

Description: The CUU control moves the Active Position upward in the display. The distance moved is determined by the parameter value. A parameter value of zero or one moves the Active Position upward by one line. A parameter value of n moves the Active Position upward by n lines.

Notes:

1. If the Active Position is at or below the Top Margin when the CUU control is executed, and an attempt is made to move the Active Position above the Top Margin, the control will be executed until the Active Position reaches the Top Margin. The Active Position will not move beyond the Top Margin.

2. If the Active Line is above the Top Margin when the CUU control is executed (due to absolute cursor positioning) and an attempt is made to move the Active Position outside of the addressable display, the control will be executed until the Active Position reaches the first line of the Page. The Active Position will not move beyond the first line of the Page.

3. No scrolling will occur as a result of this control.

4. If the line to which the Active Position is moved contains fewer columns than the Active Column, the Active Column is set to the end of that line. (This condition will only occur if the Active Line is a single-width line, and the line to which the Active Position is moved is a double-width or double-height line. The Active Column is not affected by the length of lines it moves over by a repeat move caused by a parameter value greater than one.)

State Affected:

active_position

Algorithm:

```
void cursor_up(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  short int n;
if (parv[0] == 0) n = 1;
else n = parv[0];
if (active_position.line >= top_margin)
  {
  if ((active_position.line - n) >= top_margin)
    active_position.line -= n;
  else
    active_position.line = top_margin;
  }
else
  {
  if ((active_position.line - n) >= 1)
    active_position.line -= n;
  else
    active_position.line = 1;
  }

/* new line may have a different length - adjust column */
if (active_position.column > end_of_line(active_position.line))
  active_position.column = end_of_line(active_position.line);
}
```

Known Deviations: None

CURSOR DOWN CUD

Levels: 1-4

Purpose: Move the Active Position downward by n lines.

Format:	CSI	Pn	B	default Pn: 1
	9/11	3/?	4/2	

Description: The CUD control moves the Active Position downward in the display. The distance moved is determined by the parameter value. A parameter value of zero or one moves the Active Position downward by one line. A parameter value of n moves the Active Position downward by n lines.

Notes:

1. If the Active Position is at or above the Bottom Margin when the CUD control is executed, and an attempt is made to move the Active Position below the Bottom Margin, the control will be executed until the Active Position reaches the Bottom Margin. The Active Position will not move beyond the Bottom Margin.

2. If the Active Position is below the Bottom Margin when the CUD control is executed (due to absolute cursor positioning) and an attempt is made to move the Active Position outside of the addressable display, the control will be executed until the Active Position reaches the bottom line of the Page. The Active Position will not move beyond the bottom line of the Page.

3. No scrolling will occur as a result of this control.

4. If the line to which the Active Position is moved contains fewer columns than the Active Column, the Active Column is set to the end of that line. (This condition will only occur if the Active Line is a single-width line, and the line to which the Active Position is moved is a double-width or double-height line. The Active Column is not affected by the length of lines it moves over by a repeat move caused by a parameter value greater than one.)

State Affected:

active_position

Algorithm:

```
void cursor_down(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  short int n;
if (parv[0] == 0) n = 1;
else n = parv[0];
if (active_position.line <= bottom_margin)
  {
  if ((active_position.line + n) <= bottom_margin)
    active_position.line += n;
  else
    active_position.line = bottom_margin;
  }
else
  {
  if ((active_position.line + n) <= lines_per_page)
    active_position.line += n;
  else
    active_position.line = lines_per_page;
  }

/* new line may have a different length - adjust column */
if (active_position.column > end_of_line(active_position.line))
  active_position.column = end_of_line(active_position.line);
}
```

Known Deviations: None

CURSOR FORWARD CUF

Levels: 1-4

Purpose: Move the Active Position forward by n columns.

Format:	CSI	Pn	C	default Pn: 1
	9/11	3/?	4/3	

Description: The CUF control moves the Active Position forward in the display. The distance moved is determined by the parameter value. A parameter value of zero or one moves the Active Position forward by one column. A parameter value of n moves the Active Position forward by n columns.

Notes:

1. If the Active Position is at or inside the Right Margin when the CUF control is executed, and an attempt is made to move the Active Position beyond the Right Margin, the control will be executed until the Active Position reaches the Right Margin. The Active Position will not move beyond the Right Margin.

2. If the Active Position is beyond the Right Margin when the CUF control is executed (due to absolute cursor positioning) and an attempt is made to move the Active Position outside of the addressable display, the control will be executed until the Active Position reaches the last column of the Active Line. The Active Position will not move beyond the last column of the Active Line.

3. No scrolling will occur as a result of this control.

State Affected:

active_position

Algorithm:

```
void cursor_forward(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  short int n;
if (parv[0] == 0) n = 1;
else n = parv[0];
if (active_position.column <= right_margin)
  {
  if ((active_position.column + n) <= right_margin)
    active_position.column += n;
  else
    active_position.column = right_margin;
  }
else
  {
  if ( (active_position.column + n) <=
       end_of_line(active_position.line) )
    active_position.column += n;
  else
    active_position.column = end_of_line(active_position.line);
  }
}
```

Known Deviations: None

CURSOR BACKWARD CUB

Levels: 1-4

Purpose: Move the Active Position backward by n columns.

Format:	CSI	Pn	D	default Pn: 1
	9/11	3/?	4/4	

Description: The CUB control moves the Active Position backward in the display. The distance moved is determined by the parameter value. A parameter value of zero or one moves the Active Position backward by one one column. A parameter value of n moves the Active Position backward by n columns.

Notes:

1. If the Active Position is at or inside the Left Margin when the CUB control is executed, and an attempt is made to move the Active Position beyond the Left Margin, the control will be executed until the Active Position reaches the Left Margin. The Active Position will not move beyond the Left Margin.

2. If the Active Position is beyond the Left Margin when the CUB control is executed (due to absolute cursor positioning) and an attempt is made to move the Active Position outside of the addressable display, the control will be executed until the Active Position reaches the first column of the Active Line. The Active Position will not move beyond the first column of the Active Line.

3. No scrolling will occur as a result of this control.

State Affected:

active_position

Algorithm:

```
void cursor_backward(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  short int n;
if (parv[0] == 0) n = 1;
else n = parv[0];
if (active_position.column >= left_margin)
  {
  if ((active_position.column - n) >= left_margin)
    active_position.column -= n;
  else
    active_position.column = left_margin;
  }
else
  {
  if ((active_position.column - n) >= 1)
    active_position.column -= n;
  else
    active_position.column = 1;
  }
}
```

Known Deviations: None

CURSOR POSITION CUP

Levels: 1-4

Purpose: To position the Active Position at an absolute line and column address.

Format:

	CSI	Pl	;	Pc	H	
	9/11	3/?	;	3/?	4/8	
default Pl: 1						
default Pc: 1						

Description: The CUP control moves the Active Position to an absolute line and column address as specified by the parameter values. The first parameter specifies the new line address, and the second parameter specifies the new column address. If either parameter is omitted, or is explicitly set to zero (0), that parameter value will default to one (1).

Notes:

1. This control is affected by the setting of Origin Mode. If Origin Mode is reset, addressing is performed relative to the page origin (first column in the first line). If Origin Mode is set, addressing is performed relative to the origin of the current scrolling region (the top and left margin).

2. If an attempt is made to move the Active Position outside of the addressable display area (the entire screen if Origin Mode is reset, or the scrolling region if Origin Mode is set), the Active Position will be moved in the direction indicated to the boundary of the addressable area, but will not be moved beyond the boundary of the addressable area.

State Affected:

active_position

Algorithm:

```
void cursor_position(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  line_t y;
  column_t x;
y = parv[0];
x = parv[1];
if (y == 0) y = 1;
if (x == 0) x = 1;
if (origin_mode == DISPLACED)
  {
  if ( ((top_margin-1) + y) <= bottom_margin )
    active_position.line = (top_margin-1) + y;
  else
    active_position.line = bottom_margin;

  if ( ((left_margin-1) + x) <= right_margin )
    active_position.column = (left_margin-1) + x;
  else
    active_position.column = right_margin;
  }
else
  {
  if (y <= lines_per_page)
    active_position.line = y;
  else
    active_position.line = lines_per_page;

  if (x <= end_of_line(active_position.line))
    active_position.column = x;
  else
    active_position.column = end_of_line(active_position.line);
  }
}
```

Known Deviations: None

HORIZONTAL/VERTICAL POSITION HVP

Levels: 1-4

Purpose: Move the Active Position to an absolute line and column address. (fallback implementation)

Format:

	CSI	Pl	;	Pc	f	default Pl: 1
	9/11	Pl	;	Pc	6/6	default Pc: 1

Description: The HVP control moves the Active Position to an absolute line and column address as specified by the parameter values. The first parameter specifies the new line address, and the second parameter specifies the new column address. If either parameter is omitted, or is explicitly set to zero (0), that parameter value will default to one (1).

Notes:

1. The HVP control is provided for interface compatibility with printing terminals. This implementation is not in conformance with existing standards. Conforming software will use the Cursor Position control to affect movement of the cursor, since the operation of Horizontal/Vertical Position may be redefined in future levels of the architecture.

2. This control is affected by the setting of Origin Mode. If Origin Mode is reset, addressing is performed relative to the page origin (first column in the first line). If Origin Mode is set, addressing is performed relative to the origin of the current scrolling region (the top and left margin).

3. If an attempt is made to move the Active Position outside of the addressable display area (the entire screen if Origin Mode is reset, or the scrolling region if Origin Mode is set), the Active Position will be moved in the direction indicated to the boundary of the addressable area, but will not be moved beyond the boundary of the addressable area.

State Affected:

active_position

Algorithm:

```
void horizontal_vertical_position(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  /* steal code from CUP */
  cursor_position(parv);
}
```

Known Deviations: None

CURSOR POSITION REPORT CPR

Levels: 1-4

Purpose: Report the line and column address of the Active Position.

Request Format:

CSI	6	n	
9/11	3/6	6/14	

Report Format:

CSI	Pl	;	Pc	R		default Pl: 1
9/11	3/?	;	3/?	5/2		default Pc: 1

Description: The CPR control is transmitted by the terminal in response to a specific Device Status Report (DSR) control. The parameter values indicate the Active Line and Active Column values at the time the DSR control is received by the terminal.

Notes:

1. This control is affected by the setting of Origin Mode. If Origin Mode is reset, addressing is performed relative to the page origin (first column in the first line). If Origin Mode is set, addressing is performed relative to the origin of the current scrolling region (the Top and Left Margin).

State Affected:

none

Algorithm:

```
void cursor_position_report()
{
  line_t y;
  column_t x;
if (origin_mode == DISPLACED)
  {
  y = active_position.line - top_margin + 1;
  x = active_position.column - left_margin + 1;
  }
else
  {
  y = active_position.line;
  x = active_position.column;
  }
send_char(0x9B, host_port);  /* CSI */
send_int(y, host_port);
send_char(';', host_port);
send_int(x, host_port);
send_char('R', host_port);
}
```

Known Deviations: None

EXTENDED CURSOR POSITION REPORT DECXCPR

Levels: 3x (Windowing), 4

Purpose: Report the line, column, and page address of the Active Position.

Request Format:

CSI	?	6	n
9/11	3/15	3/6	6/14

Report Format:

CSI	?	Pl	;	Pc	;	Pp	R
9/11	3/15	3/?	3/11	3/?	3/11	3/?	5/2

default Pl, Pc, Pp: 1

Description: The DECXCPR control is transmitted by the terminal in response to a specific Device Status Report (DSR) control. The parameter values indicate the Active Line, Column, and Page values at the time the DSR control is received by the terminal. The default condition with no parameters present, or parameters of 0, is equivalent to a cursor at the home position on the first Page.

Notes:

1. This control is affected by the setting of Origin Mode. If Origin Mode is reset, addressing is performed relative to the page origin (first column in the first line). If Origin Mode is set, addressing is performed relative to the origin of the current scrolling region (the Top and Left Margin).

State Affected: None

Algorithm:

```
void x_cursor_position_report()
{
  line_t y;
  column_t x;
if (origin_mode == DISPLACED)
  {
  y = active_position.line - top_margin + 1;
  x = active_position.column - left_margin + 1;
  }
else
  {
  y = active_position.line;
  x = active_position.column;
  }
send_char(0x9B, host_port);  /* CSI */
send_char('?', host_port);
send_int ( y , host_port);
send_char(';', host_port);
send_int ( x,  host_port);
send_char(';', host_port);
send_char(active_position.page, host_port);
send_char('R', host_port);
}
```

Known Deviations: None

SET/RESET NEW LINE MODE LNM

Levels: 1-4

Purpose: Change the state of New Line Mode between New Line Off (reset) and New Line On (set).

Set Format: CSI 20 h
Reset Format: CSI 20 l

Description: A conforming device shall provide a means of selecting whether a received Line Feed (LF), Vertical Tab (VT), or Form Feed (FF) character shall be treated as a single instance of that character [reset state = no_new_line], or that each of these characters is handled on receipt by additionally returning the Active Position to the first column of the display [set state = new_line]. It should be noted that setting of this mode also affects the processing of the Carriage Return (CR) key, which transmits a two-code sequence (CR LF) when New Line Mode is in the set state.

(Note: This mode should not be used in the set state by conforming software.)

State Affected:

new_line_mode

Algorithm:

```
void set_new_line_mode()
{
new_line_mode = NEW_LINE_ON;
}


void reset_new_line_mode()
{
new_line_mode = NEW_LINE_OFF;
}
```

Known Deviations: None

CARRIAGE RETURN CR

Levels: 1-4

Purpose: Move the Active Position to the Left Margin.

Format: CR
0/13

Description: The CR control moves the Active Position to the Left Margin. If the Active Position is before the left margin, or outside the top and bottom margins, the cursor moves to the first column of the Active Line.

State Affected:

active_position

Algorithm:

```
void_carriage_return()
{
if ((active_position.column >= left_margin) &&
     (active_position.line >= top_margin) &&
     (active_position.line <= bottom_margin))
  active_position.column = left_margin;
else
  active_position.column = 1;
}
```

Known Deviations: None

LINE FEED LF

Levels: 1-4

Purpose: Move the Active Position downward one line, scrolling if necessary.

Format: LF
0/10

Description: The LF control moves the Active Position downward in the display by one line. If the Active Position is already at the Bottom Margin the display will scroll upward by one line.

Notes:

1. If the Active Position is above the Bottom Margin when this control is executed, the Active Position will not move beyond the Bottom Margin.

2. If the Active Position is below the Bottom Margin when this control is executed (as the result of absolute cursor positioning) it will still move downward by one line and no scrolling will occur. In this case, the Active Position will not move beyond the bottom line of the display.

3. If the line to which the Active Position is moved contains fewer columns than the Active Column, the Active Column is set to the end of that line.

4. This control is affected by the setting of New Line Mode. If this mode is reset, no change will occur in the Active Column unless the line below the Active Line is shorter than the Active Line (see note 2). If the mode is set, the control causes the Active Column to be set to the first column of the line into which the Active Position is moved. Conforming software is recommended to use the two character sequence (CR LF) rather than New Line mode for widest compatibility.

5. Note that the fundamental differences between this control and the CUD control are the effect of New Line Mode and the occurrence of scrolling on reaching the Bottom Margin.

6. The LF control is identical in function to the IND (Index) control when New Line Mode is in the reset (No New Line) state.

State Affected:

```
    display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
    active_position
```

Algorithm:

```
void line_feed()
{
| if (new_line_mode == NEW_LINE_ON) carriage_return();
if (active_position.line == bottom_margin)
  scroll_up(top_margin, 1);
else
|   if (active_position.line < lines_per_page)
      active_position.line += 1;

/* new line may have different length - adjust column */
if (active_position.column > end_of_line(active_position.line))
  active_position.column = end_of_line(active_position.line);
}
```

Known Deviations: None

VERTICAL TAB VT

Levels: 1-4

Purpose: Move the Active Position downward one line, scrolling if necessary. (fallback implementation)

Format: VT
0/11

Description: The VT control moves the Active Position downward in the display by one line. If the Active Position is already at the Bottom Margin the display will scroll upward by one line.

Notes:

1. The implementation of the VT control is a fallback mechanism for display terminals which is intended to provide compatibility with printers which correctly implement this function.

2. This control is identical in function to the Line Feed (LF) control. It is provided for software compatibility with printer output. It should be noted, however, that printers will implement this function differently. It is therefore recommended that software use the Line Feed control to perform this function instead of Vertical Tab.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void vertical_tab()
{
line_feed();    /* fallback implementation */
}
```

Known Deviations: None

FORM FEED FF

Levels: 1-4

Purpose: Move the Active Position downward one line, scrolling if necessary. (fallback implementation)

Format: FF
0/12

Description: The FF control moves the Active Position downward in the display by one line. If the Active Position is already at the Bottom Margin the display will scroll upward by one line.

Notes:

1. The implementation of the FF control is a fallback mechanism for display terminals which is intended to provide compatibility with printers which correctly implement this function.

2. This control is identical in function to the Line Feed (LF) control. It is provided for software compatibility with printer output. It should be noted, however, that printers will implement this function differently. It is therefore recommended that software use the Line Feed control to perform this function instead of Form Feed.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void form_feed()
{
line_feed();    /* fallback implementation */
}
```

Known Deviations: None

BACK SPACE BS

Levels: 1-4

Purpose: Move the Active Position one column to the left.

Format: BS
0/8

Description: The BS control moves the Active Position backward in the display by one column.

Notes:

1. If the Active Position is at or inside the Left Margin and within the Scroll Area, it will not move beyond the Left Margin.

2. If the Active Position is already before the Left Margin, the Active Position will move left one column unless it is at the first column of the Active Line. The Active Position will not move beyond the beginning of the Active Line.

3. The BS control is not affected in any way by the setting of Auto Wrap Mode. Under no circumstances does the Active Position advance to the previous line on reaching the beginning of the Active Line.

State Affected:

active_position

Algorithm:

```
void back_space()
{
if (active_position.column > left_margin)
  active_position.column -= 1;
else
  {
  if ( ((active_position.column != left_margin) ||
        (active_position.line < top_margin) ||
        (active_position.line > bottom_margin)) &&
       (active_position.column > 1)  )
    active_position.column -= 1;
  }
}
```

Known Deviations: None

NEXT LINE NEL

Levels: 1-4

Purpose: Move the Active Position to the Left Margin of the next line, scrolling if necessary.

Format: NEL
8/5 (ESC E)

Description: The NEL control moves the Active Position to the Left Margin of the next line. If the Active Line is equal to the Bottom Margin, the Active Line is not incremented, but instead the display scrolls upward by one line. If the Active Position is before the left margin, or outside the top and bottom margins, it will move to the first column of the next line, but not beyond the last line on the page.

Notes:

1. This control is identical to sending Line Feed (LF) with Line Feed New Line Mode set. It is also identical to sending Carriage Return (CR) followed by Line Feed (LF). Software is recommended to always use the combination of Carriage Return followed by Line Feed (CR LF) to achieve this effect for widest compatibility.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void next_line()
{
carriage_return();
line_feed();
}
```

Known Deviations: None

5.4.5 Horizontal Tabulation

A conforming device shall provide the ability to set a horizontal tabulation stop at each display column. These boolean indicators must be capable of being set or reset in any possible combination. It should be noted that only tab stops are stored by the device, not tab characters.

Setting of horizontal tab stops may be performed both as a remote and local function, and therefore the settings at any point in time is indeterminate. However, when no local provision is made for the setting and/or storage of user defined tabulation stops, the following default states shall be used by conforming devices. These settings should also apply to the factory default settings for a conforming device.

tab stops =

columns 9,17,25,33,41,49,57,65,73,81,89,97,105,113,121,129

TABULATION CLEAR TBC

Levels: 1-4

Purpose: Clear Horizontal Tab Stops.

Format:	CSI	Ps	g	default Ps: 0
	9/11	3/?	6/7	

Description: The TBC control provides a means of clearing or removing previously set Horizontal Tab Stops. The range of positions affected by this control is determined by the parameter value. An omitted parameter or a parameter value of zero (0) cause the control to clear only a tabulation stop set at the Active Column. A parameter value of three (3) causes the control to clear all Horizontal Tab Stops in the display.

State Affected:

```
horizontal_tab_stops[MAX_NUM_COLUMNS]
```

Algorithm:

```
void tabulation_clear(parc, parv)
  short int parc;
  short int parv[MAX_NUM_PARAMETERS];
{
  column_t x;
  short int n;
if (parc == 0) parc = 1;
for (n=0; n<parc; n++)
  {
  if (parv[n] == 0)
    horizontal_tab_stops[active_position.column] = FALSE;
  else if (parv[n] == 3)
    for (x=1; x<= columns_per_page; x++)
      horizontal_tab_stops[x] = FALSE;
  }
}
```

Known Deviations: None

HORIZONTAL TAB HT

Levels: 1-4

Purpose: Advance the Active Position to the next Horizontal Tab Stop.

Format: HT
0/9

Description: The HT control moves the Active Position forward in the display to the next Horizontal Tab Stop in the Active Line.

Notes:

1. If no tabulation stop is reached before the Right Margin of the Active Line (within the Scroll Area), the Active Position will be set to the Right Margin of the Active Line. If the Active Column is beyond the Right Margin or the Active Line is outside the Scroll Area (due to absolute cursor positioning), the Active Position will move to the next tab stop, or last column of the Active Line, whichever comes first.

State Affected:

active_position

Algorithm:

```
void horizontal_tab()
{
 boolean_t outside;
if ((active_position.line > top_margin) &&
    (active_position.line < bottom_margin))
  outside = FALSE;
else
  outside = TRUE;
if ( (active_position.column <
      end_of_line(active_position.line)) &&
     ((active_position.column != right_margin) || (outside==TRUE))
)
  active_position.column += 1;

for (; ( (active_position.column <
          end_of_line(active_position.line)) &&
         (horizontal_tab_stops[active_position.column] != TRUE) &&
         ((active_position.column != right_margin) ||
(outside==TRUE)) );
       active_position.column += 1);
}
```

Known Deviations: None

HORIZONTAL TABULATION SET HTS

Levels: 1-4

Purpose: Set a Horizontal Tab Stop at the Active Column.

Format: HTS

8/8 (ESC H)

Description: The HTS control causes a Horizontal Tab Stop to be set at the column position indicated by the value of the Active Column at the time this control is received.

Notes:

1. None of the other Horizontal Tab Stop settings are affected by execution of this control.

State Affected:

horizontal_tab_stops[MAX_NUM_COLUMNS]

Algorithm:

```
void horizontal_tabulation_set()
{
horizontal_tab_stops[active_position.column] = TRUE;
}
```

Known Deviations: None

5.4.6 Page Size And Arrangement

Level 1 and Level 2 conforming devices support a single Logical Display Page of 24 lines by 80 columns, extendable to 24 lines by 132 columns when the "132 Column Mode" extension is present (DECCOLM).

Level 4 devices, and Level 3 devices with the "Windowing Extension" shall support commands to change the Logical Display Page Size (DECSLPP and DECSCPP). The actual Page Sizes supported shall be clearly specified in the product documentation, and means shall be provided for software to interrogate the size of the Logical Display Page (see "Terminal State Interrogation"). The Page Size and number of pages in the Logical Display may also be configurable by the user under local control.

SET/RESET COLUMN MODE DECCOLM

Levels: 1X, 2X, 3X, 4X

Purpose: Change the state of Column Mode between Eighty (reset) and One Thirty Two (set).

Set Format: CSI ? 3 h
Reset Format: CSI ? 3 l

Description: A conforming device may provide as an extension the capability of selecting 80-column [reset state = eighty] or 132-column [set state = one_thirty_two] display format.

Selecting Eighty or One_Thirty_Two Column Mode causes the display data to be cleared, the scrolling region to be eliminated, and the Active Position to be set to the display origin (1,1), even if the terminal was already in the selected state.

Notes:

1. The font used to display Page Memory may change when the number of columns on the page changes to better match the screen. Control of when the font changes may be provided as a Set-Up option.

2. If the Host Writable Status Line is enabled, receipt of this sequence will clear both the main display and the Host Writable Status Line.

State Affected:

display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
line_rendition[MAX_NUM_LINES]
active_position
top_margin
bottom_margin
column_mode
left_margin
right_margin
columns_per_page
left_right_margins_mode

Algorithm:

```
void set_column_mode()
{
  line_t y;
  column_t x;
column_mode = ONE_THIRTY_TWO;
| columns_per_page = 132;
for (y=1; y<MAX_NUM_LINES; y++)
  {
  for (x=1; x<MAX_NUM_COLUMNS; x++)
    display[y][x].code = EMPTY_CHARACTER;
  line_rendition[y] = SINGLE_WIDTH;
  }
active_position.line = 1;
active_position.column = 1;
top_margin = 1;
| bottom_margin = lines_per_page;
| left_margin = 1;
| right_margin = columns_per_page;
| left_right_margins_mode = SETABLE;
}


void reset_column_mode()
{
  line_t y;
  column_t x;
column_mode = EIGHTY;
| columns_per_page = 80;
for (y=1; y<MAX_NUM_LINES; y++)
  {
  for (x=1; x<MAX_NUM_COLUMNS; x++)
    display[y][x].code = EMPTY_CHARACTER;
  line_rendition[y] = SINGLE_WIDTH;
  }
active_position.line = 1;
active_position.column = 1;
top_margin = 1;
| bottom_margin = lines_per_page;
| left_margin = 1;
| right_margin = columns_per_page;
| left_right_margins_mode = SETABLE;
}
```

Known Deviations: None

SET COLUMNS PER PAGE DECSCPP

Levels: 3X (Windowing), 4

Purpose: Set the number of columns per page in the Logical Display.

Format:

	CSI	Pn	$	
	9/11	3/?	2/4	7/12

Description: This sequence sets the number of columns in the display page according to the numeric parameter. If the parameter is omitted or zero, the page width is defaulted to 80. If the number of columns specified is not one of the values supported by the implementation, the next higher supported number of columns per page is assumed. If the number of columns specified exceeds the maximum number supported, the maximum number supported will be used.

DECSCPP does not cause the Active Position to move. However if the active position is beyond the width of the new page when the command is executed, the active position will be moved to the right edge of the page. DECSCPP does not cause the Right Margin to move. However if the right margin is beyond the width of the new page when the command is executed, the right margin will be moved to the right edge of the page. When changing from a 132 column to a 80 column page, the information in columns 1-80 is preserved. The information in columns 81-132 is cleared. When changing from an 80 column page to a 132 column page, the information in columns 1-80 is preserved.

Notes on DECSCPP:

1. The font used to display Page Memory may change when the number of columns on the page changes to better match the screen. Control of when the font changes may be provided as a Set-Up option.

2. The number of columns per page can also be changed through Column Mode (DECCOLM). The affect of DECSCPP is similar to DECCOLM except that DECSCPP does not clear page memory.

3. DECSCPP does not change the setting of DECLRMM.

Implementation Guideline

The VT420 can configure its off-screen memory to pages of either 80 or 132 columns wide. When the page width changes, the corresponding 80 or 132

column font is automatically selected.

State Affected:

display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
right_margin
columns_per_page

Algorithm:

```
void set_columns_per_page(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  line_t y;
  column_t x;
if ( (conformance_level == LEVEL_4) ||
     (level_3_extensions[WINDOWING]==TRUE) )
  {
  if (parv[0] <= 80)
    /* set columns per page to 80 */
    {
    for (y=1; y<MAX_NUM_LINES; y++)
      {
      for (x=81; x<MAX_NUM_COLUMNS; x++)
        display[y][x].code = EMPTY_CHARACTER;
      }
    if (active_position.column > 80) active_position.column = 80;
    if (right_margin > 80) right_margin = 80;
      /* optional font change */
      if (columns_per_page > 80) select_normal_font();
    columns_per_page = 80;
    }
  else
    /* set columns per page to 132 */
      /* optional font change */
      if (columns_per_page <= 80) select_condensed_font();
    columns_per_page = 132;
  }
}
```

Known Deviations: None

SET LINES PER PAGE DECSLPP

Levels: 3X (Windowing), 4

Purpose: Set the number of lines per page in the Logical Display.

Format:

CSI	Pn	t
9/11	3/?	7/4

Description: This sequence sets the number of display lines per page according to the numeric parameter. If the parameter is omitted or zero, the page length is defaulted to 24. If the number of lines specified is not one of the values supported by the implementation, the next higher supported number of lines per page is assumed. If the number of lines specified exceeds the maximum number supported, the maximum number supported will be used.

DECSLPP does not cause Page Memory to be cleared, however, information contained in a larger page may appear in successive smaller pages, if the new height of the pages selected by the DECSLPP is too small to hold the information.

DECSLPP does not cause the scrolling regions to be reset, or the Active Position to move, except under the following circumstances: The scrolling margins may be reset to the extremes of the new Page Configuration if the current scrolling margin exceeds the physical limits of the new Page height; and, the Active Position will move to the same column in the maximum available line, if the Active Position exceeds the available number of lines in the new page size.

Conformance to Level 4 requires support for 3 pages of 24 lines.

Implementation Guideline

The VT420 can configure its off-screen memory to either 3 pages of 24 lines, 2 pages of 25 lines, 2 pages of 36 lines, or 1 page of 72 lines in double-session mode, or 6 pages of 24 lines, 5 pages of 25 lines, 4 pages of 36 lines, 2 pages of 72 lines, or 1 page of 144 lines in single-session mode.

State Affected:

- lines_per_page
- number_of_pages
- active_position

```
|       top_margin
|       bottom_margin
|
|
|   Algorithm:
|
|   void set_lines_per_page(parv)
|     short int parv[MAX_NUM_PARAMETERS];
|   {
|   if ( (conformance_level == LEVEL_4) ||
|        (level_3_extensions[WINDOWING]==TRUE) )
|     {
|     if (parv[0] <= 24)
|       {
|       lines_per_page = 24;
|       number_of_pages = 3;
|       }
|     else if (parv[0] == 25)
|       {
|       lines_per_page = 25;
|       number_of_pages = 2;
|       }
|     else if (parv[0] <= 36)
|       {
|       lines_per_page = 36;
|       number_of_pages = 2;
|       }
|     else
|       {
|       lines_per_page = 72;
|       number_of_pages = 1;
|       }
|
|     if (active_position.line > lines_per_page)
|       active_position.line = lines_per_page;
|     if (active_position.page > number_of_pages)
|       active_position.page  = number_of_pages;
|     if (top_margin > lines_per_page)
|       top_margin = 1;
|     if (bottom_margin > lines_per_page)
|       bottom_margin = lines_per_page;
|     }
|   }
|
|
|   Known Deviations:  None
```

5.4.7 Page Movement

NEXT PAGE NP

Levels: 3x (Windowing), 4

Purpose: To move the Active Position ahead one or more pages in the Logical Display.

Format:

CSI	Pn	U
9/11	3/?	5/5

Description: This sequence causes the Active Position to move ahead a number of pages, specified by the parameter. If the parameter value is zero (the default) or one, the active position moves ahead 1 page. If the sequence calls for a page beyond the last page of display memory, the highest numbered page is used. NP moves the Active Position to the home position in the new page obeying Origin Mode.

Whether or not the new page is displayed depends upon the setting of DECPCCM.

State Affected:

active_position

Algorithm:

```
void next_page(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
if ( (conformance_level == LEVEL_4) ||
     (level_3_extensions[WINDOWING]==TRUE) )
  {
  if (parv[0] == 0) parv[0] = 1;
  if ((active_position.page + parv[0]) <= number_of_pages)
    active_position.page += parv[0];
  else
    active_position.page = number_of_pages;
  if (origin_mode = DISPLACED)
    {
    active_position.line = top_margin;
    active_position.column = left_margin;
    }
  else
    {
    active_position.line = 1;
    active_position.column = 1;
    }
  }
}
```

Known Deviations: None

PRECEDING PAGE PP

Levels: 3x (Windowing), 4

Purpose: To move the Active Position back one or more pages in the Logical Display.

Format:

CSI	Pn	V
9/11	3/?	5/6

Description: This sequence causes the active position to move back a number of pages, specified by the parameter. If the parameter value is zero (the default) or one, the active position moves back 1 page. If the sequence calls for a page before the first page of display memory, the first page is used. PP moves the active position to the home position in the new page obeying Origin Mode.

Whether or not the new page is displayed depends upon the setting of DECPCCM.

State Affected:

active_position

Algorithm:

```
void preceding_page(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
if ( (conformance_level == LEVEL_4) ||
     (level_3_extensions[WINDOWING]==TRUE) )
  {
  if (parv[0] == 0) parv[0] = 1;
  if (active_position.page > parv[0])
    active_position.page -= parv[0];
  else
    active_position.page = 1;
  if (origin_mode = DISPLACED)
    {
    active_position.line = top_margin;
    active_position.column = left_margin;
    }
  else
    {
    active_position.line = 1;
    active_position.column = 1;
    }
  }
}
```

Known Deviations: None

PAGE POSITION ABSOLUTE PPA

Levels: 3x (Windowing), 4

Purpose: To move the Active Position to the specified page in the Logical Display.

Format:	CSI	Pn	SP	P	(default Pn: 1)
	9/11	3/?	2/0	5/0	

Description: This sequence moves the Active Position to the corresponding line and column on the n-th page. If there are fewer than n pages in display memory, the highest numbered page is used.

Whether or not the new page is displayed depends upon the setting of DECPCCM.

State Affected:

active_position

Algorithm:

```
void page_position_absolute(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
if ( (conformance_level == LEVEL_4) ||
     (level_3_extensions[WINDOWING]==TRUE) )
  {
  if (parv[0] == 0) parv[0] = 1;
  if (parv[0] <= number_of_pages)
    active_position.page = parv[0];
  else
    active_position.page = number_of_pages;
  }
}
```

Known Deviations: None

PAGE POSITION RELATIVE PPR

Levels: 3x (Windowing), 4

Purpose: To move the Active Position forward to the n-th succeding page in the Logical Display.

Format:	CSI	Pn	SP	Q	(default Pn: 1)
	9/11	3/?	2/0	5/1	

Description: This sequence moves the active position to the corresponding line and column of the n-th succeding page. If the sequence calls for a page beyond the last page of display memory, the highest numbered page is used.

Whether or not the new page is displayed depends upon the setting of DECPCCM.

State Affected:

active_position

Algorithm:

```
void page_position_relative(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
if ( (conformance_level == LEVEL_4) ||
     (level_3_extensions[WINDOWING]==TRUE) )
  {
  if (parv[0] == 0) parv[0] = 1;
  if ((active_position.page + parv[0]) <= number_of_pages)
    active_position.page += parv[0];
  else
    active_position.page = number_of_pages;
  }
}
```

Known Deviations: None

PAGE POSITION BACKWARD PPB

Levels: 3x (Windowing), 4

Purpose: To move the Active Position back to the n-th preceding page in the Logical Display.

Format:	CSI	Pn	SP	R	(default Pn: 1)
	9/11	3/?	2/0	5/2	

Description: This sequence moves the active position to the corresponding line and column on the n-th preceding page. If the sequence calls for a page before the first page of display memory, the first page is used.

Whether or not the new page is displayed depends upon the setting of DECPCCM.

State Affected:

active_position

Algorithm:

```
void page_position_backward(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
if ( (conformance_level == LEVEL_4) ||
     (level_3_extensions[WINDOWING]==TRUE) )
  {
  if (parv[0] == 0) parv[0] = 1;
  if (active_position.page > parv[0])
    active_position.page -= parv[0];
  else
    active_position.page = 1;
  }
}
```

Known Deviations: None

5.5 WINDOWING EXTENSION

The Windowing Extension supports Logical Display Page sizes that are larger or smaller than the physical display. A rectangular area of the screen (physical display) that displays information from a Logical Display Page is called a window.

Conforming devices may provide local window management features for creating and manipulating more than one window on the physical display at a time (guideline).

The following functions described in this subsection are used to control how an area of the Logical Display is mapped to a window on the physical display.

DECHCCM	Horizontal Cursor Coupling Mode
DECVCCM	Vertical Cursor Coupling Mode
DECPCCM	Page Cursor Coupling Mode
DECRQDE	Request Displayed Extent
DECSNLS	Select Number of Lines per Screen (documented exception)
SU	Pan Down
SD	Pan Up

The Windowing Extension also supports two commands for changing the the size of the Logical Display Page (DECSLPP and DECSCPP, see "Page Size and Arrangement"). Finally, if more than one page of display memory is supported, the Windowing Extension includes commands to move the active position among the pages (NP, PP, PPA, PPR, and PPB, see "Page Movement").

The Windowing Extension may be provided at Level 1 or higher of the Character Cell Display service class, and is indicated by the extension parameter 18 in the Device Attributes (DA) response.

SET/RESET HORIZONTAL CURSOR COUPLING MODE DECHCCM

Levels: 3x, 4x (Windowing)

Purpose: Change the state of Horizontal Cursor Coupling Mode between coupled (set state, page moves horizontally in the window to keep cursor visible), and uncoupled (reset state).

To set: CSI ? 60 h (coupled)
To reset: CSI ? 60 l (uncoupled - default)

Description: DECHCCM is a parameter to the Set Mode and Reset Mode commands. In the reset (uncoupled) state, when the active position within a page goes left or right of the borders of the displayable area (window), the page does not move with respect to the window to keep the cursor visible. The cursor is not shown in this case. The cursor is said to be "uncoupled" from the display. In the set state, when the active position within a page goes left or right of the borders of the displayable area (window), the page is moved the minimum amount to an 8 column boundary in the horizontal direction to keep the cursor position visible. The cursor is said to be "coupled" to the display.

Notes on DECHCCM:

1. Horizontal Cursor Coupling does not need to be provided when there is no way for a Logical Display Page to have more columns than can be shown in the narrowest window.

2. Horizontal Cursor Coupling may be enabled or disabled under local control in Set-Up.

3. If a Set Mode command setting DECHCCM is received while the Active Column is not visible on the display, the page is moved to include the Active Column.

4. Whether or not the cursor is actually visible may also depend on the state of Vertical and Page Cursor Coupling (DECVCCM and DECPCCM).

5. Existing implementations move the page the minimum amount to make the cursor visible. Experience suggests moving the minimum amount to an 8 column boundary would make the display easier to follow for the user (less frequent movement).

SET/RESET VERTICAL CURSOR COUPLING MODE DECVCCM

Levels: 3x, 4x (Windowing)

Purpose: Change the state of Vertical Cursor Coupling Mode between coupled (set state, page moves vertically in window to keep cursor visible), and uncoupled (reset state).

To set: CSI ? 61 h (coupled - default)
To reset: CSI ? 61 l (uncoupled)

Description: DECVCCM is a parameter to the Set Mode and Reset Mode commands. In the reset state, when the active position within a page goes above or below the borders of the displayable area (window), the page does not move with respect to the window to keep the cursor visible. The cursor is not shown in this case. The cursor is said to be "uncoupled" from the display. In the set state, when the active position within a page goes above or below the borders of the displayable area (window), the page is moved the minimum amount in the vertical direction to keep the cursor position visible. The cursor is said to be "coupled" to the display.

Notes on DECVCCM:

1. Vertical Cursor Coupling may be enabled or disabled under local control in Set-Up.

2. If a Set Mode command setting DECVCCM is received while the Active Line is not visible on the display, the page is moved to include the Active Line.

3. Whether or not the cursor is actually visible may also depend on the state of Horizontal and Page Cursor Coupling (DECHCCM and DECPCCM).

SET/RESET PAGE CURSOR COUPLING MODE DECPCCM

Levels: 3x, 4x (Windowing)

Purpose: Change the state of Page Cursor Coupling Mode between coupled (set state, switch page in window to keep cursor visible), and uncoupled (reset state).

To set: CSI ? 64 h (coupled - default)
To reset: CSI ? 64 l (uncoupled)

Description: DECPCCM is a parameter to the Set Mode and Reset Mode commands. In the reset state, when the active position moves to a new page in a multiple page configuration, the display is not refreshed to keep the cursor visible. The cursor is not shown in this case. The cursor is said to be "uncoupled" from the display. In the set state, when the active position moves to a new page, the destination page is refreshed on the Main Display to keep the cursor position visible. The cursor is said to be "coupled" to the display.

Notes on DECPCCM:

1. Page Cursor Coupling may be enabled or disabled under local control in Set-Up.

2. If a Set Mode command setting DECPCCM is received when the Active Page containing the cursor is not visible on the display, the Active Page is refreshed on the Main Display.

REQUEST/REPORT DISPLAYED EXTENT DECRQDE/DECRPDE

Levels: 3x, 4x (Windowing)

Purpose: To determine the area of page memory currently visible in the display window.

Request form:

CSI	"	v
9/11	2/2	7/6

Report form:

CSI	Ph ; Pw ; Pml ; Pmt ; Pmp	"	w
9/11	Ph ; Pw ; Pml ; Pmt ; Pmp	2/2	7/7

Description: The DECRQDE sequence, which takes no parameters, causes the device to respond with a DECRPDE control function describing the area of Page Memory being displayed on the physical screen.

The DECRPDE parameters provide the following information:

Ph - The number of lines (height) of Page Memory being displayed in the window. This does not include any status line (see "Status Display"). If no lines from the active session are visible, 0 is returned (see "Session Management").

Pw - The number of columns (width) of Page Memory being displayed in the window. Pw is not necessarily 0 when Ph is 0. Ph should be used to determine whether any lines are visible.

Pml - The column in Page Memory that is displayed in the leftmost column of the window.

Pmt - The line in Page Memory that is displayed in the topmost line of the window.

Pmp - The page of Page Memory that is being displayed through the window.

SELECT NUMBER OF LINES PER SCREEN DECSNLS

Levels: Documented Exception (Windowing)

Purpose: To select a font size and corresponding maximum number of lines that will fit on the physical display.

Format:	CSI	Pn	*	
	9/11	Pn	2/10	7/12

Description: DECSNLS selects the maximum number of lines which can be displayed on the screen by choosing a corresponding font size.

Pn is a numeric parameter between 1 and 255 which indicates the number of lines which can be displayed on the screen at one time. If a value not directly supported by the terminal is selected, the next larger size supported will be used. If Pn is greater than the largest number of lines available, the largest number supported will be used. The number of lines per screen that can be supported will be clearly specified in the product documentation.

Although DECSNLS selects a font height which determines the maximum number of lines which can be displayed, the actual number of data lines displayed can be limited by other factors including the Page Size, whether there is a Status Display, and local window management functions if any. Applications can determine the actual number of lines displayed using the Request Display Extent (DECRQDE) control sequence.

Notes on DECSNLS:

1. The terminal will not display more lines than are contained on a single page within a single window.

2. A Status Display, if enabled, may occupy lines that would otherwise be available for showing data from the Logical Display Page.

3. Unused lines will be left blank at the bottom of the window, or screen.

4. DECSNLS is a "User Preference Feature" intended for terminal management use. It should not be modified by application software except in response to a user request. The terminal may provide a means to lock DECSNLS under local control (User Features Lock in Set-Up) to prevent host modification.

5. It is suggested that software specify Pn to match the Logical Display page sizes supported. This will provide the best compatibility between implementations.

6. The terminal may provide a local mechanism to change DECSNLS automatically when the Page Size is changed ("auto resize window").

Implementation Guideline

The VT420 supports three different font heights which allow either 24, 39 or 49 data lines to be displayed on the screen, plus a status line.

PAN DOWN (SCROLL UP) SU

Levels: 3x, 4x (Windowing)

Purpose: To cause the current page to move up with respect to the window making another line visible at the bottom.

Format:	CSI	Pn	S	(default Pn: 1)
	9/11	Pn	5/3	

Description: Causes the entire visible contents of the current page to move up one or more lines on the physical display according to the parameter. The line at the top border of the window is no longer seen, and a new line appears at bottom border of the window for each position moved. A numeric parameter of zero or one causes the page to move one line.

The window cannot be moved beyond the boundaries of the page.

This control affects what is visible on the physical display only. No movement of data within the Logical Display occurs. The Active Position does not change.

PAN UP (SCROLL DOWN) SD

Levels: 3x, 4x (Windowing)

Purpose: To cause the current page to move down with respect to the window making another line visible at the top.

Format:	CSI	Pn	T	(default Pn: 1)
	9/11	Pn	5/4	

Description: Causes the entire visible contents of the current page to move down one or more lines on the physical display according to the parameter. The line at the bottom border of the window is no longer seen, and a new line appears at top border of the window for each position moved. A numeric parameter of zero or one causes the page to move one line.

The window cannot be moved beyond the boundaries of the page.

This control affects what is visible on the physical display only. No movement of data within the Logical Display occurs. The Active Position does not change.

5.6 VISUAL RENDITIONS

This subsection describes functions used to control the visual appearance of characters on the display. Attributes to be controlled include screen background (light or dark), character size (double wide and double size line attributes), character emphasis (bold, blink, underline, reverse), and character color when the Color Text Extension is implemented.

The following control functions are described herein:

- DECSCNM - Screen Mode
- Line Renditions
  - DECSWL - Single Width Line
  - DECDWL - Double-Width Line
  - DECDHLT - Double-Height Line Top
  - DECDHLB - Double-Height Line Bottom
- Character Renditions
  - SGR - Select Graphic Rendition
- Color Text Extension (SGR parameters)
  - DECCTR - Color Table Report
  - DECSTGLT - Select Text/Graphics Look-Up Table (Documented Exception)

SET/RESET SCREEN MODE DECSCNM

Levels: 1-4

Purpose: Change the state of Screen Mode between Normal (reset) and Reverse (set) screen.

Set Format: CSI ? 5 h
Reset Format: CSI ? 5 l

Description: A conforming device shall provide the capability of operating the physical display in either "normal" [reset state = normal] or "reverse" [set state = reverse] screen. The Screen Mode setting allows the terminal operator to define his screen state with respect to normal character rendition. Changing the mode reverses the definitions of the foreground and background colors. The actual color values are not specified by the architecture, and are implementation defined. All character renditions must be achievable as defined on both normal and reverse screens.

Note: Conforming software should not depend on the speed of execution when changing Screen Mode. It may take as long as the time required to repaint the entire screen, depending on the hardware implementation.

State Affected:

screen_mode

Algorithm:

```
void set_screen_mode()
{
screen_mode = REVERSE_SCREEN;
}


void reset_screen_mode()
{
screen_mode = NORMAL_SCREEN;
}
```

Known Deviations: None

5.6.1 Line Renditions

A conforming device shall support the following line renditions:

- single-height, single-width
- single-height, double-width
- double-height, double-width top
- double-height, double-width bottom

Only one rendition may be set per line (they are mutually exclusive). Setting double-width or double-height renditions doubles the character cell width within the line, but does not affect the addressing of characters in the line (within the constraints of the physical display). For example, the character in column 5 of a single-width line will still be addressed as column 5 if the line rendition is changed to double-width, although the character field will now start in column 9 of the physical display. Furthermore, when the line rendition is set to double-width or double-height, character positions cannot be addressed which are more than half the current value of the right edge of the display.

Note: Conforming software will not draw only the top or the bottom of double-height lines. When double-height lines are used, both the top and bottom must be displayed in adjacent line positions. Otherwise, the visual display of the line and the cursor symbol registration is UNDEFINED.

SINGLE-WIDTH LINE DECSWL

Levels: 1-4

Purpose: Set the Line Rendition of the Active Line to single-width

Format:

	ESC	#	5
	1/11	2/3	3/5

Description: Setting this attribute causes all characters indicated by the codes contained in the Active Line to be displayed in single-width rendition.

Notes:

1. Setting this attribute on a line which was previously double-width will cause the line length to be doubled.

2. Setting this attribute does not cause data contained in the Active Line to be lost.

State Affected:

```
line_rendition[MAX_NUM_LINES]
active_position
```

Algorithm:

```
void single_width_line()
{
  short int base;
base = (active_position.page-1) * lines_per_page;
line_rendition[active_position.line + base] = SINGLE_WIDTH;
}
```

Known Deviations: None

DOUBLE-WIDTH LINE DECDWL

Levels: 1-4

Purpose: Set the Line Rendition of the Active Line to double-width.

Format:

ESC	#	6
1/11	2/3	3/6

Description: Double-width characters are achieved by displaying the characters indicated by the codes contained in the Active Line in double-width rendition.

Notes:

1. Setting this attribute causes the length of the Active Line to be reduced by half.
2. Setting this attribute causes data which is beyond the end of the Active Line after execution of the control to be lost.
3. This control does not operate when DECLRMM (Left Right Margin Mode) is set.

State Affected:

```
line_rendition[MAX_NUM_LINES]
active_position
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
```

Algorithm:

```
void double_width_line()
{
  column_t x;
  short int base;
base = (active_position.page-1) * lines_per_page;
line_rendition[active_position.line + base] = DOUBLE_WIDTH;
if (active_position.column > end_of_line(active_position.line))
  active_position.column = end_of_line(active_position.line);
for (x=end_of_line(active_position.line)+1;
     x<=columns_per_page;
     x++)
  {
  display[active_position.line + base][x].code = EMPTY_CHARACTER;
  }
}
```

Known Deviations: None

DOUBLE-HEIGHT LINE DECDHLT,DECDHLB

Levels: 1-4

Purpose: Set the Line Rendition of the Active Line to double-height top or double-height bottom.

Format:

ESC	#	3		top
1/11	2/3	3/3		
ESC	#	4		bottom
1/11	2/3	3/4		

Description: Double-height characters are achieved by setting the Line Rendition of adjacent lines to double-height top and double-height bottom. Setting double-height top causes the upper half of the characters indicated by the codes contained in the Active Line to be displayed in double-height and double-width rendition. Setting double-height bottom causes the lower half of the characters contained in the Active Line to be displayed in double-height and double-width rendition.

Notes:

1. Conforming software will not draw only the top or the bottom of double-height lines. When double-height lines are used, both the top and bottom must be displayed in adjacent line positions. Otherwise, the visual display of the line and the cursor symbol registration is UNDEFINED.

2. Since double-height also implies double-width, setting this attribute causes the length of the Active Line to be reduced by half.

3. Setting this attribute causes data which is beyond the end of the Active Line after execution of the control to be lost.

4. This control does not operate when DECLRMM (Left Right Margin Mode) is set.

State Affected:

```
line_rendition[MAX_NUM_LINES]
active_position
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
```

Algorithm:

```
void double_height_line_top()
{
  column_t x;
  short int base;
base = (active_position.page-1) * lines_per_page;
line_rendition[active_position.line + base] = DOUBLE_HEIGHT_TOP;
if (active_position.column > end_of_line(active_position.line))
  active_position.column = end_of_line(active_position.line);
for (x=end_of_line(active_position.line)+1;
     x<=columns_per_page;
     x++)
  {

  display[active_position.line + base][x].code = EMPTY_CHARACTER;
  }
}


void double_height_line_bottom()
{
  column_t x;
  short int base;
base = (active_position.page-1) * lines_per_page;
line_rendition[active_position.line + base] = DOUBLE_HEIGHT_TOP;
if (active_position.column > end_of_line(active_position.line))
  active_position.column = end_of_line(active_position.line);
for (x=end_of_line(active_position.line)+1;
     x<=columns_per_page;
     x++)
  {
  display[active_position.line + base][x].code = EMPTY_CHARACTER;
  }
}
```

Known Deviations: None

5.6.2 Character Renditions

Each character cell in the display has a set of visual attributes associated with it. These attributes can be defined using the concept of foreground and background values. A character cell consists of a rectangular area filled with a background color, and a character pattern written in a foreground color. Background and foreground values are at least binary in nature (i.e., they must have at least two clearly distinguishable visual states). Characters are rendered in the physical display by some combination of the background and foreground states. The actual values and meanings of the background and foreground colors are not specified by the architecture, and are implementation defined.

A conforming device shall provide normal, bold, underline, blink, and reverse renditions for each character in the display. These renditions can be combined in any manner for each character position independently. The architecture requires that each rendition and every possible rendition combination be individually distinguishable one from the other, without affecting the ability to recognize the character pattern within the cell. Graphic renditions apply to all graphic character codes (2/1 to 7/14 inclusive, and 10/0 to 15/15 inclusive) as well as to the Space character (2/0).

5.6.2.1 Normal Rendition -

Normal rendition defines the rendering of characters in the physical display in its "natural" state, that is, where the character is rendered in the display foreground against a field of display background. Normal rendition is the absence of any special rendition of the character cell. (Note: Normal rendition of characters is dependent on the state of the Screen Mode, which is used to define the values of the foreground and background colors.)

5.6.2.2 Bold Rendition - Bold rendition of characters may be achieved either by emphasizing the display foreground without changing the background characteristics of the cell, or by intensifying the background color while holding the foreground value constant.

5.6.2.3 Blink Rendition - Blink rendition is achieved by alternating the stable rendition of a character with a different rendition at a fixed rate, with a duty cycle of approximately one second.

5.6.2.4 Underscore Rendition - Underscore rendition is achieved by the addition of a single horizontal bar extending the full width of the character field and rendered in the display foreground. The bar must be easily distinguishable from the underline character (5/15).

5.6.2.5 Reverse Rendition - Reverse rendition is achieved by using the current value of the foreground color as the cell background, and the current value of the background color as the foreground.

SELECT GRAPHIC RENDITION SGR

Levels: 1-4

Purpose: Designate the graphic rendition to be applied to all subsequent characters entered into the display.

Format:

CSI	Ps ; ... ; Ps	m	default Ps: 0
9/11	Ps ; ... ; Ps	6/13	

Description: The SGR control selects the graphic rendition (visual attribute) to be applied to all subsequent characters entered into the display. The parameter values indicate the combination of renditions to be selected as indicated in the following table (for both Level 1 and Level 2):

Parameter	Rendition
0	All renditions off
1	Bold
4	Underscore
5	Blink
7	Negative (reverse) image

The following additional values are recognized in Level 2 operation. Although these values may be recognized when the terminal is operating as a Level 1 device, conforming software shall not rely on this feature.

Parameter	Rendition
22	Normal intensity (Bold off)
24	Not Underscore
25	Steady (not Blink)
27	Positive image

Notes:

1. All other parameter values shall be ignored unless they are part of a well defined extension to the architecture.

2. Renditions may be selected in any combination, and will be rendered appropriately. Selecting additional renditions does not affect the setting of previously selected renditions (newly selected renditions will be combined with previously selected renditions) unless all renditions are turned off (parameter value zero (0)).

3. In no way does the selection of current rendition affect the rendition of characters already entered into the display.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
```

Algorithm:

```
void select_graphic_rendition(parc, parv)
  short int parc;
  short int parv[MAX_NUM_PARAMETERS];
{
  short int n;
if (parc == 0) parc = 1;
for (n=0; n<parc; n++)
  {
  switch(parv[n])
    {
    case 0:
      current_rendition.bold = FALSE;
      current_rendition.underscore = FALSE;
      current_rendition.blink = FALSE;
      current_rendition.reverse = FALSE;
      break;
    case 1:
      current_rendition.bold = TRUE;
      break;
    case 4:
      current_rendition.underscore = TRUE;
      break;
    case 5:
      current_rendition.blink = TRUE;
      break;
    case 7:
      current_rendition.reverse = TRUE;
      break;
    default:
      break;
    }
  if (conformance_level == LEVEL_2)
    {
    switch(parv[n])
      {
      case 22:
        current_rendition.bold = FALSE;
        break;
      case 24:
        current_rendition.underscore = FALSE;
```

```
            break;
          case 25:
            current_rendition.blink = FALSE;
            break;
          case 27:
            current_rendition.reverse = FALSE;
            break;
          default:
            break;
          }
        }
      }
    }
```

Known Deviations: None

5.6.3 ANSI Color Text Extension

The Color Text Extension provides for the selection of text foreground and background colors on a per character basis using SGR color parameters. The Color Text extension may be provided at Level 2 or higher of the Character Cell Display service class, and is indicated by the extension parameter 22 in the Device Attributes (DA) response.

5.6.3.1 SGR Color Selection -

A conforming device shall recognize parameter values 30-37 and 39 to the SELECT GRAPHIC RENDITION (SGR) control sequence to select the foreground color attribute and parameter values 40-47 and 49 to select the background color attribute, as specified in the table. The eight colors listed as factory defaults shall be available at a minimum. These colors are taken from the ANSI X3.64 standard.

SGR	meaning	factory default color
30	foreground color 0	black
31	foreground color 1	red
32	foreground color 2	green
33	foreground color 3	yellow
34	foreground color 4	blue
35	foreground color 5	magenta
36	foreground color 6	cyan
37	foreground color 7	white
39	default foreground color	<device-dependent>
40	background color 0	black
41	background color 1	red
42	background color 2	green
43	background color 3	yellow
44	background color 4	blue
45	background color 5	magenta
46	background color 6	cyan
47	background color 7	white
49	default background color	<device-dependent>

A device may provide means, either under software or local control, to modify these colors and may provide means to save the modified colors in non-volatile memory.

5.6.3.2 Interaction With Other Visual Renditions -

The foreground color and background color are maintained as part of the current_rendition. As new characters are written to the

display, the current_rendition is used to update the attribute bits associated with each character in display memory.

Selecting the Reverse Rendition (SGR parameter 7) will, if the Reverse Rendition is not already set, be equivalent to invoking the current foreground color as the new background color, and the current background color as the new foreground color. Likewise, selecting the Positive Rendition (SGR parameter 27) will, if the Positive Rendition is not already set, be equivalent to invoking the current foreground color as the new background color, and the current background color as the new foreground color.

SGR parameters 30-37 and 39, the nine foreground color specifiers, will set the foreground color without regard to whether the Reverse Rendition is set. Likewise, SGR parameters 40-47 and 49, the nine background color specifiers, will set the background color without regard to whether the Reverse Rendition is set.

SGR parameter 0 (all renditions off) will reset the foreground and background color attributes to their defaults (equivalent to SGR parameters 39 and 49). SOFT TERMINAL RESET (DECSTR) will similarly cause the foreground and background color attributes to be reset.

When the Color Text Extension is present, the Bold Rendition (SGR parameter 1) should be distinguished by a heavier weight font (Guideline). It is preferable not to use a separate color (increased intensity) for bolding in order to keep it distinct from the other color text attributes.

Changing between normal and reverse screen mode (DECSCNM) will affect which attribute bits from display memory are used to render the text foreground and background color. It does not affect the current_rendition, or how the current_rendition is applied to the character attribute bits associated with each character in display memory.

The architecture does not specify the default text foreground or background colors. It is recommended however that normal screen mode correspond to lighter text on a darker background (not mandatory). The factory default for screen mode is device dependent. A device may provide means to save the screen mode in non-volatile memory so as to override the factory default at power-up.

5.6.3.3 Color Maps - A color map is a table that associates a binary value in display memory with a color that can be produced on the display. During display refresh, the active color map is used to translate color values from display memory to colors that appear on the screen. The size of the active color map cannot exceed 2**N entries where N is the number of bits used to

represent a color in display memory (often the number of bit planes).

A device can have many color maps of which only one is active at time, or can assign color map entries from a single color map to different applications, limiting the size of each application's color map. Color maps can be read-only (fixed), or read-write (modifiable under software control).

Applications often model color selection as asking for a particular color (non-indexed, absolute color specification, read-only), or asking for a particular color map entry (indexed color, read-write). When using indexed color, it is normal to allow the application to modify a color map entry causing all objects written with that color index to change color on the display.

Conforming devices may implement the ANSI SGR colors as indexed or non-indexed, and may allow selecting between these two models under local control.

5.6.3.4 Color Table Report (DECCTR) -

A conforming device that supports the TSI and Color Text Extensions, and has a color table shall be capable of reporting the color table in the form of a DECCTR sequence as follows.

Color Table Request/Report/Restore DECCTR

Levels: 2x, 3x, 4x (TSI and Color Text)

Purpose: To Request, Report, and Restore the contents of the color table.

Request Format:	CSI 9/11	2 3/2	; 3/11	Pu 3/?	$ 2/4	u 7/5		(DECRQTSR Ps=2)
Report Format:	DCS 9/0	2 3/2	$ 2/4	s 7/3	D...D D...D	ST 9/12		(DECTSR Ps=2)
Restore Format:	DCS 9/0	2 3/2	$ 2/4	p 7/0	D...D D...D	ST ST		(DECRSTS Ps=2)

Where D...D consists of groups of 5 parameters as follows:

Pc; Pu; Px; Py; Pz / Pc; Pu; Px; Py; Pz / ...

Pc is the color number 0-255.

Pu is the color coordinate system
- 0 = illegal
- 1 = HLS (hue 0-360, lightness 0-100, saturation 0-100)
- 2 = RGB (red 0-100, green 0-100, blue 0-100)

Px, Py, Px are values in the color coordinate system specified by Pu.

; is the semicolon character (3/11)
/ is the slach character (2/15)

Description: DECRQTSR with parameter 2 is sent from the host to the terminal to request a color table report. The terminal responds with a DECTSR control with parameter 2 and each color table entry given as a group of 5 parameters in the data string. To load or restore a previously reported color map, the host transmits the DECRSTS control with parameter 2 and the desired color map definitions in the data string. When loading the color map, only the index entries specified are changed.

When using DECRQTSR to request a color table report, you can include an additional parameter Pu to select the color coordinate

system the terminal uses to report the color map (0=default, 1=HLS, 2=RGB).

5.6.3.5 Default Color Assignment (Guideline) -

When indexed color is used, the following color map assignments are recommended to maximize compatibility among implementations when the colors are not specified explicitly (not mandatory). "Normal" text foreground and background refers to when Screen Mode is reset.

Four or more plane (16 or more color) color map:

Color	Mono	Text Use
0 black	black	normal background
1 blue	gray-2	
2 red	gray-4	
3 green	gray-6	
4 magenta	gray-1	
5 black	gray-3	black text
6 white	gray-5	white text
7 gray 50%	white	normal foreground
8 gray 25%	black	
9 blue*	gray-2	blue text
10 red*	gray-4	red text
11 green*	gray-6	green text
12 magenta*	gray-1	magenta text
13 cyan*	gray-3	cyan text
14 yellow*	gray-5	yellow text
15 gray 75%	white	bold text (VT340)

* = may be less saturated

If more than four planes are supported, this series of 16 colors can be repeated as needed.

Explainatory Note

DECterm uses these assignments directly, though colors corresponding to map entries 9-14 are not less saturated.

The VT340 uses these assignments except that color 5 is cyan, and color 6 is yellow. These were changed to black and white to make black and white ANSI text colors not dependent on the default foreground and background colors in DECterm.

Color map entries 0-3 match the VT240 default colors.

5.6.3.6 Color Interaction Among Modes And Data Syntaxes (Guideline) -

ANSI Color Text, Sixels, and ReGIS can all support indexed color using a similar color model. A conforming device may share a single color map among these protocols so that changing a color definition in one protocol will affect the color map used by the other protocols. Support for the option of sharing the same color map is recommended, but not mandatory. Conforming software should not depend on cross protocol side effects if maximum portability is desired.

5.6.3.7 Alternative Text Rendition Mapping (Guideline) -

Some previous terminals which do not support ANSI Color Text allowed other SGR parameters to select text colors.

VT241
Color Entry

0	normal text background
1	reverse graphic rendition (SGR 7)
2	bold graphic rendition (SGR 1)
3	normal text foreground

For backward compatibility, the VT340 provides an alternative text rendition mapping mode selectable from Set-Up.

VT340 Alternative Text Rendition Mapping
Color Entry

0	normal text background
8	reverse graphic rendition (SGR 7)
15	bold graphic rendition (SGR 1)
7	normal text foreground

When alternative text rendition mapping is not selected, the reverse attribute is rendered by exchanging the text foreground and background colors.

A conforming device may provide an alternative text rendition mapping mode selectable in Set-Up, but this is not part of the Color Text Extension. The factory default behavior should be to treat bold and reverse graphic renditions independently of color selection (not mandatory).

Select Text/Graphics Look-Up Table DECSTGLT

Levels: 3 (Documented Exception)

Purpose: To select color look-up table used and the mapping of text renditions to look-up table entries.

Format:

CSI	Ps	)	{
9/11	3/?	2/9	7/11

Ps	Look-Up Table
0	monochrome look-up table (default)
1	color look-up table, standard text rendition mapping
2	color look-up table, alternative text rendition mapping

Description: DECSTGLT permits host selection of the video color look-up table used in the device, and the mapping of text renditions to look-up table entries. This control function assumes the existence of two color look-up tables: monochrome and color. When a color look-up table is selected, two text rendition mappings can be defined, called "standard" and "alternative". The "standard" mapping permits the reverse video rendition to use a different color look-up table entry for the background of displayed text. The "alternative" mapping restricts text rendition mapping to a single background entry, and one or two foreground entries (two if the bold graphic rendition is effected through increased intensity). In other words, the alternative mapping supports the use of bold and reverse graphic renditions to select text foreground colors.

This control function is not intended to supplant the Color Text Extension, but instead provide a means of backward compatibility to earlier video devices which provided several different text rendition mappings based on different monitor configurations.

5.7 AUDIBLE INDICATOR

A conforming device shall provide a warning "bell", or some similar audible indicator suitable to signal the operator's attention. It may be possible to change the volume of the bell sound, or disable it entirely, as a local Setup function.

WARNING BELL BEL

Levels: 1-4

Purpose: Rings the terminal bell.

Format: BEL

0/7

Description: The BEL control causes the terminal to emit an audible tone of brief duration.

Notes:

1. It may be possible to change the volume of the bell sound, or disable it entirely, as a local Setup function.

State Affected: none

5.8 GRAPHIC CHARACTER SETS

The terminal shall maintain a repertory of available character sets which may be used to interpret received data characters to be displayed. The character sets must be selectable on a per character basis, and are mutually exclusive (that is, there can be only one character set value for each character position in the Logical Display). Refer to the Code Extension Chapter of this standard (DEC STD 70-3) for more information on designating and invoking character sets, G-sets, and the structure of the in-use table.

5.8.1 Character Set Repertoire

The table below shows the graphic character sets required at each Conformance Level, and those available as Extensions.

Character Set(s)	Level at which required, or availabe as Extension
U.S. ASCII	1-4
DEC Line Drawing	1-4
DEC Supplemental Graphic	2-4
ISO Latin-1 Supplemental	2x, 3-4
UPSS	2x, 3-4
UK set (United Kingdom)	1x
NRCS	1x, 2x, 3x, 4x
JIS_ROMAN, JIS_KATAKANA	2x, 3x, 4x
DRCS	2x, 3x, 4x
DEC Technical	3x, 4x

Notes:

1. ISO Latin-1 Supplemental is available at Level 2 as part of the 8-bit Interface Architecture Extension.

2. UPSS (User Preference Supplemental Set) is not actually a character set in itself, but a pointer to one of the supplemental sets (usually DEC Supplemental or ISO Latin-1 Supplemental). UPSS is available at Level 2 as part of the 8-bit Interface Architecture Extension.

3. The UK set is an optional extension to Level 1 which if present, may be selected in Set-Up to replace ASCII (ASCII and UK cannot be designated simultaneously). This extension was implemented on the VT100, but has since been replaced by the NRCS Extension.

4. NRCS (National Replacement Character Set) is an extension which provides twelve 7-bit NRC sets, one of which may be selected under local user control (Set-Up) to replace

U.S. ASCII as the default set. See DECNRCM. Depending on the implementation, only one NRC set corresponding to the keyboard dialect may be selectable at a time.

5. JIS_ROMAN and JIS_KATAKANA are part of the Katakana Extension.

6. DRCS (Dynamically Redefinable Character Set) is an extension which allows a soft character set to be downloaded from the host and assigned a designating sequence possibly replacing an existing character set. See Chapter 10 of this standard (DEC STD 70-10), DRCS Extension.

The character sets themselves are described in DEC STD 169 Digital Standard Coded graphic Character Sets for Hardware and Software.

5.8.2 Character Set Selection

Character sets are identified to the terminal using designating escape sequences which must be registered in DEC Standard 138. Each sequence consists of zero, one, or two intermediate characters in the range 2/0 through 2/15, followed by a final character in the range 3/0 through 7/15, the combination of which uniquely identifies the character set to be designated. (Note: the designating sequence is preceded by the Escape character (1/11) and a single intermediate value 2/8, 2/9, 2/10, or 2/11 indicating which of the four G-sets the specified character set is to be designated into. For a complete description of the designating and invoking process, see the section "Code Extension Layer", DEC STD 70-3.)

DESIGNATE CHARACTER SET SCS

Levels: 1-4

Purpose: Designate the graphic character sets.

Format: ESC <G-Set designator> <Character set designator>

The G-Set designators are:

94 character sets

G0	(	2/8
G1	)	2/9
G2	*	2/10
G3	+	2/11

96 character sets

G1	-	2/13
G2	.	2/14
G3	/	2/15

The Character set designators are:

ASCII_G	B	4/2
LINE_DRAWING_G	0	3/0
DEC_SUPPLEMENTAL_G	%5	2/5 3/5
ISO_LATIN1_SUPPLEMENTAL_G	A	4/1
UPSS_G	<	3/12
DRCS_G	I I F	2/? 2/? ?/?
TCS_G	>	
JIS Roman	J	4/10
JIS Katakana	I	4/9

NRCS

British	A	
Dutch	4	(discontinued in L3)
Finnish	5 or C	
French	R	
French Canadian	Q	
German	K	
Italian	Y	
Norwegian/Danish	E or 6	
Portuguese	%6	
Spanish	Z	
Swedish	7 or H	
Swiss	=	

For example,

ESC (B designates ASCII_G to G0
ESC) 0 designates Line Drawing to G1
ESC * < designates the UPSS to G2

Description: The character set designating escape sequences provide the means of selecting the G0, G1, G2, and G3 character sets, which can be invoked into the left (GL) and right (GR) sides of the In Use Table. (For a complete description of the G-sets and the In Use Table, see the section "Code Extension Layer".)

Notes:

1. In Level 1 operation character sets can only be designated into G0 and G1, with G2 and G3 containing the ASCII character set by default.

2. Conforming software will not change the default designations of the G0 and G2 character sets. The G1 and G3 sets will be used when it is desired to designate alternate character sets. Furthermore, conforming software will not designate ASCII_G to either G2 or G3, and will not designate Supplemental_G to either G0 or G1.

3. Designating a character set which is invoked into the In Use Table at the time the control is executed causes the In Use Table to be changed to reflect the change to the designated set.

4. As a Level 1 extension, the UK (United Kingdom) Character Set may be provided, and designated with the sequence ESC I A (1/11 I 4/1). If this extension is supported, it may also be possible to locally select the UK set to replace ASCII as the default 7-bit set. In this case, the UK set will be used in all instances in which ASCII is employed throughout the Video Systems Reference Manual when the terminal is in Level 1 operation.

5. On traditional video terminals with the NRCS extension, it is permissable to only recognize the one NRC set corresponding to the keyboard dialect at a time. Workstation terminal emulators may extend the character set table to recognize all NRC sets regardless of the current keyboard dialect (not mandatory).

State Affected:

designated_graphic_sets[4]
in_use_table

Algorithm:

```
/***************
 *
 * Initialize the character set table
 *   This procedure should be called on power-up or
 *   reset to initialize the character set table
 *   with the appropriate designating sequences.
 *
 **************/
void intialize_character_set_table()
{
char_set_table[TABLE_ASCII].name = ASCII_G;
char_set_table[TABLE_ASCII].first_intermediate = 0;
char_set_table[TABLE_ASCII].second_intermediate = 0;
char_set_table[TABLE_ASCII].final = 'B';

char_set_table[TABLE_LINE_DRAWING].name = LINE_DRAWING_G;
char_set_table[TABLE_LINE_DRAWING].first_intermediate = 0;
char_set_table[TABLE_LINE_DRAWING].second_intermediate = 0;
char_set_table[TABLE_LINE_DRAWING].final = '0';

if (conformance_level >= LEVEL_2)
  {
  char_set_table[TABLE_DEC_SUPP].name = DEC_SUPPLEMENTAL_G;
  char_set_table[TABLE_DEC_SUPP].first_intermediate = '%';
  char_set_table[TABLE_DEC_SUPP].second_intermediate = 0;
  char_set_table[TABLE_DEC_SUPP].final = '5';

  if (level_2_extensions[DRCS_EXT]==TRUE)
    {
    char_set_table[TABLE_DRCS].name = DRCS_G;
    char_set_table[TABLE_DRCS].first_intermediate = 0;
    char_set_table[TABLE_DRCS].second_intermediate = 0;
    char_set_table[TABLE_DRCS].final = 0;
    }
  if (level_2_extensions[NRCS_EXT]==TRUE)
    {
    /*
       This entry is initialized to correspond
       with the keyboard dialect whenever it
       is changed in Set-Up.
     */
    char_set_table[TABLE_NRCS].name = NRCS_G;
    char_set_table[TABLE_NRCS].first_intermediate = 0;
    char_set_table[TABLE_NRCS].second_intermediate = 0;
    char_set_table[TABLE_NRCS].final = 0;
    }
  }
if ((conformance_level >= LEVEL_3) ||
    (level_2_extensions[EIGHT_BIT_IA]==TRUE))
  {
  char_set_table[TABLE_LATIN1].name = ISO_LATIN1_SUPPLEMENTAL_G;
```

```
    char_set_table[TABLE_LATIN1].first_intermediate = 0;
    char_set_table[TABLE_LATIN1].second_intermediate = 0;
    char_set_table[TABLE_LATIN1].final = 'A';

    char_set_table[TABLE_UPSS].name = UPSS_G;
    char_set_table[TABLE_UPSS].first_intermediate = 0;
    char_set_table[TABLE_UPSS].second_intermediate = 0;
    char_set_table[TABLE_UPSS].final = '<';
    }
  if (level_3_extensions[TCS]==TRUE)
    {
    char_set_table[TABLE_TCS].name = TCS_G;
    char_set_table[TABLE_TCS].first_intermediate = 0;
    char_set_table[TABLE_TCS].second_intermediate = 0;
    char_set_table[TABLE_TCS].final = '>';
    }
  }
```

```
  void designate_character_set(parc, parv)
    /* parc and parv contain the count and values
       of the indermediate characters of the
       designating sequence */
    short int parc;
    short int parv[MAX_NUM_INTERMEDIATES];
  {
    short int n;
    short int gset_number;
    short int set_size;
    boolean_t compare_designator();  /* function declaration */
  /*.....................*/
  /* find which set we're designating */
  switch (parv[0])
    {
    case ')':  /* 2/8  designate 94 into G0 */
      gset_number = 0;
      set_size = 94;
      break;
    case '(':  /* 2/9  designate 94 into G1 */
      gset_number = 1;
      set_size = 94;
      break;
    case '*':  /* 2/10 designate 94 into G2 */
      gset_number = 2;
      set_size = 94;
      break;
    case '+':  /* 2/11 designate 94 into G3 */
      gset_number = 3;
      set_size = 94;
      break;
```

```
    case '-':  /* 2/13 designate 96 into G1 */
      gset_number = 1;
      set_size = 96;
      break;
    case '.':  /* 2/14 designate 96 into G2 */
      gset_number = 2;
      set_size = 96;
      break;
    case '/':  /* 2/15 designate 96 into G3 */
      gset_number = 3;
      set_size = 96;
      break;
    }

  /* look for the designator in the table */
  /*
     Note that because the DRCS designator is processed last,
     it will replace an installed set if the designator is
     identical.
  */
  for (n=0; n<MAX_NUM_CHAR_SETS; n++)
    {
    if ( compare_designator(parc, parv, n) )
      {
      designated_graphic_sets[gset_number] =
        char_set_table[n].name;
      }
    }

  /* update the in_use_table with
     the latest designations (if any)
     for the invoked gsets */
  in_use_table.gl =
    designated_graphic_sets[in_use_table.invoked_gl];
  if (conformance_level >= LEVEL_2)
    {
    in_use_table.gr =
      designated_graphic_sets[in_use_table.invoked_gr];
    }
  /* perform hardware specific actions needed
     to update the character set displayed */
  }  /* end procedure designate_character_set */


  /*
   * compare sequence designator to table entry n
   */
  boolean_t compare_designator(parc, parv, n)
    short int parc;  /* number of sequence characters */
    short int parv[MAX_NUM_INTERMEDIATES];
    short int n;     /* char set table entry */
  {
    boolean_t status;
```

```
status = TRUE;
if (char_set_table[n].final != parv[parc-1])
  status = FALSE;
else
  {
  if (parc > 2)
    {
    if (char_set_table[n].first_intermediate != parv[1])
      status = FALSE;
    else
      {
      if (parc >= 3)
        if (char_set_table[n].second_intermediate != parv[2])
          status = FALSE;
      }
    }
  }
return(status);
}
```

Known Deviations: None

ASSIGN USER-PREFERENCE SUPPLEMENTAL SET DECAUPSS

Levels: 2x (8-bit Interface Architecture), 3-4

Purpose: To assign a user preferred supplemental set

Format:

DCS	Ps	!	u	Dscs	ST
9/0		2/1	7/5		9/12

Description: This control string assigns a character set to be the User-Preference Supplemental Set (UPSS). The User-Preference Supplemental Set serves the following functions:

1. It acts as the default supplemental set for reception. On soft terminal reset, the UPSS is designated as G2.

2. It acts as the keyboard supplemental set, that is, it determines which supplemental characters can be generated from the keyboard (including valid compose sequences).

3. It provides a level of indirection for application software. The UPSS can be explicitly designated without needing to know the actual character set assigned.

Currently DEC Supplemental and ISO Latin-1 supplemental may be used as the UPSS. The default User-Preference Supplemental Set is determined through a Set-Up feature. The value of Ps and Dscs denote which set from the terminal's repertory of character sets is to be the User-Preference Supplemental Set.

Ps selects between 94 and 96 character sets (Ps = 0 selets a 94 character set). Dscs is the intermediate and final characters of the SCS escape sequence used to explicitly designated that character set. For example a value of "% 5" for Dscs indicates that the DEC Supplemental character set is the User-Preference Supplemental set. See the SCS command for more information on the designating intermediate and final characters. Attempts to assign an invalid supplemental set will be ignored.

Notes on DECAUPSS:

1. The DRCS can be used as the UPSS by naming it as DEC Supplemental or ISO Latin-1 supplemental.

2. Assigning a new UPSS will immediately update the In-Use table if the UPSS is currently designated and invoked.

3. This sequence is sent from the terminal to the host in response to a DECRQUPSS request.

Implementation Guideline

The valid combinations for the VT320 are:

Ps = 0, Dscs = % 5 DEC Supplemental
Ps = 1, Dscs = A ISO Latin-1 supplemental graphic

State Affected:

```
    upss    /* defined in code extension layer */
```

Algorithm:

```
void assign_user_preference_set(parv, string)
  short int parv[MAX_NUM_PARAMETERS];
  char string[];
{
if ((conformance_level >= LEVEL_3) ||
    (level_2_extensions[EIGHT_BIT_IA] == TRUE))
  {
  /* update upss state in Code Extension Layer */
  if ((parv[0] = 0) && (strcmp(string, "%5") == 0))
    {
    /* size is 94 characters */
    upss = DEC_SUPPLEMENTAL_G;
    }
  else if ((parv[0] = 1) && (strcmp(string, "A") == 0))
    {
    /* size is 96 characters */
    upss = ISO_LATIN1_SUPPLEMENTAL_G;
    }

  /* update in use table */
  if (in_use_table.gl == UPSS_G)
    {
    ; /* perform hardware dependent actions as needed */
    }
  if (in_use_table.gr == UPSS_G)
    {
    ; /* perform hardware dependent actions as needed */
    }
  }
}
```

Known Deviations: None

REQUEST USER-PREFERENCE SUPPLEMENTAL SET DECRQUPSS

Levels: 2x (8-bit Interface Architecture), 3-4

Purpose: To request the user preferred supplemental set

Format:

CSI	&	u
9/11	2/6	7/5

Description: Upon receipt of this sequence, the terminal will return a DECAUPSS sequence which specifies which coded character set is assigned to be the User-Preference Supplemental Set.

State Affected: none

Algorithm:

```
void request_user_preference_set()
{
if ((conformance_level >= LEVEL_3) ||
     (level_2_extensions[EIGHT_BIT_IA] == TRUE))
  {
  send_char(0x90, host_port);  /* DCS */
  if (upss = DEC_SUPPLEMENTAL_G)
    {
    send_int(0, host_port);
    send_string("!u%5", host_port);
    }
  else if (upss = ISO_LATIN1_SUPPLEMENTAL_G)
    {
    send_int(1, host_port);
    send_string("!uA", host_port);
    }
  send_char(0x9C, host_port);  /* ST */
  }
}
```

Known Deviations: None

SET/RESET CHARACTER SET MODE DECNRCM

Levels: 1x, 2x, 3x, 4x (NRCS Extension)

Purpose: To select 8-bit Multinational Characters, or 7-bit NRCS Characters for transmission (keyboard) and reception (display).

To set: CSI ? 42 h (7-bit NRCS Characters)
To reset: CSI ? 42 l (8-bit Multinational Characters)

Description: This parameter is applicaple to the Set Mode and Reset Mode control sequence. When 8-bit Multinational Characters is selected (factory default), the terminal uses ASCII in GL and either DEC Supplemental or ISO Latin-1 Supplemental in GR (depending on the UPSS). When 7-bit NRCS Characters is selected, a 7-bit National Replacement Character set is used for transmission and reception in GL (GR is not used). The NRCS used is determined by the Keyboard Dialect chosen in Set-Up (see "Keyboard Processing", DEC STD 70-6).

Notes on DECNRCM:

1. Changing this mode has the side effect of re-initializing the designated and invoked character sets to their default state (the state at power-on or reset). If 7-bit NRCS Characters is selected, a 7-bit NRC Set may replace U.S. ASCII in the default designations.

2. 8-bit Characters are temporarily disabled when in Level 1 (VT100 mode), or a 7-bit host line environment is selected. The DECNRCM sequence will still be recognized, but in the case of the reset sequence, 8-bit characters will remain disabled until the restrictive conditions are removed.

3. This sequence is ignored when the North American or Dutch Keyboard Dialect is selected. These keyboard dialects use U.S. ASCII as their 7-bit set, so there is no need to disable the multinational set (which includes U.S. ASCII as its left half) to access 7-bit NRC characters. U.S. ASCII may be considered the NRC set for North America.

State Affected:

```
character_set_mode
r_character_set_mode
   /* used to keep track of character set mode
      when a restrictive condition (7-bit Host Line,
```

```
            or VT100 mode) is in effect */
        in_use_table
        designated_graphic_sets[4]


Algorithm:

void set_character_set_mode()
{
if (level_1_extensions[NRCS_EXT]==TRUE)
  {
  character_set_mode = SEVEN_BIT_NATIONAL;
  /* keep track of mode if there are restrictions in effect */
  r_character_set_mode = SEVEN_BIT_NATIONAL;
  if (keyboard_usage_mode == TYPEWRITER)
    {
    in_use_table.gl = nrcs_list[keyboard_dialect];
    in_use_table.gr = nrcs_list[keyboard_dialect];
    in_use_table.invoked_gl = G0;
    in_use_table.invoked_gr = G2;
    designated_graphic_sets[0] = nrcs_list[keyboard_dialect];
    designated_graphic_sets[1] = nrcs_list[keyboard_dialect];
    designated_graphic_sets[2] = nrcs_list[keyboard_dialect];
    designated_graphic_sets[3] = nrcs_list[keyboard_dialect];
    }
  else    /* keyboard usage mode is Data Processing */
    {
    in_use_table.gl = ASCII_G;
    in_use_table.gr = ASCII_G;
    in_use_table.invoked_gl = G0;
    in_use_table.invoked_gr = G2;
    designated_graphic_sets[0] = ASCII_G;
    designated_graphic_sets[1] = ASCII_G;
    designated_graphic_sets[2] = ASCII_G;
    designated_graphic_sets[3] = ASCII_G;
    }
  }
}


void reset_character_set_mode()
{
if (level_1_extensions[NRCS_EXT]==TRUE)
  {
  /* keep track of mode if there are restrictions in effect */
  r_character_set_mode = EIGHT_BIT_MULTINATIONAL;
  /* ignore if 7-bit host line or VT100 mode */
  if ((host_port_environment != SEVEN_BIT) &&
      (conformance_level != LEVEL_1))
    {
    character_set_mode = EIGHT_BIT_MULTINATIONAL;
    in_use_table.gl = ASCII_G;
    in_use_table.gr = UPSS_G;
```

```
    in_use_table.invoked_gl = G0;
    in_use_table.invoked_gr = G2;
    designated_graphic_sets[0] = ASCII_G;
    designated_graphic_sets[1] = ASCII_G;
    designated_graphic_sets[2] = UPSS_G;
    designated_graphic_sets[3] = UPSS_G;
    }
  }
}
```

Known Deviations: None

5.9 SUMMARY OF CONTROL CHARACTER PROCESSING

This subsection provides a summary of control character processing as a convenience to the reader, and to include control characters which are not described in detail in their corresponding functional subsection of this chapter (DEC STD 70-5).

5.9.1 C0 Control Characters

C0 control characters are those with values of 0/0 to 1/15, and 7/15 (0 - 31, and 127 decimal). The following is a description of the control characters recognized by conforming devices. All other C0 control characters cause no action to be taken.

Control	Col/Row	Action Taken
NUL	0/0	Ignored on input (not stored in input buffer). May be used as a time fill character. Note: When Control Representation Mode is Set, NUL is not ignored, but displayed, and may not be used as a time fill character.
ENQ	0/5	Transmit answerback message.
BEL	0/7	Causes bell tone to sound from keyboard.
BS	0/8	Causes the cursor to move to the left one character position, unless it is at the left margin. Does not cause wrap.

HT	0/9	Causes the cursor to move to the next tab stop, or the right margin if no further tab stops. Does not cause wrap.
LF	0/10	Causes a line feed or a new line operation depending on the setting of Line Feed/New Line Mode (LNM). LF causes a line to be printed if in auto print mode.
VT	0/11	Interpreted as a LF.
FF	0/12	Interpreted as a LF.
CR	0/13	Cursor goes to left margin. If cursor is already left of the left margin, cursor goes to column 1.
SO (LS1)	0/14	Invoke G1 character set into GL.
SI (LS0)	0/15	Invoke G0 character set into GL.
XON	1/1	Causes terminal to resume transmitting, if XOFF handling is enabled.
XOFF	1/3	Causes terminal to stop transmitting all codes except XOFF and XON, if XOFF handling is enabled.
CAN	1/8	If received during an escape or control sequence, the sequence is immediately terminated and not executed. No character is displayed.
SUB	1/10	If received during an escape or control sequence, the sequence is immediately terminated and not executed. The error character (reverse question mark) is displayed.
ESC	1/11	Introduces an escape sequence. Terminates any escape, control sequence, or control string in progress.

Control	Col/Row	Action Taken
DEL	7/15	Ignored on input (not stored in input buffer). May not be used as a time fill character.

5.9.2 C1 Control Characters

C1 control characters are those with positions of 8/0 to 9/15. The following is a description of the C1 control characters which are recognized by conforming devices. All other C1 control characters cause no action to be taken.

Note C1 controls can be represented by ESC Fe sequences in 7-bits. See "Code Extension Layer" (DEC STD 70-3).

Control	Col/Row	Action Taken
IND	8/4	INDEX - Causes the active position to move downward one line without changing the horizontal position. If the active position is at the bottom margin, a scroll up is performed.
NEL	8/5	NEXT LINE - Causes the active position to move to the first position on the next line downward. If the active position is at the bottom margin, a scroll action is performed. (a CR LF is executed).
HTS	8/8	HORIZONTAL TABULATION SET Sets one horizontal tabulation stop at the column of the active position (See also, TBC).
RI	8/13	REVERSE INDEX - Moves the active position to the same horizontal position on the preceding line. If the active position is at the top margin, a scroll down is performed.

SS2	8/14	SINGLE SHIFT 2 - Causes the G2 set to be invoked for the next single displayed character.
SS3	8/15	SINGLE SHIFT 3 - Causes the G3 set to be invoked for the next single displayable character.
DCS	9/0	DEVICE CONTROL STRING - Introduces a Device Control String.
SOS	9/8	START OF STRING - This control is ignored.
DECID (reserved)	9/10	DEC PRIVATE IDENTIFICATION - Causes the same response as the ANSI device attributes (see DA). This sequence is not recognized a Level 3 or above. Conforming software shall use DA. Note: If printer controller mode is enabled, the sequence will be sent to the printer.
CSI	9/11	CONTROL SEQUENCE INTRODUCER - Introduces a control sequence.
ST	9/12	STRING TERMINATOR - Terminates Control Strings introduced by DCS, APC, OSC, and PM.
OSC	9/13	Operating System Command - A control string.
PM	9/14	Privacy Message - A control string.
APC	9/15	Application Program Command - A control string.

SUBSTITUTE SUB

Levels: 1-4

Purpose: Place a reversed question mark symbol in the display.

Format: SUB
1/10

Description: The SUB control is used to indicate the presence of characters in the data stream for which there is no graphic representation. This control is processed by placing a reversed question mark in the display at the Active Position.

Notes:

1. The character to be displayed in cases where the reversed question mark cannot be used is a "half-tone blotch", which is rendered by filling the entire character cell in the display with a cross-hatched pattern.

2. Receipt of the Substitute character within an escape or control sequence causes the sequence in progress to be aborted and the Substitute control function to be executed instead (that is, the reversed question mark symbol will be displayed). (See the section "Code Extension Layer" for a complete description of the handling of control characters within escape and control sequences.)

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void substitute()
{
short int sub = 0x1A;
/*.........*/

/* place a reversed question mark in the display */
insert_or_replace_character(&sub);
}
```

Known Deviations:

Most existing terminals (VT100, VT101, VT102, VT125, VT131, VT132) display the blotch character instead of the reversed question mark.

5.10 MODES STATES (AND MODE DESCRIPTIONS)

A conforming device shall provide certain programmable state capabilities or modes as described below. It should be noted that some modes are identified as User Preference Features (see the section "Concepts and Conformance Criteria" for a list of the modes which are classified as User Preference Features). The state of these features should not be changed by conforming software, except in response to an explicit user request (for example, a request to change the screen width).

5.10.1 SM/RM - Set Mode, Reset Mode Sequence

The Set Mode/Reset Mode control sequence is used to control the state of various settable modes in the terminal.

Set Format:	CSI	Ps...Ps	l
	9/11	Ps...Ps	6/12
Reset Format:	CSI	Ps...Ps	h
	9/11	Ps...Ps	6/8

Sets/Resets one or more modes of the terminal as specified by each selective parameter in the parameter string. Each mode to be set/reset is specified by a separate parameter.

Note that ANSI specified mode parameters and DEC private mode parameters may not be mixed within the same SM or RM sequence.

5.10.1.1 ANSI Specified Modes: -

Mode Parameter	Mnemonic	Mode Function
0		Error (ignored)
2	KAM	Keyboard Action Mode
4	IRM	Insert Replacement Mode
12	SRM	Send-Receive Mode
20	LNM	Line Feed New Line Mode

5.10.1.2 DEC Private Modes: -

If the first character in the parameter string is 3/15 (?), the parameters are interpreted as DEC private parameters, according to the following table.

Mode Parameter	Mnemonic	Mode Function
0		Error (ignored)
1	DECCKM	Cursor Key Mode
3	DECCOLM	Column Mode
4	DECSCLM	Scrolling Mode
5	DECSCNM	Screen Mode
6	DECOM	Origin Mode
7	DECAWM	Autowrap Mode
8	DECARM	Autorepeating Mode
18	DECPFF	Print Form Feed
19	DECPEX	Print Extent
25	DECCEM	Cursor Enable Mode
38	DECTEK	Tektronix 4010/4014 Mode
42	DECNRCM	National Character set mode
61	DECVCCM	Vertical Coupling
64	DECPCCM	Page Coupling
68	DECKBUM	Keyboard Usage Mode
69	DECLRMM	Left Right Margin Mode
73	DECXRLM	Transmit Rate Limiting

Any other parameter values are ignored.

5.10.1.3 Modes That Cannot Be Changed - The following modes which are specified in the ANSI X3.64-1979 standard may be considered to be permanently set, permanently reset, or not applicable, as noted. Refer to that standard for further information concerning these modes.

Mode Mnemonic	Mode Function	State
EBM	Editing Boundary Mode	Reset
FEAM	Format Effector Action Mode	Reset
FETM	Format Effector Transfer Mode	NA
HEM	Horizontal Editing Mode	Reset
MATM	Multiple Area Transfer Mode	NA
PUM	Positioning Unit Mode	Reset
SATM	Selected Area Transfer Mode	NA
SRTM	Status Reporting Transfer Mode	Reset
TSM	Tabulation Stop Mode	Reset
VEM	Vertical Editing Mode	Reset

5.10.2 Mode Descriptions

The detailed descriptions for many of the modes that use the SM/RM sequences are included in other parts of this standard (DEC STD

70) to keep related functions together.

SET/RESET SEND RECEIVE MODE SRM

Levels: 1-4

Purpose: To turn the local echo feature ON or OFF

Set Format: CSI 12 h (local echo off, default)
Reset Format: CSI 12 l (local echo on)

Description: SRM is a parameter applicable to the Set Mode and Reset Mode control sequences. The reset state causes local echo to be ON. The set state causes local echo to be OFF.

NOTE

Local echo is temporarily disabled when either or both of the following conditions are true:
- The terminal is in printer controller mode
- The terminal is in local mode

State Affected:

```
send_receive_mode   /* See Keyboard Processing */
r_send_receive_mode
```

Algorithm:

```
void set_send_receive_mode()
{
send_receive_mode = ECHO_OFF;
/* remember state in case there are restrictions */
r_send_receive_mode = ECHO_OFF;
}


void reset_send_receive_mode()
{
/* remember state in case there are restrictions */
r_send_receive_mode = ECHO_ON;
if ((printer_controller_mode != PRINTER_CONTROLLER_ON) &&
    (line_local_mode != LOCAL))
  send_receive_mode = ECHO_ON;
}
```

Known Deviations: None

5.11 EDITING FUNCTIONS

This subsection describes the following functions used to edit data, or control editing in the Logical Display.

- IRM - Insert/Replacement Mode
- ICH - Insert Character
- DCH - Delete Character
- IL - Insert Line
- DL - Delete Line
- DECIC - Insert Column
- DECDC - Delete Column
- ECH - Erase Character
- EL - Erase in Line
- DECSEL - Selective Erase in Line
- ED - Erase in Display
- DECSED - Selective Erase in Display
- Character Attributes (selective erase)
  - DECSCA - Select Character Attribute

SET/RESET INSERT/REPLACEMENT MODE IRM

Levels: 1X, 2-4

Purpose: Change the state of Insert/Replacement Mode between Replace (reset) and Insert (set) graphic characters.

Set Format: CSI 4 h
Reset Format: CSI 4 l

Description: A conforming device shall provide the ability to select whether characters entered into character positions which already contain data shall overwrite the character at that position [reset state = replace], or shall be inserted by causing all data on the line starting at the Active Position to be shifted one column to the right to accommodate the new character [set state = insert].

Notes:

1. The IRM controls are part of the Level 1 Editing Extension. This extension is required in all new Level 1 implementations. They are required functions in Level 2.

State Affected:

insert_replacement_mode

Algorithm:

```
void set_insert_replacement_mode()
{
insert_replacement_mode = INSERT;
}


void reset_insert_replacement_mode()
{
insert_replacement_mode = REPLACE;
}
```

Known Deviations: None

INSERT OR REPLACE GRAPHIC CHARACTER

Levels: 1-4

Purpose: To place a graphic character in the display at the Active Position.

Format: implied on receipt of graphic character data

Description: This procedure is called on receipt of graphic character data to place the data in the appropriate position in the display structure. The character code, as well as the values of the Current Rendition, Current Attribute, and the character set indicated by the In Use Table (left side if code <= 7/14, otherwise right side) are stored in the display list.

The Active Position is advanced by one column each time this operation is performed until it reaches the Right Margin, or the last column of the Active Line. If the Active Position is within the Scroll Area, it will advance to the Right Margin. If the Active Position is outside the Scroll Area, it will advance to the last column of the Active Line.

If Auto Wrap mode is off, the Active Position will not advance beyond the Right Margin or last column of the Active Line. Additional graphic characters received will overwrite this position.

If Auto Wrap Mode is on and the Active Position is within the Vertical Scroll Margins, the Active Position will advance from the Right Margin or last column of the Active Line, to the Left Margin of the next line (causing the display to scroll if necessary) when a graphic character is received and the Last Column Flag is set (see Auto Wrap Mode, DEC STD 70-D for details).

If Auto Wrap Mode is on and the Active Position is outside the Vertical Scroll Margins, the Active Position will advance from the last column of the Active Line, to the first column of the next line when a graphic character is received and the Last Column Flag is set (see Auto Wrap Mode, DEC STD 70-D for details). No scrolling will occur.

Notes:

1. This operation is affected by the setting of Insert/Replacement Mode. When this mode is in the reset state (Replace) the new data will overwrite existing data at the Active Position. When this mode is in the set state (Insert) the contents of the active line will be shifted right one column starting at the Active Position to accommodate the new data. Data shifted past the Right

Margin within the Scroll Area, or the last column of the Active Line outside the Scroll Area is lost.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
active_position
```

Algorithm:

```
void insert_or_replace_character(parv)
  short int parv[];
{
  column_t x;
  short int line;    /* actual line in logical display */
  short int margin;  /* effective right boundary */
line = (active_position.page * lines_per_page) +
        active_position.line;
if (insert_replacement_mode == INSERT)
  {
  /* determine right boundary for shifting characters */
  if ( (active_position.column <= right_margin) &&
       (active_position.line >= bottom_margin) &&
       (active_position.line <= top_margin)    &&
       (right_margin < end_of_line(line))    )
    margin = right_margin;
  else
    margin = end_of_line(line);
  /* make room for character to be inserted */
  for ( x=margin;
        x > active_position.column;
        x-- )
    strncpy( display[line][x], display[line][x-1],
             sizeof(character_t) );
  }
/* put code in the display, strip 8th bit */
display[line][active_position.column].code = parv[0] & 0x7F;

display[line][active_position.column].rendition =
current_rendition;
display[line][active_position.column].attribute =
current_attribute;

/* write the character set */
if (single_shift == NO_SINGLE_SHIFT)
  {
  if (parv[0] <= 126)
    display[line][active_position.column].character_set =
      in_use_table.gl;
  else
```

```
      display[line][active_position.column].character_set =
        in_use_table.gr;
    }
  else if (single_shift == SS2)
    display[line][active_position.column].character_set =
      designated_graphic_sets[2];
  else if (single_shift == SS3)
    display[line][active_position.column].character_set =
      designated_graphic_sets[3];

  /* advance the active position after entering character */
| if (active_position.column < margin)
|   active_position.column += 1;
| else  /* active position is at effective margin */
|   if (auto_wrap_mode = WRAP_ON)
|     {
|     if (last_column_flag)
|       {
|       carriage_return();
|       line_feed();
|       last_column_flag = FALSE;
|       }
|     else
|       last_column_flag = TRUE;
|     }
| }   /* end insert_or_replace_character() */
```

Known Deviations: None

INSERT CHARACTER ICH

Levels: 2-4

Purpose: Insert empty characters at the Active Position.

Format:	CSI	Pn	@	default Pn: 1
	9/11	Pn	4/0	

Description: The ICH control causes empty characters to be inserted at the Active Position. A parameter value of zero or one causes a single empty character to be inserted. A parameter value of n causes n empty characters to be inserted. Data on the Active Line from the Active Column to the Right Margin is shifted forward as in character insertion. Characters shifted beyond the Right Margin are lost.

Notes:

1. The extent of the display affected by execution of this control is limited to the Active Line within the left and right margins. Data which is shifted beyond the Right Margin is lost.

2. If the active position is outside the left or right margin, this control is ignored.

3. The Active Position is not affected by execution of this control.

4. Execution of this control is not affected by the setting of Insert/Replacement Mode.

State Affected:

display[MAX_NUM_LINES][MAX_NUM_COLUMNS]

Algorithm:

```
void insert_character(parv)
  int parv[MAX_NUM_PARAMETERS];
{
  column_t x;
  short int line;     /* actual line in logical display */
  short int margin;   /* effective right boundary */
  short int n;
/*.....................*/

/* ignore if we're outside the scrolling area */
if ( (active_position.line >= bottom_margin) &&
     (active_position.line <= top_margin)    &&
     (active_position.column >= left_margin) &&
     (active_position.column <= right_margin)  )
  {
  line = (active_position.page * lines_per_page) +
          active_position.line;
  /* determine right boundary for shifting characters */
  if (right_margin < end_of_line(line))
    margin = right_margin;
  else
    margin = end_of_line(line);

  if (parv[0] == 0) n = 1;
  else n = parv[0];
  if ((active_position.column + n) > margin)
    n = margin - active_position.column + 1;

  /* make room for characters to be inserted */
  for ( x=margin;
        x > active_position.column + n;
        x-- )
    strncpy( display[line][x], display[line][x-n],
             sizeof(character_t) );
  for (x=1; x<=n; x++)
    {
    display[line][active_position.column+(x-1)].code =
      EMPTY_CHARACTER;
    }
  }
}
```

Known Deviations: None

DELETE CHARACTER DCH

Levels: 1X, 2-4

Purpose: Delete characters at the Active Position.

Format:	CSI	Pn	P	default Pn: 1
	9/11	Pn	5/0	

Description: The DCH control causes characters to be deleted at the Active Position. A parameter value of zero or one causes a single character to be deleted. A parameter value of n causes n characters to be deleted. Data on the Active Line between the Active Column and Right Margin is shifted left to close up the deleted character positions, and empty characters are inserted at the Right Margin to accommodate the shift.

Notes:

1. The DCH control is part of the Level 1 Editing Extension. This extension is required in all new Level 1 implementations. It is a required function in Level 2.

2. The extent of the display affected by execution of this control is limited to the Active Line within the left and right margins.

3. If the Active Position is outside the left or right margins, this control is ignored.

4. The Active Position is not affected by execution of this control.

5. Execution of this control is not affected by the setting of Insert/Replacement Mode.

State Affected:

display[MAX_NUM_LINES][MAX_NUM_COLUMNS]

Algorithm:

```
void delete_character(parv)
  int parv[MAX_NUM_PARAMETERS];
{
  column_t x;
  short int line;    /* actual line in logical display */
  short int margin;  /* effective right boundary */
  short int n;
/*.....................*/

/* ignore if we're outside left or right margin */
if ( (active_position.column >= left_margin) &&
     (active_position.column <= right_margin)  )
  {
  line = (active_position.page * lines_per_page) +
          active_position.line;
  /* determine right boundary for shifting characters */
  if (right_margin < end_of_line(line))
    margin = right_margin;
  else
    margin = end_of_line(line);

  if (parv[0] == 0) n = 1;
  else n = parv[0];
  if ((active_position.column + n) > margin)
    n = margin - active_position.column + 1;

  /* make delete characters */
  for ( x=active_position.column;
        x <= margin-n;
        x++ )
    strncpy( display[line][x], display[line][x+n],
             sizeof(character_t) );
  for (x=1; x<=n; x++)
    {
    display[line][margin-(x-1)].code =
      EMPTY_CHARACTER;
    }
  }  /* end active position within the left and right margins */
} /* end delete_character() */
```

Known Deviations: None

INSERT LINE IL

Levels: 1x, 2-4

Purpose: Insert lines of empty characters at the Active Position.

Format:				
	CSI	Pn	L	default Pn: 1
	9/11	Pn	4/12	

Description: The IL control causes lines of empty characters to be inserted at the Active Position. A parameter value of zero or one causes a single line to be inserted. A parameter value of n causes n lines to be inserted. Data in the display is scrolled downward to the Bottom Margin from the active line to accommodate the insertion. Only that portion of the display between the top, bottom, left, and right margins is affected. IL is ignored if the Active Position is outside the Scroll Area.

Notes:

1. The IL control is part of the Level 1 Editing Extension. This extension is required in all new Level 1 implementations. It is a required function in Level 2.

2. The Active Position is set to the Left Margin in the active line after execution of this control. The active line does not change.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
line_rendition[MAX_NUM_LINES]
active_position
```

Algorithm:

```
void insert_line(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  short int n;
/*.....................*/
/* ignore if we're outside the scrolling area */
if ( (active_position.line >= bottom_margin) &&
     (active_position.line <= top_margin)    &&
     (active_position.column >= left_margin) &&
     (active_position.column <= right_margin)  )
  {
  if (parv[0] == 0) n = 1;
  else n = parv[0];
  if ((active_position.line + n) > bottom_margin)
    n = (bottom_margin - active_position.line) + 1;
  scroll_down(active_position.line, n);
  active_position.column = left_margin;
  }
}
```

Known Deviations: None

DELETE LINE DL

Levels: 1x, 2-4

Purpose: Delete lines at the Active Position.

Format:				
	CSI	Pn	M	default Pn: 1
	9/11	Pn	4/13	

Description: The DL control causes lines of characters to be deleted at the Active Position. A parameter value of zero or one causes a single line to be deleted. A parameter value of n causes n lines to be deleted. Data in the display is scrolled upward from the Bottom Margin to the active line to accommodate the deletion. Only that portion of the display between the top, bottom, left, and right margins is affected. DL is ignored if the active position is outside the scroll area.

Notes:

1. The DL control is part of the Level 1 Editing Extension. This extension is required in all new Level 1 implementations. It is a required function in Level 2.

2. The Active Position is set to the Left Margin in the active line after execution of this control. The active line does not change.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
line_rendition[MAX_NUM_LINES]
active_position
```

Algorithm:

```
void delete_line(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  short int n;
/*......................*/
/* ignore if we're outside the scrolling area */
if ( (active_position.line >= bottom_margin) &&
     (active_position.line <= top_margin)    &&
     (active_position.column >= left_margin) &&
     (active_position.column <= right_margin)  )
  {
  if (parv[0] == 0) n = 1;
  else n = parv[0];
  if ((active_position.line + n) > bottom_margin)
    n = (bottom_margin - active_position.line) + 1;
  scroll_up(active_position.line, n);
  active_position.column = left_margin;
  }
}
```

Known Deviations: None

INSERT COLUMN DECIC

Level: 4x (Horizontal Scrolling)

Purpose: Insert columns of empty characters at the Active Position.

Format:

	CSI	Pn	'	}	default Pn: 1
	9/11	Pn	2/7	7/13	

Description: The DECIC control causes Pn columns to be inserted at the active column position. The contents of the display are shifted to the right from the active column to the right margin. The inserted columns are set to blank with normal rendition. Only that portion of the display between the top, bottom, left, and right margins is affected. DECIC is ignored if the active position is outside the Scroll Area.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
```

Algorithm:

```
/* routine insert_column(parv) to be supplied */
```

Known Deviations: None

DELETE COLUMN DECDC

Level: 4x (Horizontal Scrolling)

Purpose: Delete columns at the Active Position

Format:

CSI	Pn	'	~	default Pn: 1	
9/11	Pn	2/7	7/14		

Description: The DECDC control causes Pn columns to be deleted at the active column position. The contents of the display are shifted to the left from the right margin to the active column. Columns containing blank characters with normal rendition are shifted into the display from the right margin. Only that portion of the display between the top, bottom, left, and right margins is affected. DECDC is ignored if the active position is outside the Scroll Area.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
```

Algorithm:

```
/* routine delete_column(parv) to be supplied */
```

Known Deviations: None

ERASE CHARACTER ECH

Levels: 2-4

Purpose: Erase characters at the Active Position.

Format:	CSI	Pn	X	default Pn: 1
	9/11	Pn	5/8	

Description: The ECH control causes the character at the Active Position and the next n-1 characters to be erased and their attributes set to normal. A parameter value of zero causes a single character to be erased. No reformatting of data on the line occurs, and the Active Position does not move.

Notes:

1. The extent of the display affected by execution of this control is limited to the Active Line. If a parameter value is given which is greater than the distance from the Active Position to the end of the Active Line, only character positions up to the end of the Active Line will be erased. This control is not affected by the margins.

2. Execution of this control is not affected by the setting of Insert/Replacement Mode.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
```

Algorithm:

```
void erase_character(parv)
  short int parv[MAX_NUM_PARAMETERS];
{
  column_t x;
  short int line;  /* actual line in logical display */
  short int n;
/*.........*/
line = active_position.page * lines_per_page +
       active_position.line;
if (parv[0] == 0) n = 1;
else n = parv[0];
if ((active_position.line + n) > end_of_line(line))
  n = (end_of_line(line) - active_position.column) + 1;
for (x=active_position.column;
     x <= active_position.column + (n-1);
     x++)
  {
  display[line][x].code = EMPTY_CHARACTER;
  }
}
```

Known Deviations: None

ERASE IN LINE EL

Levels: 1-4

Purpose: Erase character positions within the Active Line.

Format:	CSI	Ps	K	default Ps: 0
	9/11	Ps	4/11	

Description: The EL control causes a set of adjacent character positions in the Active Line to be set to the empty state, that is the empty character (imaged as SPACE), the empty rendition, and the empty character attribute. The set of character positions affected by this control is determined by the parameter value. An omitted parameter or a value of zero (0) cause the control to affect character positions from the Active Column to the end of the Active Line (inclusive). A parameter value of one (1) causes the control to affect character positions from the beginning of the Active Line to the Active Column (inclusive). A parameter value of two (2) causes the control to affect all character positions in the Active Line (inclusive).

Notes:

1. The line renditon is not affected by execution of this control.

2. The syntax of the EL control is identical to the Selective Erase in Line (DECSEL) control, except that DECSEL includes a private parameter specifier ? (3/15).

3. This control is not affected by the margins.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
line_rendition[MAX_NUM_LINES]
```

Algorithm:

```
/*
 * erase_in_line
 *   both standard and selective_erase
 */
void erase_in_line(parc, parv, private)
  short int parc;
  short int parv[MAX_NUM_PARAMETERS];
  boolean_t private;
{
  column_t x;
  short int line;  /* actual line in logical display */
  short int n;
/*................*/
if (private) selective_erase_in_line(parc, parv);
else
  {
  line = active_position.page * lines_per_page +
         active_position.line;
  if (parc == 0) parc = 1;
  for (n=0; n<parc; n++)
    {
    switch(parv[n])
      {
      case 0:
        for (x=active_position.column;
             x <= end_of_line(line); x++)
          display[line][x].code = EMPTY_CHARACTER;
        break;
      case 1:
        for (x=1; x<=active_position.column; x++)
          display[line][x].code = EMPTY_CHARACTER;
        break;
      case 2:
        for (x=1; x<=end_of_line(line); x++)
          display[line][x].code = EMPTY_CHARACTER;
        break;
      }
    }
  }
}    /* end erase_in_line() */
```

Known Deviations: None

ERASE IN DISPLAY ED

Levels: 1-4

Purpose: Erase character positions within the display.

Format:	CSI	Ps	J	default Ps: 0
	9/11	Ps	4/10	

Description: The ED control causes a set of adjacent character positions (extending across line boundaries) in the display to be set to the empty state. That is the empty character (imaged as SPACE), the empty rendition, and the empty character attribute. The set of character positions affected by this control is determined by the parameter value. An omitted parameter or a value of zero (0) cause the control to affect character positions from the Active Position to the end of the bottom line in the display (inclusive). A parameter value of one (1) causes the control to affect character positions from the beginning of the first line of the display to the Active Position (inclusive). A parameter value of two (2) causes the control to affect all character positions in the display (inclusive).

Notes:

1. The Line Renditions of all lines completely erased (all character positions in the line set to the erased state, that is, empty characters) as a result of execution of this control are set to single-width.

2. This control affects the entire display and is not affected by the setting of the scrolling region or the setting of Origin Mode.

3. The syntax of the ED control is identical to the Selective Erase in Display (DECSED) control, except that DECSED includes a private parameter specifier ? (3/15).

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
line_rendition[MAX_NUM_LINES]
```

Algorithm:

```
/*
 * erase_in_display
 *   both standard and selective_erase
 */
void erase_in_display(parc, parv, private)
  short int parc;
  short int parv[MAX_NUM_PARAMETERS];
  boolean_t private;
{
  line_t y;
  column_t x;
  short int base;  /* first line(-1) of current page */
  short int line;  /* actual line in logical display */
  short int n;
/*................*/
if (private) selective_erase_in_display(parc, parv);
else
  {
  base = (active_position.page-1) * lines_per_page;
  line = base + active_position.line;
  if (parc == 0) parc = 1;
  for (n=0; n<parc; n++)
    {
    switch(parv[n])
      {
      case 0:  /* erase to end of display */
        for (x=active_position.column;
             x <= end_of_line(line); x++)
          display[line][x].code = EMPTY_CHARACTER;
        if (active_position.column = 1)
          line_rendition[line] = SINGLE_WIDTH;
        for (y=line+1; y<(base+lines_per_page); y++)
          {
          for (x=1; x<=end_of_line(y); x++)
            {
            display[y][x].code = EMPTY_CHARACTER;
            line_rendition[y] = SINGLE_WIDTH;
            }
          }
        break;
      case 1:  /* erase from beginning of display */
        for (y=1; y<(base+line); y++)
          {
          for (x=1; x<=end_of_line(y); x++)
            {
            display[y][x].code = EMPTY_CHARACTER;
            line_rendition[y] = SINGLE_WIDTH;
            }
          }
        for (x=1; x<=active_position.column; x++)
          display[line][x].code = EMPTY_CHARACTER;
```

```
        if (active_position.column = end_of_line(line))
          line_rendition[line] = SINGLE_WIDTH;
        break;
      case 2:  /* erase entire display */
        for (y=1; y<(base+lines_per_page); y++)
          {
          for (x=1; x<=end_of_line(y); x++)
            {
            display[y][x].code = EMPTY_CHARACTER;
            line_rendition[y] = SINGLE_WIDTH;
            }
          }
        break;
      }
    }
  }  /* end not private */
}    /* end erase_in_display() */
```

Known Deviations: None

SELECTIVE ERASE IN LINE DECSEL

Levels: 2x, 3x, 4x (Selective Erase)

Purpose: Change character positions within the Active Line which have the Selectively Erasable attribute to the Space character (2/0).

Format:	CSI	?	Ps	K	default Ps: 0
	9/11	3/15	Ps	4/11	

Description: The DECSEL control causes a set of adjacent character positions in the Active Line which have the attribute "Selectively Erasable" to be changed to the Space character (2/0). The character rendition and attributes associated with each position do not change. Characters which have been entered into the display with the Selectively Erasable attribute reset (Not Selectively Erasable), as well as empty character positions, are not affected by this control, and no reformatting of the data will occur.

The set of character positions affected by this control is determined by the parameter value. An omitted parameter or a value of zero (0) cause the control to affect character positions from the Active Column to the end of the Active Line (inclusive). A parameter value of one (1) causes the control to affect character positions from the beginning of the Active Line to the Active Column (inclusive). A parameter value of two (2) causes the control to affect all character positions in the Active Line (inclusive).

Notes:

1. Use of this control permits efficient "clearing" of parts of the screen while leaving other parts alone, such as in form filling applications. (See Select Character Attribute, DECSCA, for a description of how to invoke the selectively erasable character attribute). This capability is an extension to Level 2 and so is not available in all Level 2 terminals.

2. This control is only executed if the Level 2 extension Selectively Erasable Characters is supported. In Level 1 operation and in other Level 2 devices it will be ignored.

3. The Line Rendition is not affected by execution of this control.

4. The syntax of the DECSEL control is identical to the Erase in Line (EL) control, except that DECSEL includes a private parameter specifier ? (3/15).

5. This control is not affected by the margins.

State Affected:

```
display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
line_rendition[MAX_NUM_LINES]
```

Algorithm:

```
void selective_erase_in_line(parc, parv)
  short int parc;
  short int parv[MAX_NUM_PARAMETERS];
{
  column_t x;
  short int line;  /* actual line in logical display */
  short int n;
/*...............*/
line = active_position.page * lines_per_page +
       active_position.line;
if (parc == 0) parc = 1;
for (n=0; n<parc; n++)
  {
  switch(parv[n])
    {
    case 0:
      for (x=active_position.column;
           x <= end_of_line(line); x++)
        {
        if (display[line][x].attribute.selective_erase)
          display[line][x].code = 32;  /* Space */
        }
      break;
    case 1:
      for (x=1; x<=active_position.column; x++)
        {
        if (display[line][x].attribute.selective_erase)
          display[line][x].code = 32;  /* Space */
        }
      break;
    case 2:
      for (x=1; x<=end_of_line(line); x++)
        {
        if (display[line][x].attribute.selective_erase)
          display[line][x].code = 32;  /* Space */
        }
      break;
    }
  }
```

```
}     /* end selective erase_in_line() */
```

Known Deviations: None

SELECTIVE ERASE IN DISPLAY — DECSED

Levels: 2x, 3x, 4x (Selective Erase)

Purpose: Change character positions within the display which have the Selectively Erasable attribute to the Space character (2/0).

Format:	CSI	?	Ps	J	default Ps: 0
	9/11	3/15	Ps	4/10	

Description: The DECSED control causes a set of adjacent character positions in the display which have the attribute "Selectively Erasable" to be changed to the Space character (2/0). The character rendition and attributes associated with each position do not change. Characters which have been entered into the display with the Selectively Erasable attribute reset (Not Selectively Erasable), as well as empty character positions, are not affected by this control, and no reformatting of the data will occur.

The set of character positions affected by this control is determined by the parameter value. An omitted parameter or a value of zero (0) cause the control to affect character positions from the Active Position to the end of the bottom line in the display (inclusive). A parameter value of one (1) causes the control to affect character positions from the beginning of the first line of the display to the Active Position (inclusive). A parameter value of two (2) causes the control to affect all character positions in the display (inclusive).

Notes:

1. Use of this control permits efficient "clearing" of parts of the display while leaving other parts alone, such as in form filling applications. (See Select Character Attribute, DECSCA, for a description of how to invoke the selectively erasable character attribute).

2. This control is only executed if the Level 2 extension Selectively Erasable Characters is supported. In Level 1 operation and in other Level 2 devices it will be ignored.

3. The Line Renditions of all lines are not affected by execution of this control.

4. This control affects the entire display and is not affected by the setting of the scrolling region or the setting of Origin Mode.

5. The syntax of the DECSED control is identical to the Erase in Display (ED) control, except that DECSED includes a private parameter specifier ? (3/15).

State Affected:

display[MAX_NUM_LINES][MAX_NUM_COLUMNS]
line_rendition[MAX_NUM_LINES]

Algorithm:

```
void selective_erase_in_display(parc, parv)
  short int parc;
  short int parv[MAX_NUM_PARAMETERS];
{
  line_t y;
  column_t x;
  short int base;  /* first line(-1) of current page */
  short int line;  /* actual line in logical display */
  short int n;
/*................*/
base = (active_position.page-1) * lines_per_page;
line = base + active_position.line;
if (parc == 0) parc = 1;
for (n=0; n<parc; n++)
  {
  switch(parv[n])
    {
    case 0:  /* erase to end of display */
      for (x=active_position.column;
           x <= end_of_line(line); x++)
        {
        if (display[line][x].attribute.selective_erase)
          display[line][x].code = 32;  /* Space */
        }
      if (active_position.column = 1)
        line_rendition[line] = SINGLE_WIDTH;
      for (y=line+1; y<(base+lines_per_page); y++)
        {
        for (x=1; x<=end_of_line(y); x++)
          {
          if (display[y][x].attribute.selective_erase)
            display[y][x].code = 32;  /* Space */
          }
        }
      break;
    case 1:  /* erase from beginning of display */
      for (y=1; y<(base+line); y++)
```

```
        {
        for (x=1; x<=end_of_line(y); x++)
          {
          if (display[y][x].attribute.selective_erase)
            display[y][x].code = 32;  /* Space */
          }
        }
      for (x=1; x<=active_position.column; x++)
        {
        if (display[line][x].attribute.selective_erase)
          display[line][x].code = 32;  /* Space */
        }
      if (active_position.column = end_of_line(line))
        line_rendition[line] = SINGLE_WIDTH;
      break;
    case 2:  /* erase entire display */
      for (y=1; y<(base+lines_per_page); y++)
        {
        for (x=1; x<=end_of_line(y); x++)
          {
          if (display[y][x].attribute.selective_erase)
            display[y][x].code = 32;  /* Space */
          }
        }
      break;
    }
  }
}    /* end selective_erase_in_display() */
```

Known Deviations: None

5.11.1 Character Attributes

Each character cell in the display may also have logical (non-visual) attribute associated with it, which affect its characteristics when processed or retransmitted by the terminal. These attributes can be combined in any manner for each character position independently. The architecture requires that each character attribute and every character attribute combination to have effect individually and independently from every other character attribute and character rendition (see previous section). Character attributes apply to all graphic characters codes (2/1 to 7/14 inclusive, and 10/1 to 15/14 inclusive) as well as to the Space character (2/0).

5.11.1.1 Normal Character Attribute - The Normal character attribute defines the non-visible attributes in the physical display in its "natural" state, that is, where the character behaves with no special non-visible attributes.

5.11.1.2 Selectively Erasable Character Attribute - One logical attribute may be applied to characters as a Level 2 or higher extension. This attribute is referred to as "Selectively Erasable Character". When this extension is provided, the terminal will recognize two forms of the commands Erase_In_Line and Erase_In_Display: one in which all characters in the given extent are erased, regardless of their logical attributes, and another in which all character in the given extent which have been assigned the Selectively Erasable Character attribute are turned into Space characters (2/0). (See the Select Character Attribute, Erase In Line, and Erase In Display control functions for a complete description of this feature).

SELECT CHARACTER ATTRIBUTE DECSCA

Levels: 2x, 3x, 4x (Selective Erase)

Purpose: Designate the character attribute(s) to be applied to all subsequent characters entered into the display.

Format:

CSI	Ps ; ... ; Ps	"	q	default Ps: 0
9/11	Ps ; ... ; Ps	2/2	7/1	

Description: The DECSCA control selects the character attribute (logical attribute) to be applied to all subsequent characters entered into the display. The parameter values indicate the combination of attributes to be selected as indicated in the following table:

Parameter	Rendition
0	Attributes off
1	Not Selectively Erasable (attribute on)
2	Selectively Erasable

Notes:

1. Character attribute parameter values are assigned in pairs. The lower (odd) value sets the attribute to the on state and the higher (even) value resets the attribute to the off state (same as normal or when all attributes are turned off (value 0).

2. All other parameter values will be ignored.

3. Attributes may be selected in any combination, and will be rendered appropriately. Selecting additional attributes does not affect the setting of previously selected attributes (newly selected attributes will be combined with previously selected attributes) unless all attributes are turned off (parameter value zero (0)).

4. In no way does the selection of attributes affect the rendition (SGR) or attributes (DECSCA) of characters already entered into the display.

5. This control is ignored unless the Level 2 or higher extension for Selectively Erasable Characters is supported. The Selectively Erasable Character attribute affects the execution of the controls Selective Erase In Line (DECSEL) and Selective Erase In Display (DECSED).

State Affected:

display[MAX_NUM_LINES][MAX_NUM_COLUMNS]

Algorithm:

```
void select_character_attribute(parc, parv)
  short int parc;
  short int parv[MAX_NUM_PARAMETERS];
{
  short int n;
/*................*/
if (conformance_level >= 2)
  {
  if (parc == 0) parc = 1;
  for (n=0; n<parc; n++)
    {
    switch(parv[n])
      {
      case 0:
        current_attribute.selective_erase = TRUE;
        break;
      case 1:
        current_attribute.selective_erase = FALSE;
        break;
      case 2:
        current_attribute.selective_erase = TRUE;
        break;
      }
    }
  }
} /* end select_character_attribute() */
```

Known Deviations: None

5.12 OLTP FEATURES

This subsection describes features defined at Level 4 to improve performance for On Line Transaction Processing (OLTP). These include Rectangular Area Operations, Data Integrity Checking, and Text Macros.

5.12.1 Rectangular Area Operations

This subsection describes the following functions used to edit data, or control editing of rectangular areas in the Logical Display.

DECCRA - Copy Rectangular Area
DECFRA - Fill Rectangular Area
DECERA - Erase Rectangular Area
DECSERA - Selective Erase Rectangular Area
DECCARA - Change Att In Rectangular Area
DECRARA - Reverse Attributes Rectangular Area
DECSACE - Select Attribute Change Extent

COPY RECTANGULAR AREA DECCRA

Level: 4

Purpose: To copy a rectangular array of characters from one part of a logical display page to another.

Format:

CSI	Pts;	Pls;	Pbs;	Prs;	Pps;	Ptd;	Pld;	Ppd	$	v
9/11	Pts;	Pls;	Pbs;	Prs;	Pps;	Ptd;	Pld;	Ppd	2/4	7/6

Description: When this control is received, the terminal copies the character values and attributes in the specified rectangular area of character positions defined by Pts, Pls, Pbs, and Prs on the specified page Pps, to the area specified by the coordinate Ptd, Pld on the specified page Ppd. Pts, Pbs, and Ptd are line numbers. Pls, Prs, and Pld are column numbers. Pps and Ppd are page numbers. The character positions (Pts, Pls) and (Pbs, Prs) define the rectangular area that is to be copied from page Pps.

The coordinates of the rectangular area are affected by the setting of Origin Mode. This control is not otherwise affected by the margins.

If the rectangular area contains both single and double width lines, the area copied FROM may not appear rectangular on the display. For example if Pl=5 and Pr=10, then characters 5 through 10 from all lines within the rectangular area will be copied whether they are single width or double width characters. Text copied to the destination area take on the line attributes of that area.

Pts must be less or equal to Pbs, and Pls must be less than or equal to Prs, otherwise the entire control function is ignored. Default values are Pts = 1, Pls = 1, Pbs = last-line-of-page, Prs = last-column-of-page, Pps = 1, Ptd = 1, Pld = 1, and Ppd = 1. If a value exceeds the width or height of the active page, it is treated as the width or height of the active page. If a page value exceeds the number of pages available in the current Page Configuration, it is treated as the last page in the current configuration.

If the destination rectangle specified is partially off of the Page, clipping of information will occur; only information which will appear in valid character positions in the page will be copied.

The Active Position does not change.

FILL RECTANGULAR AREA DECFRA

Level: 4

Purpose: To fill a group of adjacent character positions defined by a rectangular area.

Format:

CSI	Pch	;	Pt	;	Pl	;	Pb	;	Pr	$	x
9/11	Pch	;	Pt	;	Pl	;	Pb	;	Pr	2/4	7/8

Description: When this control is received, the terminal fills the rectangular area of character positions defined by Pt, Pl, Pb, and Pr, replacing both the character and the rendition present in those character positions with the character defined by the decimal value of Pch and the current renditions set through the SGR command.

The Pch parameter may be a decimal value from 32 to 126, or from 160 to 255. The decimal value refers to the character in the current GL and GR "in-use" character table, which is used to fill the specified rectangular area. If the value of Pch is not in the above range, the command is ignored.

Pt and Pb are line numbers, and Pl and Pr are column numbers. The character positions (Pt, Pl) and (Pb, Pr) define the rectangular area.

The coordinates of the rectangular area are affected by the setting of Origin Mode. This control is not otherwise affected by the margins.

Pt must be less or equal to Pb, and Pl must be less than or equal to Pr, otherwise the entire control function is ignored. Default values are Pch = 32, Pt = 1, Pb = last-line-of-page, Pl = 1, and Pr = last-column-of-page. If a value exceeds the width or height of the active page, it is treated as the width or height of the active page.

Notes on DECFRA:

1. If the rectangular area contains both single and double width lines, the area affected may not appear rectangular on the display. For example if Pl=5 and Pr=10, then all lines within the rectangular area, whether single or double width, will have character positions 5 through 10 affected.

2. DECFRA does not change any line attributes. Double height or double width characters will be filled into positions on lines containing those attributes. The Active Position does not move.

ERASE RECTANGULAR AREA — DECERA

Level: 4

Purpose: To erase a group of adjacent characters defined by a rectangular area.

Format:

CSI	Pt	;	Pl	;	Pb	;	Pr	$	z
9/11	Pt	;	Pl	;	Pb	;	Pr	2/4	7/10

Description: When this control is received, the terminal erases the rectangular area of character positions defined by Pt, Pl, Pb, and Pr, clearing all character attributes. Line attributes are not changed. When an area is erased, all characters are replaced with the Space character (2/0). Any unoccupied character positions remain unoccupied. The Active Position does not move.

Pt and Pb are line numbers, and Pl and Pr are column numbers. The character positions (Pt, Pl) and (Pb, Pr) define the rectangular area.

The coordinates of the rectangular area are affected by the setting of Origin Mode. This control is not otherwise affected by the margins.

Pt must be less or equal to Pb, and Pl must be less than or equal to Pr, otherwise the entire control function is ignored. Default values are Pt = 1, Pb = last-line-of-page, Pl = 1, and Pr = last-column-of-page. If a value exceeds the width or height of the active page, it is treated as the width or height of the active page.

Notes on DECERA:

1. If the rectangular area contains both single and double width lines, the area affected may not appear rectangular on the display. For example if Pl=5 and Pr=10, then all lines within the rectangular area, whether single or double width, will have character positions 5 through 10 affected.

SELECTIVE ERASE RECTANGULAR AREA DECSERA

Level: 4

Purpose: To erase a group of unprotected characters defined by a rectangular area.

Format:

CSI	Pt	;	Pl	;	Pb	;	Pr	$	{
9/11	Pt	;	Pl	;	Pb	;	Pr	2/4	7/11

Description: This control function erases unprotected characters within the rectangular area of character positions defined by Pt, Pl, Pb, and Pr. Unprotected characters are those with the "Selective Erase" character attribute (see DECSCA). When a character is erased, it is replaced with the Space character (2/0). Unoccupied character positions are not affected. This sequence does not change or clear any video character attributes (SGR or DECSCA), video line attributes, or the selection attribute. The active position does not move.

Pt and Pb are line numbers, and Pl and Pr are column numbers. The character positions (Pt, Pl) and (Pb, Pr) define the rectangular area.

The coordinates of the rectangular area are affected by the setting of Origin Mode. This control is not otherwise affected by the margins.

Pt must be less or equal to Pb, and Pl must be less than or equal to Pr, otherwise the entire control function is ignored. Default values are Pt = 1, Pb = last-line-of-page, Pl = 1, and Pr = last-column-of-page. If a value exceeds the width or height of the active page, it is treated as the width or height of the active page.

Notes on DECSERA:

1. If the rectangular area contains both single and double width lines, the area affected may not appear rectangular on the display. For example if Pl=5 and Pr=10, then all lines within the rectangular area, whether single or double width, will have character positions 5 through 10 affected.

CHANGE ATTRIBUTES RECTANGULAR AREA DECCARA

Level: 4

Purpose: To change the character rendition of a group of adjacent characters without having to re-write them.

Format:

	CSI	Pt	;	Pl	;	Pb	;	Pr	;	Ps1 ; Ps2...Psn	$	r
	9/11	Pt	;	Pl	;	Pb	;	Pr	;	Ps1 ; Ps2...Psn	2/4	7/2

Description: This sequence changes the video attributes (bold, blink, underline, and reverse) for all of the character positions in a rectangular area without altering any characters in those positions. The active position does not move.

The area of the page which is affected is indicated by the first 4 parameters. Pt and Pb are line numbers; Pl and Pr are column numbers. The character positions (Pt, Pl) and (Pb, Pr) define the top-left and bottom-right corners of the rectangular area. Pt must be less than or equal to Pb, and Pl must be less than or equal to Pr. Default values are Pt = 1, Pb = last-line-of-page, Pl = 1, and Pr = last-column-of-page. If a value exceeds the width or height of the active page, it is treated as the width or height of the active page.

The coordinates of the rectangular area are affected by the current setting of Origin Mode. This control is not otherwise affected by the margins.

The character positions affected depend on the current setting of DECSACE (STREAM or RECTANGLE). See DECSACE for details.

The video attribute(s) that the character positions are set to is indicated by one or more subsequent selective parameters. These parameters correspond to the parameters of the Set Graphic Rendition control function (SGR):

Parameter	Parameter Meaning
0	Attributes off (normal intensity, positive image, not underlined, not blinking)
1	Bold or increase intensity
4	Underscore
5	Blink
7	Negative (reverse) image
22	Normal intensity
24	Not underlined
25	Not blinking
27	Positive image

All other parameter values shall be ignored unless they are part of a well defined extension to the architecture (such as Color Text). When multiple parameters are used, they are cumulative, as if several DECCARA control functions were received and executed in sequence.

The default for the video attribute parameters, if no valid subsequent parameter is received, is 0, which will clear all video attributes in the specified area. This command does not change the current graphic rendition.

Notes on DECCARA:

1. If the rectangular area contains both single and double width lines, the area affected may not appear rectangular on the display. For example if Pl=5 and Pr=10, then all lines within the rectangular area, whether single or double width, will have character positions 5 through 10 affected.

REVERSE ATTRIBUTES RECTANGULAR AREA DECRARA

Level: 4

Purpose: To reverse character attributes of a group of adjacent characters without having to re-write them.

Format:

```
CSI   Pt ; Pl ; Pb ; Pr ; Ps1 ; Ps2...Psn  $    t
9/11  Pt ; Pl ; Pb ; Pr ; Ps1 ; Ps2...Psn  2/4  7/4
```

Description: This sequence reverses one or more video attributes (bold, blink, underline, reverse) for all of the character positions in a rectangular area without altering any characters in those positions. The active position does not move. To reverse the attribute bit within the specified range means to set the specified attribute bits if it was not previously set, and reset the bit(s) if it was already set.

The area of the page which is affected is indicated by the first 4 parameters. Pt and Pb are line numbers; Pl and Pr are column numbers. The character positions (Pt, Pl) and (Pb, Pr) define the top-left and bottom-right corners of the rectangular area.

The coordinates of the rectangular area are affected by the setting of Origin Mode. This control is not otherwise affected by the margins.

The character positions affected depend on the current setting of DECSACE (STREAM or RECTANGLE). See DECSACE for more details.

Pt must be less or equal to Pb, and Pl must be less than or equal to Pr, otherwise the entire control function is ignored. Default values are Pt = 1, Pb = last-line-of-page, Pl = 1, and Pr = last-column-of-page. If a value exceeds the width or height of the active page, it is treated as the width or height of the active page.

The video attribute(s) to be reversed are in the affected area are indicated by one or more subsequent parameters. These parameters are similar to the parameters of the Set Graphic Rendition control function (SGR):

Parameter	Parameter Meaning
0	Reverse all attributes
1	Reverse bold attribute
4	Reverse underscore attribute
5	Reverse blinking attribute
7	Reverse negative (reverse) image attribute

All other parameter values shall be ignored unless they are part

of a well defined extension to the architecture. Note if the Color Text Extension is present, the color text SGR values are ignored since the "reverse" of a color is not defined by the extension.

When multiple parameters are used, they are cumulative, as if several DECRARA control functions were received and executed in sequence.

There is no default for the video attribute parameters, if no valid subsequent parameter is received, the command is ignored.

This sequence does not change the current graphic rendition (see SGR).

Notes on DECRARA:

1. If the rectangular area contains both single and double width lines, the area affected may not appear rectangular on the display. For example if Pl=5 and Pr=10, then all lines within the rectangular area, whether single or double width, will have character positions 5 through 10 affected.

SELECT ATTRIBUTE CHANGE EXTENT DECSACE

Level: 4

Purpose: To specify what character positions are affected by DECCARA and DECRARA control functions (STREAM or RECTANGLE)

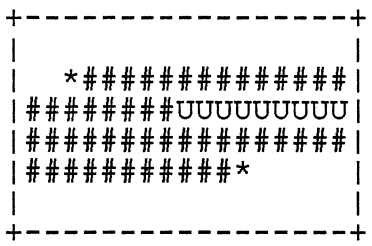
Format:

CSI	Ps	*		x
9/11	Ps	2/10		7/8

Ps = 0 stream of character positions are affected (default)
Ps = 1 stream of character positions are affected
Ps = 2 rectangular area of character positions are affected

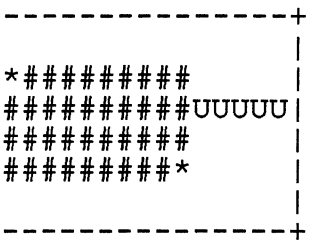
Description: This control function is used to specify what character positions are affected by the DECCARA and DECRARA control functions.

When Ps = 0 or 1, DECCARA and DECRARA affects the stream of character positions beginning with the first character position specified in the command, and ending with the second character position specified. The following diagram shows an example of these character positions. The two asterisks represent the character positions specified in the DECCARA or DECRARA control function. The "U"'s represent unoccupied character positions. These unoccupied positions are not affected by DECCARA or DECRARA when in this mode.



Ps = 0

When Ps = 2 DECCARA and DECRARA affects all character positions within a rectangular area, the top-left and bottom right corners of which are specified in the control function. The following diagram shows an example of these character positions. The two asterisks represent the character positions specified in the DECCARA or DECRARA control function. The "U"'s represent unoccupied character positions. When DECCARA or DECRARA is executed, these character positions are changed to blanks and are affected.



In both cases the specified character positions (marked by asterisks) are also affected.

5.12.2 Data Integrity

The data integrity features described here are application functions intended to improve the reliability of OLTP applications. They are not a substitute for error free communications as might be provided by an error correcting protocol.

REQUEST CHECKSUM OF RECTANGULAR AREA — DECRQCRA

Level: 4

Purpose: To cause the terminal to compute and report the checksum of a rectangular area, allowing applications to insure it has not been changed.

Format:
```
CSI   Pid ; Pp ; Pt ; Pl ; Pb ; Pr  *     y
9/11  Pid ; Pp ; Pt ; Pl ; Pb ; Pr  2/10  7/9
```

Description: Upon receipt of a DECRQCRA by the device, a checksum of the indicated rectangular area of the indicated page will be generated and reported to the host in a Checksum Report (DECCKSR).

Pid is an arbitrary numeric label provided by the application to identify the checksum request. It will be provided in the DECCKSR response from the device. This label serves to differentiate between multiple checksum reports if necessary.

Pp is the page number where the rectangular area of interest is located. If Pp is 0 or omitted, subsequent parameters are ignored and a checksum for all page memory will be reported. If Pp is higher than the total number of pages available in the device, the highest numbered page is used.

The final four parameters indicate the top, left (Pt, Pl) and bottom, right (Pb, Pr) coordintate of the rectangular area on page Pp of which the checksum is to be calculated. The coordinates of the rectangular area are affected by the setting of Origin Mode (DECOM). Pt must be less than or equal to Pb, and Pl must be less than or equal to Pr or the sequence is ignored.

Default values are: Pt = 1; Pb = last-line-of-page; Pl = 1; and Pr = last-column-of-page. If the final four parameters are unspecified, a checksum of page Pp is generated.

MEMORY CHECKSUM DEVICE STATUS REPORT DSR

Level: 4

Purpose: To request a checksum of the current text macro definitions, allowing the host to insure they have not been changed.

Format:

CSI	?	6	3	; Pid	n	
9/11	3/15	3/6	3/3	; Pid	6/14	

Description: This request causes a Checksum Report (DECCKSR) to be issued reporting the checksum of current text macro definitions.

Pid is an optional numeric parameter which provides a label to identify the checksum request. It will be provided in the DECCKSR response from the terminal. This label serves to differentiate between multiple checksum reports if necessary.

CHECKSUM REPORT DECCKSR

Level: 4

Purpose: To report a requested checksum.

Format:

DCS	Pn	!	~	D...D	ST
9/0	Pn	3/2	7/14	D...D	9/12

Description: DECCKSR is issued in response to a Request Checksum of Rectangular Area (DECRQCRA) or a Device Status Report (DSR) with a parameter value of ?63 indicating a request for a checksum of the macro definitions.

The numeric parameter of the DECCKSR control string is a numeric label for the checksum report specified in the DECRQCRA or DSR control function. This label serves to differentiate between multiple checksum reports if necessary.

The data of the control string consists of four hexadecimal digits (3/0 through 3/9 and 4/1 through 4/6) indicating the checksum.

5.12.3 Macros

DEFINE MACRO DECDMAC

Level: 4

Purpose: To define a string of characters as a text macro, allowing the host to shorten subsequent transmissions.

Format:

DCS	Pid	;	Pdt	;	Pen	!	z	D...D ST
9/0	Pid	;	Pdt	;	Pen	2/1	7/10	D...D 9/12

Where

Pid is the macro ID number: 0-63.

Pdt is the delete control:

- 0 delete only a macro with the same ID number (if one exists) before defining this macro.
- 1 delete all macro definitions before defining this macro.

Pen is the macro encoding:

- 0 text (ASCII) encoding.
- 1 hex pair encoding.

Description: This control string defines a macro consisting of ANSI commands. Once defined, a macro can be invoked using DECINVM to cause the macro's definition to be processed by the terminal as if received from the host.

Omitted parameters are defaulted to zero. Pid must be a number between 0 and 63; Pdt 0 or 1; and Pen 0 or 1; otherwise the control will be ignored.

The data of the control string is the series of text and ANSI commands which are performed when the macro is invoked.

Macros can invoke other macros with up to 16 levels of nesting, but cannot invoke themselves recursively. Macro definitions cannot be nested. That is, a DECDMAC cannot be included as part of a macro definition.

When text encoding is used (Pen = 0), the macro consists of the graphic characters appearing in the control string. Only characters from 2/0 through 7/14 and 10/00 through 15/15 may be included in a macro using this encoding method.

When hex pair encoding is used (Pen = 1), each pair of characters in the definition represents the hex value of a single ASCII

character which will be stored in the definition of the macro. If a non-hex pair is encountered that is not a valid repeat sequence, the entire macro definition is ignored.

A Repeat Introducer (!) is defined to allow a string of hex pairs to be repeated in the definition any number of times. Repeat sequences are imbedded within the text of the macro definition. If a non-hex character is encountered in the repeat string, the entire macro definition is ignored.

The format is:

```
! Pn ; D...D ;
```

Where

- ! is the repeat introducer
- Pn is a numeric parameter which specifies the number of times to repeat the sequence. If Pn is not specified, the sequence is inserted into the definition once.
- D...D is the sequence to repeat within the definition, Pn times.
- ; A semicolon separates the repeat count from the sequence of hex pairs. Another semicolon terminates the sequence.

If any non parameter characters are encountered before the first semicolon, the entire macro definition is ignored. If a string terminator is encountered before the final semicolon, the repeat string is terminated normally.

6k bytes of RAM is available for storage of macro definitions. Any macro definitions which cannot fit in the remaining macro storage space are ignored.

All macro definitions are cleared on receipt of a Reset to Initial State (RIS) or Secure Reset (DECSR). Soft Terminal Reset (DECSTR) has no effect on stored macro definitions.

Note that characters from 0/8 through 0/13 can be included to format the device control string, but are not part of the macro definition.

INVOKE MACRO DECINVM

Level: 4

Purpose: To invoke a previously defined macro definition.

Format:

	CSI	Pid	*	z
	9/11	Pid	2/10	7/10

Description: Causes the specified macro to be invoked. The macro must have been previously defined using DECDMAC. The terminal substitutes the DECINVM sequence with the contents of the macro definition, and executes the substituted text.

Pid is the macro id number. If a macro is not found with the specified id, the command is ignored.

The current state of the machine is not saved or restored when macros are invoked. Any functions executed by a macro invocation are still in effect when the macro invocation finishes.

If the terminal is in Printer Controller mode, the macro is not expanded by the terminal. The command to invoke the macro is passed on to the printer, to be expanded by the printer's text macro facility.

MACRO SPACE REPORT DECMSR

Level: 4

Purpose: To report the space remaining for macro definitions in response to DSR request.

Format:

CSI	Pn	*		{
9/11	Pn	2/10		7/11

Description: DECMSR is reported in response to a Device Status Report (DSR) with a private selectable parameter ?62, specifying a request to report the available space for macro definitions.

The numeric parameter specifies the number of bytes available for macro definitions divided by 16 (rounded down).

5.13 SAVING AND RESTORING TERMINAL STATE

5.13.1 Cursor Save Buffer

A conforming device shall provide a means of saving and restoring on command the value of the Active Position, as well as certain of the terminal states. The state which must be saved in Level 1 operation includes the Current Rendition value (determined by the most recent occurrence of Select_Graphic_Rendition), the current value of Origin Mode, the character sets designated to G0, G1, G2, and G3, and the character set most recently invoked into the left hand side (GL) of the In Use Table.

In Level 2 operation, all of this state must also be saved, as well as the character set most recently invoked into the right hand side (GR) of the In Use Table. If the Level 2 extension Selectively Erasable Characters is supported, the Current Attribute value (determined by the most recent occurrence of Select_Character_Attribute) must be saved. If the ANSI Color Text Extension is supported, the current foreground and background text writing colors are saved as part of the Current Rendition.

| Level 3 and 4 require the same information as Level 2.

SAVE CURSOR DECSC

Levels: 1-4

Purpose: Saves the Active Position and other selective state information.

Format:

	ESC	7
	1/11	3/7

Description: In Level 1 operation, the DECSC control provides a means of storing the values of the Active Position, the Current Rendition, Origin Mode, the currently designated G0, G1, G2, and G3 sets, and the currently invoked GL set.

In Level 2 or higher it also stores the currently invoked GR set. The Current Attribute value is also saved if the Level 2 extension for Selectively Erasable Characters is supported. These values can later be restored using the DECRC control.

Notes:

1. Only one level of storage is provided by this control. Each additional execution of this control will cause previously stored values to be lost.

2. The values of the specified state information (see above) are not affected by the execution of this control.

3. On initialization or reset the values of the save buffer are cleared to the following state:

```
position.line = 1;
position.column = 1;
position.page = 1;
rendition.bold = FALSE;
rendition.underscore = FALSE;
rendition.blink = FALSE;
rendition.reverse = FALSE;
attribute.selective_erase = FALSE;
origin_mode = ABSOLUTE;
left = in_use_table.gl;
right = in_use_table.gr
g0 = designated_graphic_sets[0];
g1 = designated_graphic_sets[1];
g2 = designated_graphic_sets[2];
g3 = designated_graphic_sets[3];
```

4. Execution of Soft Terminal Reset (DECSTR) also causes the Save Buffer to be set to the default values as given above.

State Affected:

```
    save_buffer_t  cursor_save_buffer;
```

Algorithm:

```
void save_cursor()
{
cursor_save_buffer.position.line =
            active_position.line;
cursor_save_buffer.position.column =
            active_position.column;
cursor_save_buffer.position.page =
            active_position.page;
cursor_save_buffer.rendition.bold =
           current_rendition.bold;
cursor_save_buffer.rendition.underscore =
           current_rendition.underscore;
cursor_save_buffer.rendition.blink =
           current_rendition.blink;
cursor_save_buffer.rendition.reverse =
           current_rendition.reverse;
cursor_save_buffer.origin_mode = origin_mode;
cursor_save_buffer.left = in_use_table.gl;
cursor_save_buffer.g0 = designated_graphic_sets[0];
cursor_save_buffer.g1 = designated_graphic_sets[1];
cursor_save_buffer.g2 = designated_graphic_sets[2];
cursor_save_buffer.g3 = designated_graphic_sets[3];
if (conformance_level >= LEVEL_2)
  {
  cursor_save_buffer.right = in_use_table.gr;
  cursor_save_buffer.attribute.selective_erase =
                     current_attribute.selective_erase;
  }
} /* end save_cursor */
```

Known Deviations: None

RESTORE CURSOR DECRC

Levels: 1-4

Purpose: Restores the previously saved values of the Active Position and other selective state information.

Format:

	ESC	8
	1/11	3/8

Description: In Level 1 operation, the DECRC control provides a means of restoring the values of the Active Position, the Current Rendition, Origin Mode, the currently designated G0, G1, G2, and G3 sets, and the currently invoked GL set.

In Level 2 or higher it also restores the currently invoked GR set. The Current Attribute value is also saved if the Level 2 extension for Selectively Erasable Characters is supported. These values are the ones which were previously stored using the Save Cursor (DECSC) control function.

Notes:

1. The values of the specified state information (see above) at the time this control is received by the terminal are lost.

2. The values stored in the Cursor Save Buffer are not affected by execution of this control.

3. If the restored cursor position lies outside of the scrolling region, and the restored value of Origin Mode is set, the Active Position will be moved to the closest position within the scrolling region. That is, if the restored cursor position is above the Top Margin, the Active Position will be set to the Top Margin, and if the restored position is below the Bottom Margin, the Active Position will be set to the Bottom Margin. In either case, the Active Column will not change, but will remain as the stored column value. For example: the Active Position is at line 5, column 5 when the cursor is saved, and Origin Mode is set; the scrolling region is subsequently changed to 10,15; when the cursor is restored, the Active Position will be moved to line 10, column 5, to insure that it will lie inside the scrolling region. This reason for this recovery is to guarantee that the Active Position will always be within the bounds of the scrolling region, and will never have a negative value.

State Affected:

```
    active_position
    current_rendition
    origin_mode
    in_use_table
    designated_graphic_sets[4]
    current_attribute
```

Algorithm:

```
void restore_cursor()
{
|   short int base;  /* first line(-1) of page */
| if (cursor_save_buffer.position.page > number_of_pages)
|   active_position.page = number_of_pages;
| else active_position.page = cursor_save_buffer.position.page;
| base = (active_position.page-1) * lines_per_page;
| if (cursor_save_buffer.position.column >
|     end_of_line(active_position.line + base))
|   active_position.column = end_of_line(active_position.line +
| base);
| else
|   active_position.column = cursor_save_buffer.position.column;
active_position.line = cursor_save_buffer.position.line;

current_rendition.bold =
  cursor_save_buffer.rendition.bold;
current_rendition.underscore =
  cursor_save_buffer.rendition.underscore;
current_rendition.blink =
  cursor_save_buffer.rendition.blink;
current_rendition.reverse =
  cursor_save_buffer.rendition.reverse;

origin_mode = cursor_save_buffer.origin_mode;
if (origin_mode == DISPLACED)
  {
  if (active_position.line < top_margin)
    active_position.line = top_margin;
  if (active_position.line > bottom_margin)
    active_position.line = bottom_margin;
|   if (active_position.column < left_margin)
|     active_position.column = left_margin;
|   if (active_position.column > right_margin)
|     active_position.column = right_margin;
  }

in_use_table.gl = cursor_save_buffer.left;
designated_graphic_sets[0] = cursor_save_buffer.g0;
designated_graphic_sets[1] = cursor_save_buffer.g1;
designated_graphic_sets[2] = cursor_save_buffer.g2;
```

```
designated_graphic_sets[3] = cursor_save_buffer.g3;

if (conformance_level >= LEVEL_2)
  {
  in_use_table.gr = cursor_save_buffer.right;
  if (level_2_extensions[SELECTIVE_ERASE]==TRUE)
    current_attribute.selective_erase =
      cursor_save_buffer.attribute.selective_erase;
  }
}
```

Known Deviations:

In the VT100, VT101, VT102, VT125, VT131, and VT132, if the restored cursor position lies outside of the scrolling region, and the restored value of Origin Mode is set, the Active Position will not be moved into the scrolling region, and an illegal (e.g., negative) cursor position value may result.

5.13.2 Terminal State Interrogation

Terminal State Interrogation (TSI) allows a host computer to query most of the settable state in the terminal. Terminal state is divided into three general catagories based on the ANSI syntax used to control it: (1) modes controlled by set__mode/reset__mode; (2) selections and settings such as current graphic rendition, and scroll margins. (3) presentation state such as tab stops and character sets. The terminal provides control functions to query state in each of these catagories, as well as a compact report that allows most settable state to be saved and later restored in one operation.

All TSI controls operate at Levels 2, 3 and 4. Level 3 defines TSI as an implicit extension at Level 2. The following control functions are provided to support Terminal State Interrogation.

DECRQM - Request Mode
: Sequence sent from host to terminal to request the state of a single mode.

DECRPM - Report Mode
: Sequence sent from terminal to host to report the state of the mode requested by DECRQM.

DECNKM - Numeric Keypad Mode
: This is a selectable mode sequence which controls the keypad allowing its state to be reported by DECRPM above. See "Keyboard Processing", DEC STD 70-6.

DECRQSS - Request Selection or Setting

Sequence sent from host to terminal to request the state of a selection or setting which is not encoded as a selectable mode (example: SGR).

DECRPSS - Report Selection or Setting
Sequence sent from terminal to host to report settings and selections in response to DECRQSS.

DECRQPSR - Request Presentation State Report
Sequence sent from host to terminal to request presentation state (cursor information and tab stops)

DECPSR - Presentation State Report
Control string sent from terminal to host in response to DECRQPSR. Two presentation state reports are defined: (1) Cursor Information Report - DECCIR; (2) Tabulation Stop Report - DECTABSR

DECRSPS - Restore Presentation State
Control string sent from host to terminal to restore the presentation state reported by DECPSR.

DECRQTSR - Request Terminal State Report
Sequence sent from host to terminal to request a compact report of most settable state.

DECTSR - Terminal State Report
Control string sent from terminal to host which reports most terminal state in one compact sequence.

DECRSTS - Restore Terminal State
Control string sent from host to VT420 to restore the terminal state reported by DECTSR.

REQUEST MODE DECRQM

Levels: 2x (TSI), 3-4

Purpose: To request the state of modes controlled by Set Mode/Reset Mode sequences.

Format:

CSI		Ps	$	p	(ANSI modes)
CSI	?	Ps	$	p	(DEC private modes)
9/11	3/15	Ps	2/4	7/0	

Description: This sequence takes one selective parameter specifying a single selectable mode controllable using the Set Mode and Reset Mode sequences. The terminal responds with a Report Mode control sequence (DECRPM) which specifies the state of the mode as follows: (0) Unknown (mode is not recognized by terminal); (1) Set; (2) Reset; (3) Set and cannot be changed; or (4) Reset and cannot be changed.

All ANSI standard modes defined in X3.64-1979 are reported as either Set or Reset. DEC Private modes that cannot be set or reset from the host will report as unknown [Reason: the list is long and changes frequently. Since DEC defines these modes, applications can infer how to handle unknown modes as needed].

Implementation Guideline

The VT420 allows the following selectable modes to be queried through the use of the DECRQM control:

ANSI Specified Mode Parameter	Mnemonic	DEC Private Mode Parameter	Mnemonic
1	GATM *	?1	DECCKM
2	KAM	?2	DECANM ***
3	CRM **	?3	DECCOLM
4	IRM	?4	DECSCLM
5	SRTM *	?5	DECSCNM
		?6	DECOM
7	VEM *	?7	DECAWM
10	HEM *	?8	DECARM
11	PUM *		
12	SRM		
13	FEAM *		
14	FETM *		
15	MATM *		
16	TTM *	?18	DECPFF
17	SATM *	?19	DECPEX
18	TSM *	?25	DECTCEM
19	EBM *		
20	LNM	?42	DECNRCM
		?60	DECHCCM *
		?61	DECVCCM
		?64	DECPCCM
		?66	DECNKM
		?67	DECBKM
		?68	DECKBUM
		?69	DECVSSM
		?73	DECXRLM

* This mode is always Reset in the VT420 and can not be changed.

** The terminal will not respond to a DECRQM sequence while in Control Representation Mode (this mode can only report RESET). CRM can not be changed from the host.

*** The terminal will not respond to a DECRQM sequence when not in ANSI mode (this mode can only report SET).

REPORT MODE DECRPM

Levels: 2x (TSI), 3-4

Purpose: To report the state of modes controlled by Set Mode/Reset Mode sequences (terminal to host).

Format:

CSI		Ps1	;	Ps2	$	y	(ANSI modes)
CSI	?	Ps1	;	Ps2	$	y	(DEC private modes)
9/11	3/15	Ps1	;	Ps2	2/4	7/9	

Description: This sequence is returned to the host upon receipt of a valid DECRQM control sequence. This sequence reports the state of a particular selectable mode to the host. The first selective parameter (Ps1) is the selectable mode number (the question mark ("?") must be added for DEC private selectable modes), that was requested from the DECRQM sequence. The second selective parameter (Ps2) indicates the state of that mode:

Ps2	State of mode
0	UNKNOWN (mode not recognized by terminal).
1	Mode is in SET state
2	Mode is in RESET state
3	Mode is in SET state, and cannot change
4	Mode is in RESET state, and cannot change

REQUEST SELECTION OR SETTING — DECRQSS

Levels: 2x (TSI), 3-4

Purpose: To request the state of a selection or setting which is not encoded as a Set Mode/Reset Mode sequence.

Format:

DCS	$	q	D...D	ST
9/0	2/4	7/1	D...D	9/12

Description: Request Selection or Setting is sent to the terminal to determine the selection or setting of a single device feature, such as the current Graphic Rendition (SGR). The data of the device control string for DECRQSS contains the identifying intermediate and final characters of the control sequence used to make the selection. For example, if the command is to request the state of SGR, the data would consist of a single character 'm', which is the final character (no intermediates) in the control sequence for SGR. The terminal responds with a Report Selection or Setting (DECRPSS) sequence.

Only one setting or selection may be queried per request.

Implementation Guideline

The VT420 allows the state of features selectable through the following sequences to be determined through DECRQSS:

Setting Control Sequence	Abbreviation	Final Character(s)
Select Active Status Display	DECSASD	$ }
Select Attribute Change Extent	DECSACE	* x
Set Character Attribute	DECSCA	" q
Set Conformance Level	DECSCL	" p
Set Columns Per Page	DECSCPP	$ |
Set Lines Per Page	DECSLPP	t
Set Number of Lines per Screen	DECSNLS	* |
Set Status Line Type	DECSSDT	$ ~
Set Left and Right Margins	DECSLRM	s
Set Top and Bottom Margins	DECSTBM	r
Set Graphic Rendition	SGR	m

REPORT SELECTION OR SETTING DECRPSS

Levels: 2x (TSI), 3-4

Purpose: To report the state of a selection or setting which is not encoded as a Set Mode/Reset Mode sequence (terminal to host).

Format:

DCS	Ps	$	r	D...D	ST
9/0	Ps	2/4	7/2	D...D	9/12

Description: The Report Selection or Setting device control string is issued from the terminal in response to a Request Selection or Setting command. The value of Ps indicates whether the Request was successful:

Ps	meaning
0	UNKNOWN selection or setting, request unsuccessful
1	request successful, data contains selection information

The data returned by the DECRPSS control string is the entire control function that the host would send to the terminal to select or set the parameter being inquired, based on the current setting or selection of that parameter, with the single exception of the leading Control Sequence Introducer character (9/11, or 1/11 5/11 in 7 bits).

The returned control function will contain no omitted parameters, even if the inquired value is the default value. In the case where there is no default value for the inquired parameter, the returned control function will be constructed so as to overwrite the existing setting or selection, if and when that information is returned to the device.

For example, if the current Graphic Rendition is blinking, and reversed, the data would consist of the characters '0;5;7m', which is the parameter string and final character necessary to change the graphic rendition to that state.

Note that the coding of the DECRPSS control string which indicates UNKNOWN, is always coded as follows:

DCS 0 $ r ST

REQUEST PRESENTATION STATE REPORT DECRQPSR

Levels: 2x (TSI), 3-4

Purpose: To request presentation state (cursor information and tab stops).

Format:

CSI	Ps	$	w
9/11	Ps	2/4	7/7

Description: DECRQPSR causes the terminal to respond with a Presentation State Report (DECPSR). The control sequence takes one selective parameter which specifies the type of Presentation State Report to be returned.

Ps	Type of DECPSR
0	ignored, no report sent
1	Cursor Information Report (DECCIR)
2	Tabulation Stop Report (DECTABSR)

PRESENTATION STATE REPORT DECPSR

Levels: 2x (TSI), 3-4

Purpose: To report the presentation state (cursor information and tab stops, terminal to host).

Format:

DCS	Ps	$	u	D...D	ST
9/0	Ps	2/4	7/5	D...D	9/12

Description: The Presentation State Report is a device control string containing data which represents the settings of presentation attributes in the terminal. A Presentation State Report is sent in response to a Request Presentation State Report control sequence (DECRQPSR).

The type of Presenation State Report is specified by the value of Ps:

Ps	Type of DECPSR	
0	illegal	
1	Cursor Information Report	(DECCIR)
2	Tabulation Stop Report	(DECTABSR)

CURSOR INFORMATION REPORT DECCIR

Levels: 2x (TSI), 3-4

Purpose: To report the presentation state cursor information

Format:

DCS	1	$	u	D...D	ST
9/0	3/1	2/4	7/5	D...D	9/12

Description: The data of a device control string for a Cursor Information Report consists of a string of parameters containing the information saved through a Save Cursor (DECSC) command, according to the following format:

Pr ; Pc ; Pp ; Srend ; Satt ; Sflag ; Pgl ; Pgr ; Scss ; Sdesig

where:

Pr, Pc — The cursor position (example: 24;132)

Pp — The current page (example: 3)
Note that in a single page configuration, the Pp parameter would be 1.

Srend — One or more bytes containing the encoded Graphic Renditions that must be saved, according to the following bit-encoded format:

bit 8: always reset (0)
bit 7: always set (1)
bit 6: extension indicator: if this bit is set (1), another byte of rendition information follows (always reset on VT420).
bit 5: always reset (0)
bit 4: reverse video rendition
bit 3: blink rendition
bit 2: underline rendition
bit 1: bold rendition

For the actual rendition bits (1 through 5, above) if the bit is set (1), it indicates that the rendition has been selected.
Example: if bold and underline are set the terminal would report a single byte, coded in binary as 01000011, or a hex 43, ASCII "C".

Satt — One or more bytes containing the encoded Character Attributes that must be saved, according to the following bit-encoded format:

bit 8: always reset (0)
bit 7: always set (1)
bit 6: extension indicator: if this bit is set (1), another byte of character attribute information follows (always reset on VT420).
bit 5: reserved for future use
bit 4: reserved for future use
bit 3: reserved for future use
bit 2: reserved for future use
bit 1: selectively erasable attribute

For the actual character attribute bits (1 through 5, above) if the bit is set (1), it indicates that the character attribute has been selected. Example: if the selective erase attribute is set, the terminal would report a single byte, coded in binary as 01000001, or a hex 41, ASCII "A".

Sflag — One or more bytes containing several flags and modes that must be saved, according to the following bit-encoded format:

bit 8: always reset (0)
bit 7: always set (1)
bit 6: extension indicator: if this bit is set (1), another byte of flags follows (always reset on VT420).
bit 5: reserved for future use
bit 4: auto-wrap pending (1 if pending)
bit 3: SS3 pending (1 if SS3 received)
bit 2: SS2 pending (1 if SS2 received)
bit 1: origin mode (1 = set 0 = reset)

For the flags above, if the bit is set (1) it indicates that the mode or state is set (as indicated). Example: if origin mode DECOM is set, auto-wrap is pending, and an SS3 has been received, the terminal would report a single byte, coded in binary as 01001101, or a hex 4D, ASCII "M".

Pgl, Pgr — Character G-sets which are currently invoked into GL and GR. These parameters are characters from 0 through 3, depending on which of G0, G1, G2, or G3 are invoked into GL and GR, respectively.

Scss — One byte containing the encoded information regarding the size of the character sets designated into G0 through G3 according to the following bit-encoded format:

bit 8: always reset (0)
bit 7: always set (1)
bit 6: extension indicator: if this bit is set (1), another byte of character size information follows (always reset on VT420).
bit 5: reserved for future use
bit 4: size of G3 set
bit 3: size of G2 set
bit 2: size of G1 set
bit 1: size of G0 set

For the character set size bits (1 through 4, above) if the bit is set (1), it indicates that the graphic set contains a 96-character character set. If the bit is reset, the graphic set contains a 94-character character set. Example: if ISO Latin Alphabet Nr 1 Supplemental, a 96-character character set is designated into G2 and G3, and ASCII, a 94-character character set is designated into G0 and G1, the terminal would report a single byte, coded in binary as 01001100 = hex 4C = ASCII "L".

Sdesig A string of bytes which consist of the strings of designating tails of the character sets designated into G0, G1, G2, and G3, in that order. The designating tail consists of the intermediate and final characters of the Select Character Set (SCS) designating escape sequence for the character set.

(Example: If ASCII were designated into G0, the Line Drawing (DEC Special Graphics) set into G1, and the DEC Supplemental Set into G2 and G3, the value of Sdesig would be: 'B0%5%5')

Note: this string is deliberately placed at the end of the data of the DECCIR control string, because the semicolon (;) separator character used between the previous parameters is a valid character in a SCS escape sequence.

Note: This string will report the User Preference Supplemental Set (UPSS) designator if that is how a character set was designated. DECRQUPSS can be used to determine the actual UPSS if needed.

Example

An example DECCIR report, assuming the default state of the terminal would be:

<DCS>1$u1;1;1;@;@;@;0;2;@;BB<< <ST>

indicating that:

- "1;1;1" - cursor home on first page
- "@;@" - no graphic rendition or character attribute active; DECOM reset, no SS2, SS3, or auto-wrap is pending
  (ASCII @ is binary 01000000)
- "0;2" - G0 in GL and G2 in GR
- "@" - all character sets designated are 94-character character sets
- "BB<<" - ASCII in G1 and G2, User Preference Supplemental Set in G2 and G3

TABULATION STOP REPORT DECTABSR

Levels: 2x (TSI), 3-4

Purpose: To report the presentation state tab stops

Format:

DCS	2	$	u	D...D	ST
9/0	3/2	2/4	7/5	D...D	9/12

Description: The data of a device control string for a Tabulation Stop Report consists of a string of numeric parameters specifying where tabulation stops are currently set in the terminal, separated by slash ("/") characters. An example of a DECTABSR when the terminal has 8 column tabs set would be as follows:

DCS 2 $ u 9 / 17 / 25 / 33 / 41 / 49 / 57 / 65 / 73 ST

RESTORE PRESENTATION STATE DECRSPS

Levels: 2x (TSI), 3-4

Purpose: To report previously reported presentation state (cursor information or tab stops).

Format:

DCS	Ps	$	t	D...D	ST
9/0	Ps	2/4	7/4	D...D	9/12

Description: This device control string is sent to reset the terminal to the presentation state described in the data of the control string. This DCS takes one selective parameter, which indicates the format of the data of the DCS. The data of the DCS must be in the format of one of the Presentation State Reports (DECPSR). The type of DECPSR format that the data consists of is specified by the selective parameter:

Ps	Type of DECPSR
0	error, restore ignored
1	Cursor Information Report (DECCIR)
2	Tabulation Stop Report (DECTABSR)

When this sequence is received by the terminal, the terminal will begin to change the terminal state according to the state encoded in the data of the control string.

If an invalid value is detected, the terminal will ignore the error, if possible, and if not will ignore the remainder of the data, parsing until a ST control is encountered. This may leave the terminal state partially unrestored. No error indication is returned to the host.

REQUEST TERMINAL STATE REPORT DECRQTSR

Levels: 2x (TSI), 3-4

Purpose: To request a compact report of most settable state.

Format:

CSI	Ps	$	u	
9/11	Ps	2/4	7/5	

Description: DECRQTSR causes the terminal to issue a Terminal State Report (DECTSR). Terminal State Reports transmit information on global state within the terminal. The type of DECTSR returned depends on the value of the selective parameter.

Ps	Type of DECTSR
0	ignored, no report sent
1	Terminal State Report (DECTSR)

If the terminal does not recognize the type of DECTSR requested, the entire command is ignored.

TERMINAL STATE REPORT (1) DECTSR

Levels: 2x (TSI), 3-4

Purpose: To report most settable terminal state in one compact sequence.

Format:

DCS	1	$	s	D...D	ST
9/11	3/1	2/4	7/3	D...D	9/12

Description: The Terminal State Report is a device control string containing a collection of terminal device attributes. A Terminal State Report is sent in response to a Request Terminal State Report control sequence (DECRQTSR) with Selective Parameter "1".

The data of the device control string for a Terminal State Report consists of a Sixel encoded, string of 8-bit bytes which contains the the state information for the device. This string is guaranteed not to be longer than 256 characters, including the DCS control character, introducer sequence, and ST control character.

The state information reported is the same as the state information available through the other discrete TSI controls. This generally represents all settable state in the terminal with the exception of large data items that are not practical to report (the contents of the logical display, DRCS, UDK definitions, and Macros).

The format of the data is firmware revision dependent. Software should not expect the format to remain the same across firmware revisions or different members of a terminal family.

The format of DECTSR is: DCS 1 $ s D1...Dn <checksum1> <checksum2> ST

Where D1...Dn are data reported
<checksum1> <checksum2> is a two bytes check sum of D1...Dn

(checksum = 2's complement of D1 + D2 + ... + Dn;
8-bit addition with no Carry)

Implementation Guideline:

On the VT420, there are 110 bytes in D1...Dn, system parameters are contained in low nibble of each byte, and D1...Dn are in the range 4/0 to 4/15 (bit 6 of Dn always set, bit 7,5,4 of Dn always reset).

RESTORE TERMINAL STATE DECRSTS

Levels: 2x (TSI), 3-4

Purpose: To restore previously reported terminal state (DECTSR).

Format:

DCS	Ps	$	p	D...D	ST
9/0	Ps	2/4	7/0	D...D	9/12

Description: This device control string is sent to reset the terminal to the state described in the data of the control string. This DCS takes one selective parameter, which indicates the format of the data of the DCS. The data of the DCS must be in the format of one of the Terminal State Reports (DECTSR). The type of DECTSR format that the data consists of is specified by the selective parameter:

Ps	Type of DECTSR
0	illegal, restore ignored
1	Terminal State Report (DECTSR)

When this sequence is received, buffered, and checksummed by the terminal, the terminal will begin to change the terminal state according to the state encoded in the data of the control string. If the checksum fails, the entire sequence is ignored. If an invalid value is detected, the terminal will ignore the error if possible, and if not, ignore the remainder of the data, parsing until a ST control is encountered. This may leave the terminal in a partially unrestored state. No error indication is returned to the host.

Note

Software should not depend on the format of the Terminal State Report (DECTSR) to be the same across members of the VT400 family, nor even across different firmware revisions of a single terminal model. Software should not use DECRSTS to restore a terminal to a saved state unless that state was previously read from the terminal.

5.14 INTERNAL FUNCTIONS AND PROCEDURES

5.14.1 End Of Line

```
/*************
*
* end_of_line
* This function returns the last position of the line
* based on its line attributes and column mode.
*
* assumes line is on the current page
*************/
short int end_of_line (line)
  line_t line;
{
|   line_t base;  /* first line(-1) of page */
  column_t end_line;
/*...........*/
if (column_mode = ONE_THIRTY_TWO) end_line = 132;
else end_line = 80;

| base = (active_position.page-1) * lines_per_page;
| if (line_rendition[line + base] != SINGLE_WIDTH)
  end_line /= 2;
return(end_line);
}
```

5.14.2 Scroll Up

```
/****************
*
* scroll_up
*
* This procedure causes the scroll area between the line
* designated and the Bottom Margin (inclusive) on the
* active page to scroll upward by the number of lines
* designated in the count.  Line and Character Renditions
* are scrolled with the data.  A new single-width line with
* all character positions empty is inserted at the Bottom Margin.
* Lines scrolled off the top of the scrolling region are lost.
*
******************/
void scroll_up(top_line, count)
  line_t top_line;
  line_t count;
{
  line_t y;
  column_t x;
|   line_t base;      /* first line(-1) of page */
| /*..............*/
| base = (active_position.page-1) * lines_per_page;
while (count > 0)
  {
|   for (y=top_line+base; y<bottom_margin+base; y++)
    {
    line_rendition[y] = line_rendition[y+1];
|   for (x=left_margin; x<=right_margin; x++)
      strncpy( display[y][x], display[y-1][x],
               sizeof(character_t) );
    }
|   line_rendition[bottom_margin+base] = SINGLE_WIDTH;
|   for (x=left_margin; x<=right_margin; x++)
    display[bottom_margin][x].code = EMPTY_CHARACTER;
  count -= 1;
  }
} /* end scroll_up */
```

5.14.3 Scroll Down

```
/****************
*
* scroll_down
*
* This procedure causes the scroll area between the line
* designated and the Bottom Margin (inclusive) on the
* active page to scroll downward by the number of lines
* designated in the count.  Line and Character Renditions
* are scrolled with the data.  A new single-width line with
* all character positions empty is inserted at the Top Margin.
* Lines scrolled off the bottom of the scrolling region are lost.
*
******************/
void scroll_down(top_line, count)
  line_t top_line;
  line_t count;
{
  line_t y;
  column_t x;
|   line_t base;      /* first line(-1) of page */
| /*..............*/
| base = (active_position.page-1) * lines_per_page;
while (count > 0)
  {
|   for (y=bottom_margin+base; y>top_line+base; y--)
    {
    line_rendition[y] = line_rendition[y-1];
|   for (x=left_margin; x<=right_margin; x++)
      strncpy( display[y][x], display[y-1][x],
               sizeof(character_t));
    }
|   line_rendition[top_line+base] = SINGLE_WIDTH;
|   for (x=left_margin; x<=right_margin; x++)
    display[top_line][x].code = EMPTY_CHARACTER;
  count -= 1;
  }
} /* end scroll_down */
```

5.14.4 Scroll Left

```
/*****************
*
* scroll_left
*
* This procedure causes the scroll area between the column
* designated and the Right Margin (inclusive) on the
* active page to scroll leftward by the number of columns
* designated in the count.  A new column with all character
* positions empty is inserted at the Right Margin.
* Characters scrolled past the scrolling region are lost.
*
******************/
void scroll_left(left_column, count)
  column_t left_column;
  column_t count;
{
  line_t y;
  column_t x;
  line_t base;      /* first line(-1) of page */
/*...............*/
base = (active_position.page-1) * lines_per_page;
while (count > 0)
  {
  for (y=top_margin+base; y<=bottom_margin+base; y++)
    {
    for (x=left_column; x<right_margin; x++)
      strncpy( display[y][x], display[y][x+1],
               sizeof(character_t));
    display[y][right_margin].code = EMPTY_CHARACTER;
    }
  count -= 1;
  }
} /* end scroll_left */
```

5.14.5 Scroll Right

```
/*****************
*
* scroll_right
*
* This procedure causes the scroll area between the column
* designated and the Right Margin (inclusive) on the
* active page to scroll right by the number of columns
* designated in the count.  A new column with all character
* positions empty is inserted at the designated column.
* Characters scrolled past the scrolling region are lost.
*
******************/
void scroll_right(left_column, count)
  column_t left_column;
  column_t count;
{
  line_t y;
  column_t x;
  line_t base;      /* first line(-1) of page */
/*...............*/
base = (active_position.page-1) * lines_per_page;
while (count > 0)
  {
  for (y=top_margin+base; y<=bottom_margin+base; y++)
    {
    for (x=right_margin; x>left_column; x--)
      strncpy( display[y][x], display[y][x-1],
               sizeof(character_t));
    display[y][left_column].code = EMPTY_CHARACTER;
    }
  count -= 1;
  }
} /* end scroll_right */
```

5.15 CONTROL FUNCTION REFERENCE TO OTHER CHAPTERS

This subsection lists ANSI control functions that are not described in this chapter (DEC STD 70-5, Character Cell Display) and indicates where in the VSRM they are described as a reference to the rest of ANSI host interface.

Control Function Reference

- Terminal Management Functions
  - DA - Device Attributes
  - DECRPTUI - Report Terminal Unit ID
  - DSR - Device Status Report
  - DECID - Identify Device
  - DECSCL - Select Conformance Level
  - DECSR - Secure Reset
  - DECSRC - Secure Reset Confirmation
  - DECSTR - Soft Terminal Reset
  - DECSTUI - Set Terminal Unit ID (Restricted)
  - RIS - Reset to Initial state
- Code Extension Functions
  - Announce Subset of Code Extension Facilities
  - Locking Shifts: LS0, LS1, LS2, LS3, LS1R, LS2R, LS3R
  - Single Shifts: SS2, SS3
  - S7C1T - Select 7-bit C1 Transmission
  - S8C1T - Select 8-bit C1 Transmission
- Keyboard Processing Functions
  - DECARM - Autorepeat Mode
  - DECBKM - Backarrow Key Mode
  - DECCKM - Cursor Keys Mode
  - DECKBUM - Keyboard Usage Mode
  - DECKPAM - Keypad Application Mode
  - DECKPNM - Keypad Numeric Mode
  - DECNKM - Numeric Keypad Mode
  - DSR - Device Status Report (keyboard status)
  - KAM - Keyboard Action Mode
  - DECLFC - Local Functions Control
  - DECLFKC - Local Function Key Control
  - DECSMKR - Select Modifier Key Reporting
  - DECKPM - Key Position Mode
  - DECEKBD - Extended Keyboard Report
- Terminal Synchronization
  - XON
  - XOFF
  - BREAK
  - DECXRLM - Transmit Rate Limiting Mode
- Extensions
  - DRCS
    - DECDLD - Downline Load
  - UDK
    - DECUDK - User Defined Keys

```
|         DSR    - Device Status Report (UDK lock)
|       Session Management Extension
|         DECES  - Enable Sessions
|       Printer Port
|         DECPEX -  Print Extent Mode
|         DECPFF -  Print Form Feed Mode
|         DSR    -  Device Status Report (printer port)
|         MC     - Media Copy
|       Status Display
|         DECSASD - Select Active Status Display
|         DECSSDT - Select Status Display Type
|     Documented Exceptions
|       DECANM - ANSI/VT52 Mode
|       DECALN - Screen Alignment
|       DECAWM - Autowrap Mode
|       DECTST - Invoke Confidence Test
|       CRM    - Control Representation Mode
```

5.16 CHANGE HISTORY

5.16.1 Rev 0.4 To 0.5

The following changes were made at this revision to properly integrate this section with the rest of the Video Systems Reference Manual:

1. The name was changed from "Character Cell Display Level 1" to "Character Cell Display". All references to Level 1 operation only were removed, and explicit reference to both Level 1 and Level 2 operation was added for all functions.

2. The Reference Standards section was updated.

3. The Conformance section, the list of conforming products, and the tables of required functions were removed and made a separate chapter in the Video SRM ("Conformance Requirements").

4. The program flow diagrams and the executive routines were moved to the chapter "Specification Program Structure".

5. The Reset to Initial State control was moved to the chapter "Terminal Management".

6. Some variable names were changed to be consistent with their use throughout the rest of the Video SRM.

7. The algorithm for Insert or Replace Character was corrected to include the processing of Single Shift control functions.

8. The Set Mode and Reset Mode control functions were moved to the chapter "Specification Program Structure", and their detailed implementation was broken out into a separate section for each mode related to Character Cell Display operation.

9. The description of Blink rendition was changed.

10. A note was added concerning software dependence on the speed of implementing changes of state in Screen Mode.

11. The purpose section for Line Feed, Vertical Tab, Form Feed, Index, Reverse Index, and Next Line was reworded to indicate scrolling if necessary, and to indicate (where appropriate) the use of fallback implementation in video terminals.

12. The section on Substitute was reworded to remove the reference to "fallback" character.

13. The note on software use of Horizontal and Vertical Position was changed to remove the reference to retransmission.

14. A definition was added for the term "Margin".

15. Beginning of line and end of line conditions were added to the Erase In Display algorithm.

16. Clearing of the Last Column Flag was added to the algorithms for Delete Character and Reset Auto Wrap Mode.

17. A note was added to the section on slow scrolling to indicate that it must render text "recognizable", but not necessarily "readable".

18. A note was added on conforming software use of Double Height lines.

19. A section was added for Character Set Selection, which was previously a part of the chapter "Code Extension Layer".

20. A section was added for the Warning Bell, which was previously a part of the "Terminal Management" chapter.

21. A section was added for Text Cursor Enable Mode, which was previously a part of the "Terminal Management" chapter.

22. A section was added for Control Representation Mode, which is currently planned for Level 2 terminals (subject to review).

5.16.2 Revision 0.5 To AX10

1. Added the Selectively Erasable Characters extension, which required the following:

   1. A state description for Character Attributes.

   2. New entries in the state tables.

   3. Addition of the Select Character Attribute (DECSCA) control function.

   4. Modification of the Erase In Line (EL) and Erase In Display (ED) control functions.

   5. Addition of the Selective Erase In Line (DECSEL) and Selective Erase In Display (DECSED) control functions.

   It also required changes to the following routines:

   Insert or Replace Graphic Character
   Save Cursor
   Restore Cursor

2. Removed Control Representation Mode (CRM) to an appendix to the SRM. It will not be invokable from the host.

3. Added the Erase Character (ECH) and Insert Character (ICH) control functions.

4. Added a note to the description of Auto Wrap Mode to indicate that Space characters also affect the wrap condition.

5. Added the following control functions to the list of functions which clear the Auto Wrap state (Last Column Flag):

   Erase Character
   Insert Character
   Erase In Line
   Selective Erase In Line
   Erase In Display
   Selective Erase In Display

6. Added a note that the Current Attribute is saved in the Cursor Save Buffer if the Selectively Erasable Characters extension is supported.

7. Added values to the Select Graphic Rendition (SGR) control function for Level 2 to turn off Bold, Blink,

Underscore, and Reverse renditions.

8. Made minor corrections to the algorithms for Double Width and Double Height lines.

9. Added a note to the Substitute control function referring to the section "Code Extension Layer" for the handling of Substitute within escape and control sequences.

10. Added a deviation note to the Index control function for the VT100 and VT125 to indicate that in these terminals the control is affected by the setting of New Line Mode.

11. Changed the algorithm for Tabulation Clear (TBC) to process multiple selective parameters.

12. Made extensive changes to the algorithms for Erase In Line (EL) and Erase In Display (ED) to provide the following:
    - o processing of multiple selective parameters.
    - o setting of line rendition to single width when all characters are erased.
    - o processing of private parameter for selective erase function.

13. Corrected the coding of Delete Character (DCH), which is CSI Pn P.

14. Added a note on the use of User Preference Features (modes) which are not to be used by software except in response to an explicit user request.

15. Made notes that setting and resetting Column Mode (DECCOLM) will clear the screen, etc., even if the terminal was already in the selected state.

16. Noted that the Warning Bell may be disabled in Setup.

17. Modified the descriptions of Vertical Tab (VT) and Form Feed (FF) to point to the Line Feed algorithm.

18. Made Insert Replacement Mode (IRM), Delete Character (DCH), Delete Line (DL), and Insert Line (IL) a Level 1 Editing Extension, which is required in all future Level 1 implementations, and in Level 2. Removed all VT100 family deviation notes on these functions.

19. Made Auto Wrap Mode an exception to Level 1, and removed all references to this and the Last Column Flag. Also removed all VT100 family deviation notes on this function.

20. Modified the Save Cursor (DECSC) and Restore Cursor (DECRC) descriptions to differentiate between Level 1 and Level 2 functionality.

21. Added a character set table to the state descriptions and modified the algorithms for Designate Character Set to accommodate redefinition of the designator sequences using DRCS.

22. Corrected dual coding in the algorithm for Insert or Replace Graphic Characters.

23. Added a note to the Restore Cursor control indicating the handling of the condition in which the cursor is restored outside of the scrolling region with Origin Mode set.

24. Made corrections in the algorithms for Erase Character, Delete Character, Insert Character, Delete Line, and Insert Line.

25. Added a note on the use of the UK character set in Level 1 operation to the section on character set designation.

5.16.3 Rev AX10 To AX11

1. Removed Rev AX10 change bars. Added change bars to any change that could affect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

2. Added note that level 2 SGR parameters may be recognized when device is operating in level 1 mode, but that conforming software shall not rely on this feature.

3. Corrected sense of Selectively Erasable Attribute (set using DECSCA). The default or attribute off condition is that characters are selectively erasable.

5.16.4 Rev AX11 To AX12

1. Removed Rev AX11 change bars. Added change bars to any change that could affect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

2. Updated Reference Standards section and put at end after Change History per convention for other chapters.

3. Added section to Introduction explaining chapter organization as a number of small sections. Re-organized entire chapter to group related controls by function instead of syntax.

4. Modified all code to operate on the current page.

Terminology Section:

5. Removed definitions of the following terms which are defined in the Code Extension section:
   - Bit Combination
   - Character
   - Control Character
   - Control Function
   - Control Sequence
   - Control String
   - Escape Character (ESC)
   - Escape Sequence
   - Final Character
   - Intermediate Character
   - Numeric Parameter
   - Parameter
   - Parameter String
   - Selective Parameter

6. Added definition of Logical Display - the lines and columns available for storing and presenting graphic characters as seen by the application or host computer. A Logical Display may be organized as a single rectangular array of lines and columns (one page), or divided into a number of identically sized pages, each of which is a rectangular array of lines and columns.

7. Added definition of Page - An addressable area of a Logical Display organized as a rectangular array of lines and columns. The number of lines and columns defines the Page Size.

8. Added subsection titled "Code Extension Terms" listing terms defined in the Code Extension chapter of the VSRM (DEC STD 70-3).

9. Removed the following terms which do not appear in the text of the chapter: Introducer; Transfer; Transmission Control.

10. Clarified definition of Control Sequence Introducer (CSI) - The C1 control (represented by a single character in 8-bits, or a two character ESC Fe sequence in 7-bits) which initiates a control sequence. CSI is a prefix affecting the interpretation of a limited number of contiguous bit combinations. [This should be in the Code Extension chapter as well, but isn't at this time]

11. Clarified definition of Default - A function-dependent value that is assumed when no explicit value, or a value of 0, is specified.

12. Clarified definition of Margin - A line which marks the upper or lower boundary of the scrolling region (vertical scroll margin). A column which marks the left or right boundary of the scrolling region (horizontal scroll margin).

Display Coordinate System and Addressing

13. Logical Display: Described concept of Logical Display and Page Size, default sizes, and ability to change it.

14. Margins and Scrolling: Re-organized text to combine syntax and semantics to be easier to follow. Added section summarizing interaction of controls with scrolling region. Added list of controls defined in this section. Added descriptions for: DECSLRM, DECLRMM, DECFI, DECBI to support left and right margins.

15. Cursor Movement: Added DECXCPR extended cursor position report for use at Level 4, or with the Windowing Extension. Updated algorithms for CUF and CUB to check right and left margins.

Page Size and Arrangement:

16. Added description of DECSCPP and DECSLPP to support Level 4, or Windowing Extension.

Page Movement:

17. New Subsection describing page movement controls for use at Level 4, or with the Windowing Extension.

Windowing Extension:

18. Completely new subsection describing Windowing Extension.

Visual Renditions

19. Added new subsection on ANSI Color Text Extension. SGR parameters 31-37, 39, 41-47, 49. Color Maps. DECCTR - color table report. Interactions with other visual attributes. Guidelines for default color assignment. DECSTGLT - Select Text Graphics Look-Up Table (for alternate text rendition mapping).

Graphic Character Sets

20. Expanded section on Grahic Character sets including which sets are required at different conformance levels and extensions. Designating sequences for character sets. 8-bit Interface Architecture Extension. NRCS Extension.

Editing Functions:

21. Modified description of character insertion to cover behavior with respect to the margins.

22. Modified description of ICH, IL, and DL to be restriced to text within the Left Right Margins.

23. Added descriptions of DECIC and DECDC to support left and right margins.

24. Noted ECH, EL, DECSEL, ED, and DECSED are not affected by the margins.

OLTP Features

25. Completely new subsection describing VT420 OLTP features.

Saving and Restoring Terminal State

26. Terminal State Interrogation: Completely new subsection describing TSI

27. Modified note on DECCIR (Cursor Information Report) to specify sequence returns UPSS designator if G-set was designated that way.

Internal Functions and Procedures

28. Modified routines scroll_up and scroll_down to operate within the left and right margins.

29. Added routines scroll_left and scroll_right.

Control Function Reference to Other Chapters

30. Completely new subsection listing control functions and what chapter of the VSRM they are described in.

5.17 REFERENCE STANDARDS

DIGITAL STANDARDS

EL-00138-00	DEC STD 138 Registry of Control Functions for Character-Imaging Devices
EL-00169-00	DEC STD 169 Digital Standard Coded Graphic Character Sets for Hardware and Software

Copies of Digital Standards Can be obtained from Standards and Methods Control, CTS1-2/D4, DTN 287-3724, JOKUR::SMC

ANSI AND ISO STANDARDS

ANSI X3.4 - 1986	American National Standard Code for Information Interchange (ASCII character set)
ANSI X3.41 - 1974	American National Standard Code Extension Techniques for use with the 7-Bit Coded Character Set of the American National Standard Code for Information Interchange
ANSI X3.64 - 1979	Additional Controls for use with American National Standard Code for Information Interchange
dpANS X3.134.1-1985	8-bit ASCII Structure and Rules
dpANS X3.134.2-1985	7-bit and 8-bit ASCII Supplemental Multinational Graphic Character Set
ISO 646	7-Bit Coded Character Set for Information Interchange
ISO 2022:1986	Information Processing / ISO 7-Bit and 8-Bit Coded Character Sets - Code Extension Techniques
ISO 6429:1988	Information Processing / Additional Controls for use with Character Imaging Devices
ISO 8859-1:1987	Information Processing / 8-Bit Single-Byte Coded Character Sets - Part 1 : ISO Latin Alphabet Nr 1;

Copies of ANSI and ISO Standards can be obtained from local Digital Libraries.

Section Index

DEC STD 070-6 VIDEO SYSTEMS REFERENCE MANUAL -

KEYBOARD PROCESSING

Document Identifier: A-DS-EL00070-06-0 Rev A, 19-Apr-1988

ABSTRACT: This specification describes the interfaces to terminals using the Digital Standard Keyboard of 1980 (Level 1) and the corporate standard keyboard (LK201 and future) (Level 2 and Level 3). It specifies both the coding interface between the application program and the terminal and the human interface between the terminal user and the keyboard. Furthermore, the same coding interface is provided to the application program whether it is running in a host computer or inside a personal computer or workstation, thereby achieving transportability of application programs.

APPLICABILITY: Mandatory for engineers designing hardware for terminal products and software engineers designing programs using terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria".

STATUS: APPROVED 19-Apr-1988; see EL-INDEX-00 for expiration date.

The material contained within this document is assumed to define mandatory standards unless it is clearly marked as: (a) not mandatory, or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements.

TITLE: DEC STD 070-6 VIDEO SYSTEMS REFERENCE MANUAL - KEYBOARD PROCESSING

DOCUMENT IDENTIFIER: A-DS-EL00070-06-0 Rev A, 19-Apr-1988

REVISION HISTORY:

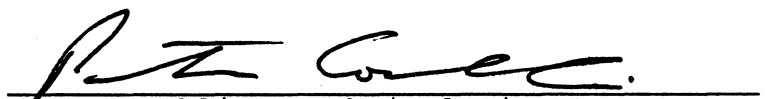
Original Draft	04-Sep-1982
Revision 0.1	09-Nov-1982
Revision 0.2	14-Jan-1983
Revision AX01	28-Feb-1983
Revision AX11	18-Mar-1985
Revision AX12	19-Feb-1988
Revision A	19-Apr-1988

FILES:

User Documentation	EL070S6.mem
Internal Documentation	EL070S6.rno
	EL070S6.rnt
	EL070S6.rnx

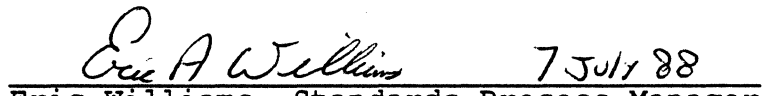
Document Management Category:	Terminal Interface Architecture (STI)
Responsible Department:	DSG Terminals Architecture
Responsible Person:	Peter Sichel
Authors:	Peter Sichel, Ram Sudama
SMC Writer:	Patricia Winner

APPROVAL: This document, prepared by the Desktop Systems Group, has been reviewed and recommended for approval by the General Review Group for its category for use throughout Digital.



Peter Conklin, Technical Director,
Desktop Systems Group

Peter Sichel, Desktop Systems Group



Eric Williams, Standards Process Manager

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, VIDEO::TERMARCH

Copies of this document can be ordered from Standards and Methods Control, CTS1-2/D4, DTN 287-3724, JOKUR::SMC

Please provide your name, badge number, cost center, mailstop, and ENET node when ordering.

CONTENTS

6.1 INTRODUCTION

6.1.1 Keyboard Overview

The terminal interface described in this specification applies specifically to handling of the LK201 corporate standard keyboard and all future corporate standard keyboards.

Logically, the keyboard is part of the terminal; however, it is physically and electrically packaged separately. This specification does not specify the interface between the keyboard and the rest of the terminal. That interface is described in full in the LK201 Functional Specification.

This specification describes the keyboard layouts for two different product classes:

1. LK201- { Ax, Ex, Nx } DEC Standard Keyboard
2. LK201- { Bx, Fx, Px } DECmate/WPS

The physical layout is the same; only the key legends and the Permanent Label Strip differ between the product classes.

Keyboards operate in one of two usage modes: typewriter or data processing which changes the meaning of a small number of keys as indicated by dual legends on the key cap. Some keyboards are used for both typewriter and data processing with the same legends. Differences between these modes are examined in detail in the tables of the Main Key Array section.

6.1.1.1 Keyboard Product Classes -

The Digital Standard keyboard product class includes the LK201- {Ax, Ex, Nx}. The "E" designation, as in LK201- Ex, is used because letter variations for "Ax" were exhausted. The "Ex" subfamily consists of upgraded keyboards which have a predecessor in the "Ax" subfamily and also keyboards which are entirely new and have no predecessor in the "Ax" subfamily. Examples of the former are the LK201- ED, LK201- AD Danish Keyboard and the LK201- EN, LK201- AN Norwegian Keyboard. Examples of keyboards with no predecessor in the "Ax" subfamily are the LK201- EH, Greek, and the LK201- ER Arabic Keyboards.

The LK201- Nx subfamily is for the VT300 series terminals. The LK201- NA is unique in that it has front legends on the numeric and editing keypads that other LK201 keyboards do not. In detail, the LK201- {NA, NF, NH} are to be used with the VT330/340 series and the LK201- {NG, NM, NX} are to be used with the VT320 series terminals.

The DECmate/WPS keyboard product class includes the LK201- {Bx, Fx, Px}. The "F" designation, as in LK201- Fx, is used because letter variations for "Bx" were exhausted. The LK201- Px is used with the VT300 series terminals.

6.1.1.2 Keyboard Versions And Dialects -

The LK201 is manufactured in a number of versions to accomodate local language and application specific requirements. These versions differ only in the labeling of keys and visual indicators. To the terminal system all the keyboards will appear identical. With the exception of the Main Key Array, the codes transmitted by the terminal for all key positions are the same for all versions of the keyboard and the only differences are the languages of the legends and label strips.

Digital terminals support a number of keyboard layouts corresponding to different keyboard versions. Within the terminal, these different layouts are actually different mappings from keystrokes on the main key array to character codes produced, and are called "Keyboard Dialects". The terminal user selects the desired Keyboard Dialect in Set-Up, usually corresponding to the keyboard version. It is this selection of Keyboard Dialect that controls the characters produced for a given series of keystrokes, not the physical keyboard layout. The names of the Keyboard Dialects are based on the geographical region, country, or language for which the keyboard is primarily intended.

The following is a list of the Keyboard Dialects and corresponding keyboard versions or layouts defined within this specification. Some of the dialects have changed to accomodate more recent country requirements. The most recent or recommended variation within each dialect is listed first.

Keyboard Dialect	Version	
North American	LK201-EE	Combined U.S./U.K. (VT320, VT220CR)
	LK201-NA	Combined U.S./U.K. (VT330/VT340 with local editing legends)
	LK201-AA	1st U.S. (VT200)
British	LK201-EE	Combined U.S./U.K.
	LK201-AE	1st U.K. (VT200)
Flemish (Belgium)	LK201-AB	Same as Belgian/French (LK201-AP)
Canadian (French)	LK201-AC	
Danish	LK201-ED	2nd Danish (VT300)
	LK201-AD	1st Danish (VT200)
Finnish	LK201-NX	3rd Finnish (VT320)
	LK201-NF	2nd Finnish (VT330/VT340)
	LK201-AF	1st Finnish (VT200)
		Same as 1st Swedish (LK201-AM)
Austrian/German	LK201-NG	2nd Austrian/German (VT300)
	LK201-AG	1st Austrian/German
Dutch	LK201-NH	2nd Dutch (VT320)
		Same as North American (LK201-EE)
	LK201-AH	1st Dutch
Italian	LK201-AI	
Swiss (French)	LK201-AK	
Swiss (German)	LK201-AL	
Swedish	LK201-NM	2nd Swedish (VT300)
	LK201-AM	1st Swedish
		Same as 1st Finnish (LK201-AF)
Norwegian	LK201-EN	2nd Norwegian (VT300)
	LK201-AN	1st Norwegian
Belgian/French	LK201-AP	Same as Flemish (LK201-AB)
Spanish	LK201-AS	
Portuguese	LK201-AV	

Notes:

1. Many keyboards can be ordered with or without country power cords (to form "country kits"). The following keyboard series are generally equivalent:

W/PWR Cord		Keyboard Alone
Ax	=	Lx
Bx	=	Mx
Ex	=	Rx

   This specification uses the keyboard with power cord designations (most common on video terminals) wherever possible.

2. WPS or Gold keyboard layouts add gold legends to the alpha-numeric layouts above. Gold layouts corresponding to the keyboard versions above, where -Ax is regular, and -Bx is WPS, are:

   -BA to AA, -BB to AB, -BC to AC,
   -BD to AD, -BE to AE, -BF to AF,
   -BG to AG, -BH to AH, -BI to AI,
   -BK to AK, -BL to AL, -BM to AM,
   -BN to AN, -BP to AP, -BS to AS,
   -PE to EE, -PA to NA.

6.1.2 Scope

This specification defines the coding interface between terminals and software application processes and the logical part of the human interface between the keyboard and the terminal user.

The coding interface is defined to be the hardware or software interface across which coded control functions and graphic characters are passed. The same coding interface shall exist:

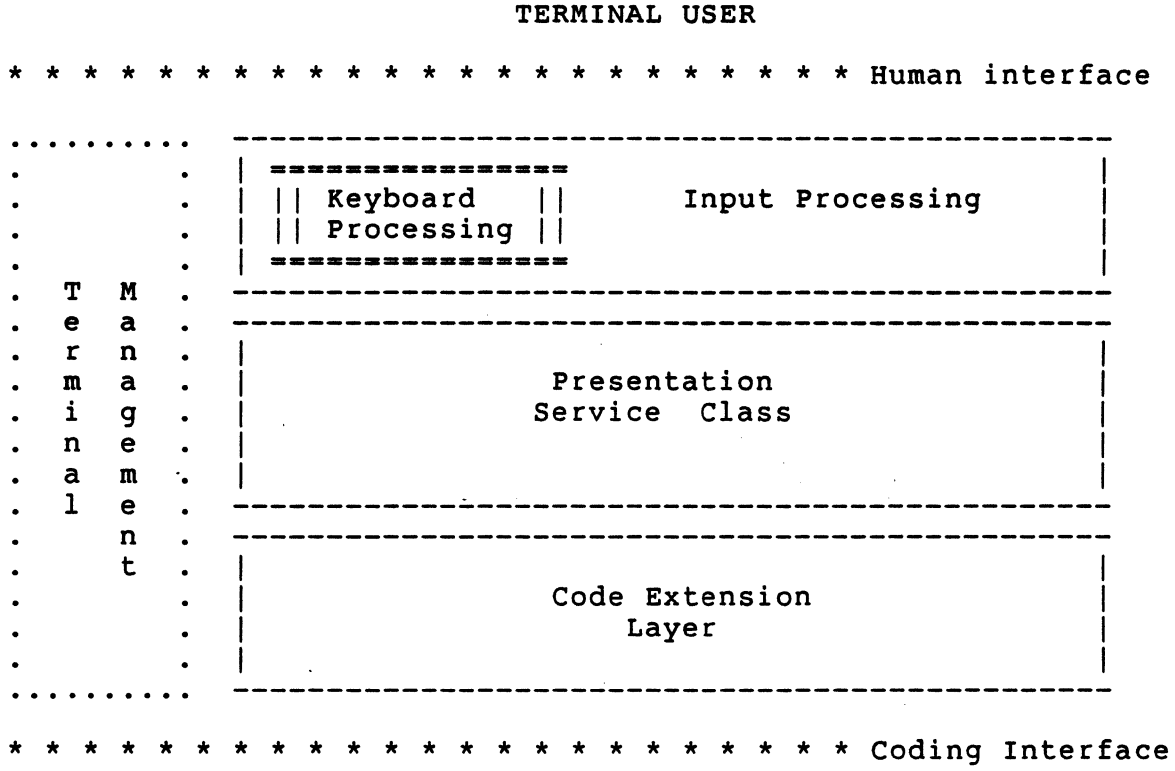
1. Between a fixed function terminal and a host computer.
2. Between a personal computer/workstation simulating a fixed function terminal and a host computer.
3. Between a personal computer/workstation and an application program running inside the personal computer/workstation.

By making the coding interface the same in the above three situations, transportability of application programs is possible.

This specification also defines the logical part of the interface between the keyboard and the terminal user not covered in any section of DEC STD 107 Digital Standard for Terminal Keyboards and its revisions, specifically the operation of the Compose key and Non-Spacing Diacritical keys for all national keyboards.

6.1.3 Relationship To TIA

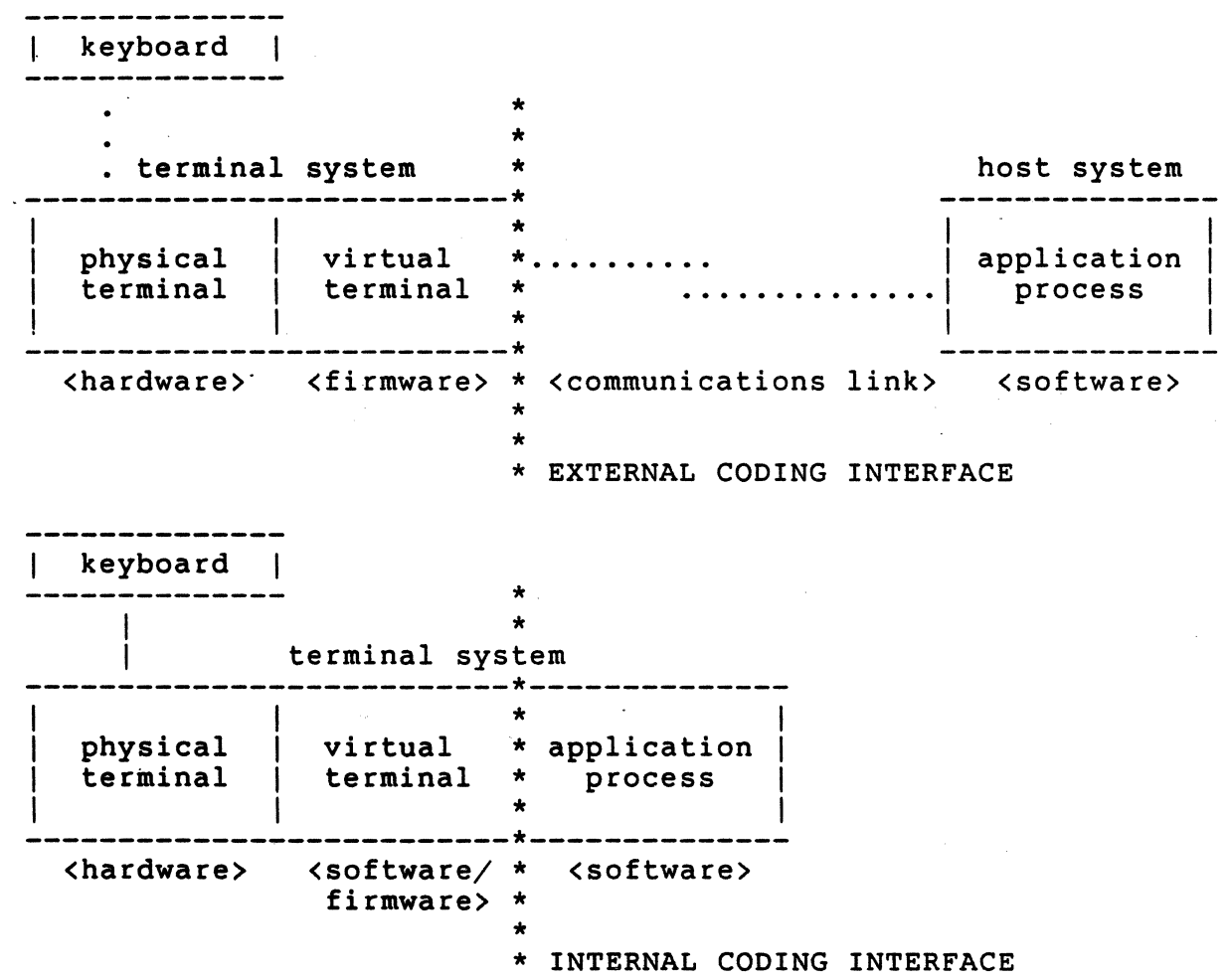
The Keyboard Processing interface forms a service class of the Input Processing layer of the Terminal Interface Architecture. It is backwards compatible at the coding interface with the Level 1 Keyboard previously described in DEC STD 107-0, of which it is a complete super set.



Structuring of the Terminal Interface Architecture

6.1.4 Coding Interface

The coding interface defined within this specification applies to both internal and external product interfaces. External interfaces are interfaces between a terminal, personal computer or workstation and a remote system. Internal interfaces are interfaces between a terminal sub-system and software application processes running within a personal computer or workstation. An application process consists of the following software components: terminal driver, operating system, layered application subroutine package, and the application program. This specification does not deal with which of these components convert the coded information to other forms, if any, before it is processed by the application program.



External and Internal Terminal Coding Interfaces

6.2 CONFORMANCE REQUIREMENTS

All conforming terminal keyboard implementations will provide at least two compatibility levels, or modes of operation (part of terminal conformance level). Level 1 provides backward compatibility with previous keyboards designed in conformance with DEC STD 107 (VT100, LA36, and so on) and provides the 7-bit ASCII control and graphic characters. Level 2 is a compatible super set of Level 1 and provides the 8-bit DEC STD 169 Multinational Character Set, the Editing Keypad, and the Application Function Keys. Level 3 is a compatible super set of Level 2 which offers more flexible control over the keyboard encoding (ISO Latin Alphabet Nr. 1, DECKBUM to switch between Typewriter and Data Processing Mode).

A terminal conforming to Level 1 shall be able to transmit to a 7-bit host communication port and may optionally provide a local Set-Up feature to transmit to an 8-bit host port (with the 8th bit always zero). A terminal conforming to Level 2 shall provide a local Set-Up feature to transmit to a 7-bit or 8-bit host communication port.

A terminal conforming to Level 2 shall be able to switch to Level 1 operation. When a Level 2 terminal is operating in Level 2 mode and 7-bit host communication port mode, only a subset of Level 2 functionality shall be transmitted.

6.2.1 Level 1 Conformance Requirements

Terminals, workstations, and personal computers conforming to Level 1:

a. Shall implement all keys indicated in Level 1 Operation.

b. Shall neither implement additional keys nor make operative keys that are not part of Level 1 Operation.

c. Shall make inoperative all keys that are part of Level 2 operation only.

d. Shall implement the following control functions for receipt from software application processes and as a local Set-Up feature:

   1. Auto Repeat Mode (DECARM)
   2. Cursor Key Mode (DECCKM)
   3. Keypad Mode (DECKPAM/DECKPNM)
   4. Line Feed/New Line Mode - see the chapter "Character Cell Display"

e. Shall implement the following control function for receipt from software application processes:

   - Keyboard Action Mode (KAM)

f. Shall implement the following local Set-Up features:

   - Caps/Shift Lock Mode as Caps Lock Operation; may optionally implement a local Set-Up feature to change the mode to Shift Lock Operation.
   - May optionally provide means to switch Host Port Environment Mode to 8-bits.

6.2.2 Level 2 Conformance Requirements

Terminals, workstations, and personal computers conforming to Level 2:

a. Shall implement all keys indicated in Level 2 Operation.

b. Shall implement no additional keys.

c. May implement the 8-bit Architecture Extension.

d. May implement the NRCS Extension.

e. Shall implement the following control function for receipt from a software application process and as local Set-Up features:

   - All Level 1 control functions.

f. Shall implement the following control functions for receipt from a software application process:

   1. All Level 1 functions.
   2. Select Conformance Level (DECSCL) to switch between Level 1 Operation and Level 2 Operation.

g. Shall implement the following local Set-Up features:

   1. Keyclick Mode.
   2. Caps/Shift Lock Mode. The terminal user shall be able to set the Caps/Shift Lock Mode to either Caps Lock Operation or Shift Lock Operation. The factory default shall be Caps Lock Operation.
   3. May optionally implement means to disable the "warning bell", or "margin bell" indicator as a Level 2 extension.
   4. May optionally implement additional local Set-Up features not specified by the architecture, for example, volume of bell, volume of keyclick.

6.2.3 Level 3 Conformance Requirements

All Level 2 conformance requirements plus the following:

a. Shall implement the 8-bit Architecture Extension.

b. Shall implement the following control function for receipt from a software application process and as local Set-Up features:

   - Keyboard Usage Mode (DECKBUM) to select between Typewriter and Data Processing keys.

c. Shall implement the following control function for receipt from a software application process:

   - Select Conformance Level (DECSCL) to switch between Level 1 Operation, Level 2 Operation, and Level 3 Operation.

d. May implement the Keypad Compose mechanism for "Composing Arbitrary 8-bit Characters" described later in this document.

6.2.4 Level 1 Operation

For purposes of conformance, inoperative means that the key shall not transmit anything, and shall not keyclick.

When Level 1 operation is selected, the following restrictions shall apply:

a. Only 7-bit ASCII character codes shall be transmitted with the 8th bit omitted or zero depending on whether the host port environment is set to 7-bits versus 8-bits respectively. Keys which would normally produce an 8-bit code with the 8th bit set shall be inoperative.

b. The Editing Keypad keys with generic names E1, E2, E3, E4, E5, and E6 are inoperative.

c. The Function keys with generic names F6, F7, F8, F9, and F10 are inoperative.

d. Function key with generic name F11 shall transmit the ESC code (1/11).

e. Function key with generic name F12 shall transmit the BS code (0/8).

f. Function key with generic name F13 shall transmit the LF code (0/10).

g. Function keys with generic names F14, F15, F16, F17, F18, F19, and F20 are inoperative.

h. The only valid Compose sequences shall be those which generate 7-bit ASCII character codes. Those sequences that would produce an 8-bit code with the 8th bit set, shall not transmit anything, and shall echo the error bell after the final key stroke of the sequence.

6.2.5 Level 2 And Level 3 Operation

When Level 2 or Level 3 operation is selected and the Host Port Environment Mode is set to 8-bits, the following conditions shall apply:

a. All keyboard keys are live.

b. The Function keys with generic names F11, F12, and F13 transmit their designated control sequences (i.e., ESC, BS, and LF can only be generated as control character combinations from the main key array).

c. All valid 7-bit and 8-bit Compose sequences may be typed by the terminal user.

d. No additional Compose sequences for existing or new characters may be typed by the terminal user.

When Level 2 or Level 3 operation is selected and Host Port Environment Mode is set to 7-bits, the following conditions shall apply:

- Shall make inoperative all explicit or implied compose sequences that would generate an 8-bit code with the 8th bit one.

- Shall transmit all explicit or implied compose sequences that would generate an 8-bit code with the 8th bit zero.

6.2.6 Keyboard Character Encoding

In Level 1 operation the keyboard shall transmit ASCII from GL, no supplemental characters are available.

In Level 2 operation the keyboard shall transmit ASCII from GL and the DEC Supplemental Set from GR.

If the Level 2 8-Bit Architecture Extension is supported (refer to DEC STD 070-3 Code Extension Layer), the keyboard shall transmit ASCII from GL and the User Preference Supplemental Set (UPSS, DEC Supplemental or ISO Latin-1) from GR (same as Level 3 operation below).

In Level 3 operation the keyboard shall transmit ASCII from GL and the User Preference Supplemental Set (UPSS, DEC Supplemental or ISO Latin-1) from GR.

Level 3 terminals, and Level 2 terminals with the 8-bit Architecture Extension, support at least two different 8-bit multinational character encodings. Level 2 and Level 3 terminals with the NRCS Extension also support a number of 7-bit country specific (National) character encodings. At any time, only one of these character encodings is in use. No character set shift functions (SO, SI, etc.) will be transmitted as a side effect of typing a keyboard key (except if ctrl-N or ctrl-O are explicitly typed). The user can choose one of the following keyboard encodings via Set-Up:

- DEC Multinational (factory default)
  SET-UP: 8-bit Characters, UPSS DEC Multinational
  Primary keyboard set is ASCII
  Supplemental keyboard set is DEC Supplemental

- ISO Latin Alphabet Nr 1
  SET-UP: 8-bit Characters, UPSS ISO Latin-1
  Primary keyboard set is ASCII
  Supplemental keyboard set is ISO Latin-1 supplemental

- 7-bit ASCII character set
  SET-UP: 7-bit Characters, Data Processing Keys
  Primary keyboard set is ASCII
  No supplemental characters available

- 7-bit National Replacement Character Set (NRCS)
  SET-UP: 7-bit Characters, Typewriter Keys
  Primary keyboard set is one of 12 NRCS
  No supplemental characters available

The selection of a particular 7-Bit NRCS is bound to the selection of the keyboard dialect. See guideline under heading "Keyboard Dialect". In 7-bit Characters mode, choosing Data Processing Keys always uses ASCII as the primary keyboard set.

Deviation Note

Older Level 2 terminals with the NRCS Extension did not always use ASCII as the primary keyboard set when Data Processing Keys was selected (prior to the VT220CR). This use of a 7-bit NRCS other than ASCII in 7-bit Characters Data Processing mode has been deprecated.

Explanatory Note

The character set mode, 7-bit/8-bit Characters corresponds to National/Multinational Mode (NRCS Extension) on the VT200 but with two simplifications. 8-bit Characters are temporarily disabled when in "VT100 mode", or a "7-bit host line" is selected.

In practice, this means the terminal no longer supports ASCII with the Typewriter Keys layout except on the North American Keyboard. This makes sense. The typewriter layout has no inherent advantage when using ASCII, it simply has more dead keys. Confusing distinctions like VT100 Multinational mode have been eliminated.

The remainder of this specification documents keyboard behavior with both the 8-bit Architecture and NRCS Extensions.

6.3 REFERENCED STANDARDS AND RELATED PUBLICATIONS

DIGITAL STANDARDS

DEC STD 070-3	Code Extension Layer
DEC STD 070-4	Terminal Management
DEC STD 070-7	Printer Port Extension
DEC STD 070-12	Terminal Synchronization
EL-00107-00	DEC STD 107-0 Digital Standard for Terminal Keyboards Standard Keyboard Layouts
EL-00138-00	DEC STD 138-0 Registry of Control Functions for Character-Imaging Devices
EL-00169-00	DEC STD 169-0 Digital Standard Coded Graphic Character Sets for Hardware and Software
A-SP-LK200-A-0	LK200 Functional Specification
A-SP-LK201-A-2	LK201 Keyboard Design Specification

Copies of Digital Standards Can be obtained from Standards and Methods Control, CTS1-2/D4, DTN 287-3724, JOKUR::SMC

Please provide your name, badge number, cost center, mailstop, and ENET node when ordering.

ANSI AND ISO STANDARDS

ANSI X3.4 - 1986	American National Standard Code for Information Interchange (ASCII character set)
ANSI X3.41 - 1974	American National Standard Code Extension Techniques for use with the 7-Bit Coded Character Set of the American National Standard Code for Information Interchange
ANSI X3.64 - 1979	Additional Controls for use with American National Standard Code for Information Interchange
ANSI X4.23 - 1982	Keyboard Arrangement for Alphanumeric Machines
ISO 2022-1982	Information Processing / ISO 7-Bit and 8-Bit Coded Character Sets - Code Extension Techniques

ANSI AND ISO STANDARDS (continued)

ISO 6429-1982	Information Processing / Additional Controls for use with Character Imaging Devices
ISO 8859-1:1987	Information Processing / 8-Bit Single-Byte Coded Character Sets - Part 1 : ISO Latin Alphabet Nr 1

Copies of ANSI and ISO Standards can be obtained from local Digital Libraries.

6.4 TERMINOLOGY

Application Label Strip - A removable label strip associated with an application software package that runs under control of an operating system or runs stand-alone.

Application Process - The collection of software consisting of a terminal driver, an operating system, layered application subroutines or packages, and application programs running inside a personal computer, workstation, or host computer.

Byte - A bit string that is operated on as a unit and whose size is independent of redundancy or framing techniques. In this specification, bytes are 7-bit or 8-bit.

Coded Character (or more simply "Character") - The coded representation of a member of a coded character set used for the organization, control or representation of data. Note: In ISO and ANSI standards, the term "character" does not imply coding. The terms "character" and "coded character" are used interchangeably in this document. The term "legend" refers to the unencoded character.

Coding Interface - A software or hardware interface through which bytes of character-coded information are passed between terminal equipment and an application process across a host port or between terminal equipment and a printer across a printer port.

Compose Sequence - A sequence of two or three keys used to input a single graphic character that may not be available as a single key.

Control Character - A control function; the coded representation of which consists of a single byte.

Control Function - An action that effects the recording, processing, transmission, or interpretation of data and that has a coded representation consisting of one or more bytes.

Non-Spacing Diacritical Key - A key with a diacritical mark as a key legend that is used as the first key of a two-keystroke sequence to create an accented letter.

Diacritical Mark - A symbol used in combination with the letters A through Z or a through z to form an accented letter or umlaut.

Environment - The number of bits (7 or 8) used to transmit or receive data across a coding interface independent of framing or parity across a host or printer port.

Explicit Compose Sequence - A three-key compose sequence starting with the Compose key.

Graphic Character - A character, other than a control function, that has a visual representation normally handwritten, printed, or displayed.

Host Port - The coding interface between a terminal and an application process whether the application process is running inside the terminal or in a host computer.

Implied Compose Sequence - A two-key compose sequence starting with a Non-Spacing Diacritical key.

Inoperative - A key which does not cause the terminal to take any action or modify the effect of other keys. An inoperative key does not transmit any bytes, and does not key click.

Key - The physical part of a keyboard that the terminal user presses to input control and graphic character information.

Key Legend (or more simply "Legend") - The visual symbol embossed or engraved on a key.

Key Position ID - A standard ID that identifies the physical position of a key on the keyboard. Note: taken from ANSI and ISO keyboard standards. See DEC STD 107 which specifies the physical interface between keyboard and the rest of the terminal.

Keyboard Dialect - A mapping from keystrokes on the main key array to character codes produced. A supported keyboard layout (arrangement of keycap labels) corresponding to a keyboard version (manufactured model such as LK201-AA). The names of Keyboard Dialects are based on the geographical region, country, or language for which the keyboard is primarily intended (example: North American).

Keyboard Usage Mode - One of "Typewriter" or "Data Processing". Selects a variation of the keyboard layout intended for office or data processing use. Data processing legends appear to the right of the typewriter legends on those keys where they are different.

Permanent Label Strip - A label strip that is permanently fastened to the keyboard in manufacturing that labels the top row of keys on the keyboard (Local Function Keys and Application Function Key Row).

Printer Port - The coding interface between a terminal and a printer.

Receive - To accept coded character information from an application process sent to a terminal across the coding interface.

Removable Label Strip - A removable label strip that labels the top row of keys on the keyboard (Local Function Keys and Application Function Keys).

Set-Up - A mode of operation of the terminal in which the terminal user communicates directly with the terminal to change certain operational modes. Note: the human interface for Set-Up mode is beyond the scope of this specification; only the Set-Up features required for conformance are specified along with some optional ones.

SPACE - A character that is both 1) a graphic character with a visual representation consisting of the absence of a graphic symbol and 2) a control character that acts as a format effector that causes the active position to be advanced one character position.

System Label Strip - A removable label strip associated with an operating system.

Terminal User - The human user of the terminal (as distinguished from an application process or program).

Transmit - To transfer coded character information from a terminal to an application process across the coding interface.

6.5 PHYSICAL KEYBOARD DESCRIPTION

The keyboard is broken down into six distinct functional, as well as physical, areas. The rest of the specification is organized according to these areas. In the following, the key position IDs are indicated in parentheses:

1. The Main Key Array (E00-E13, D00-D12, C99-C13, B99-B11, A99-A09)

2. The Cursor Keys (B16-B18, C17)

3. The Numeric Keypad (E20-E23, D20-D23, C20-C23, B20-B22, A20-A23)

4. The Editing Keypad (E16-E18, D16-D18)

5. The Application Function Keys (G05-G23)

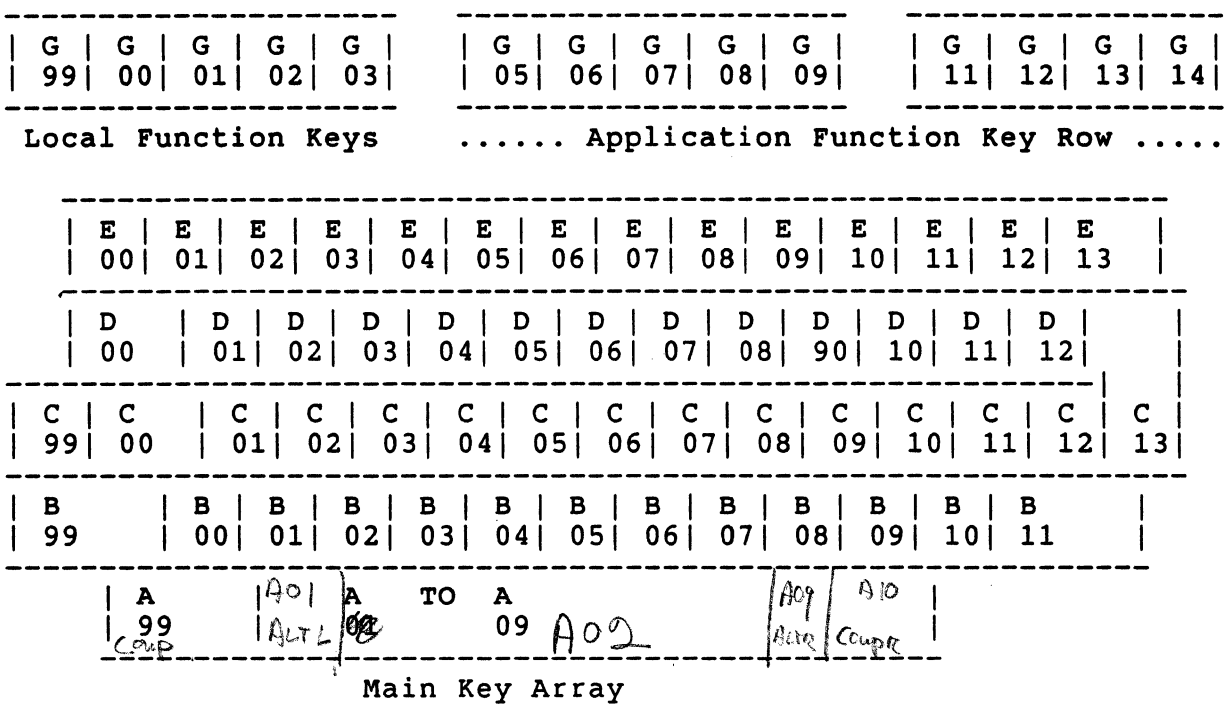
6. The Local Function Keys (G99-G03)

6.5.1 Physical Keyboard Map

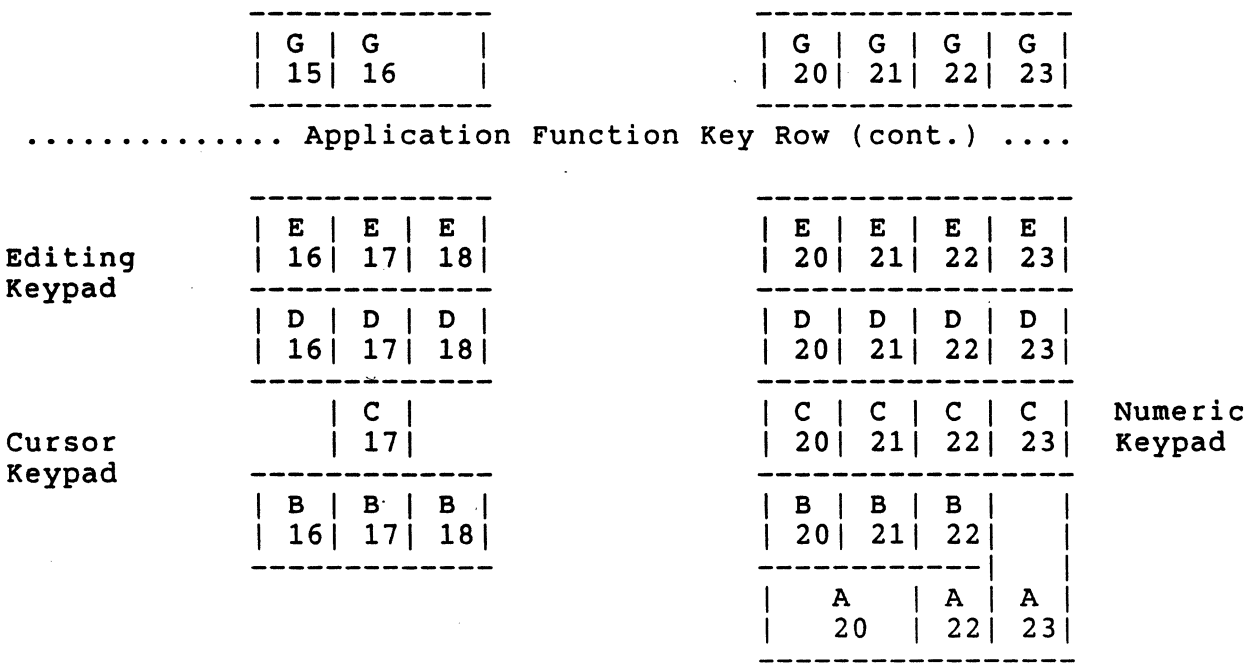
The physical keyboard map provides a means to identify the physical position of keys independent of the keyboard version or operating mode. Each key is given a physical Key Position ID.

Keyboard Map by Physical Key Position ID

Left Half - Keyboard Position Map



Right Half - Keyboard Position Map

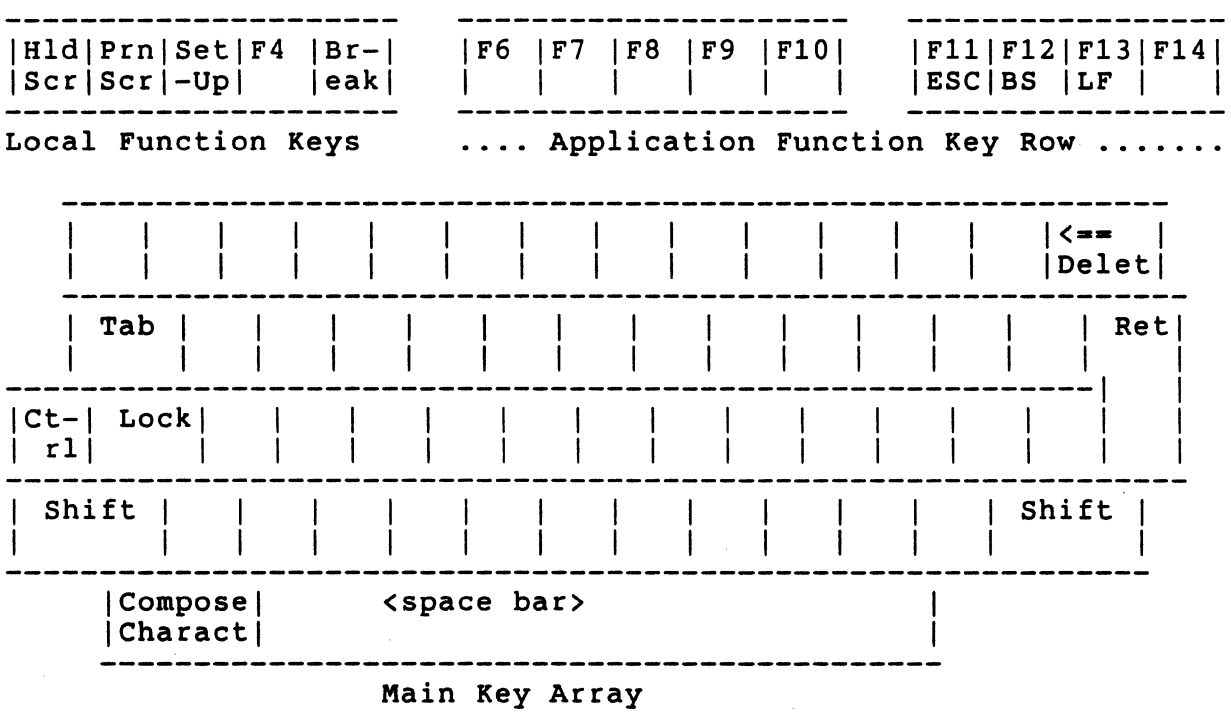


6.5.2 Keyboard Map By Logical Key Name

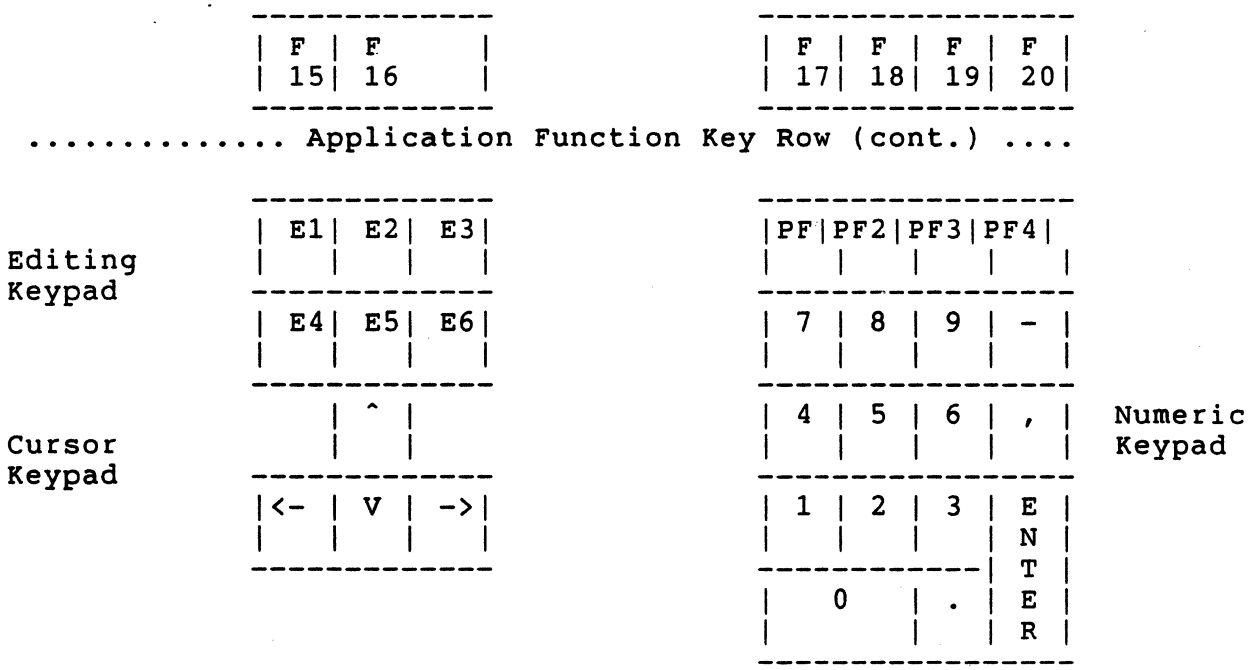
The following maps are provided to relate the description of the keyboard operations described in this specification to the actual codes generated at the coding interface. These maps identify those keys which have explicit names as legends on the key caps or Permanent Label Strip, and leave the alpha-numeric keys blank. The blank keys on the main array all have multiple mappings depending on the Keyboard Dialect and Keyboard Usage Mode. The names provided in this specification are the English equivalents, but on the various national keyboards they are translated into different forms. See DEC STD 107-2 when revised.

On all Permanent Label Strips, the legends above the G15 and G16 keys are replaced with the legends for the four LEDs that are physically located there or are blank for those product classes that do not actuate the LED. The LED legends are Hold Screen, Lock, Compose, and Wait. On all product classes of the keyboard, the Local Function keys and the Application Function Keys are blank, except for the G15 and G16 keys. The following keys have legends that depend on the keyboard product class: G15, G16, E16, E17, E18, D16, D17, D18, all numeric keypad keys.

Left Half - Keyboard Named Key Map



Right Half - Keyboard Named Key Map



6.5.2.1 Application Function Key Labels -

The Application Function Keys do not have any legends on them for any of the keyboard product classes, except for the two keys that have no corresponding room on the label strip for legends because of the LED legends.

Application Function Key Legends

Key Id	DEC Standard	DECmate/WPS
G15	Help	Help
G16	Do	Do

6.5.2.2 Editing Keypad Labels -

The Editing Keypads for the product classes have the following keycap legends:

Editing Keypad Legends

Key Id	DEC Standard	DECmate/WPS
E16	Find	Find
E17	Insert Here	Insert Here
E18	Remove	Remove
D16	Select	Select
D17	Prev Screen	Prev Screen
D18	Next Screen	Next Screen

Editing Keypad Front Legends

(Used for Local Editing Extension on VT330/VT340)

Key Id	LK201- {NA, PA}
E16	Home Cursor
E17	Insert/ Overstrike
E18	CLR PAGE Clr Field
D16	(EDIT)
D17	Prev Page
D18	Next Page

6.5.2.3 Cursor Keypad Legends -

The legends for the Cursor Keypad as shown below:

Cursor Keypad Legends

Key Id	Legend
C17	<upward arrow>
B16	<left arrow>
B17	<down arrow>
B18	<right arrow>

6.5.2.4 Numeric Keypad Legends -

The legends for the Numeric Keypad are shown below:

Numeric Keypad Legends

Key Id	DEC Standard	DECmate/WPS
E20	PF1	<gold>
E21	PF2	Page
E22	PF3	Del Word
E23	PF4	Del Char
D20	7	Sent
D21	8	Tab Pos
D22	9	Underline
D23	- (minus)	Cut
C20	4	Word
C21	5	Para
C22	6	Bold
C23	, (comma)	Paste
B20	1	Back Up
B21	2	Line
B22	3	Upper Case
A20	0	Advance
A22	. (period)	Sel
A23	Enter	Enter <>

Numeric Keypad Front Legends

Key Id	LK201- {NA, PA}
E20	TAB |<--- --->|
E21	Insert Line
E22	Delete Line
E23	Delete Char
C23	(Space) *
A23	Transmit

* The VT330/VT340 provides a Set-Up option which allows C23 to transmit COMMA or SPACE (not mandatory). This is not a local editing feature.

6.5.2.5 Permanent Label Strips -

The following table describes the legends on the permanent label strips:

Permanent Label Strips

Key Id	DEC Standard	DECmate/WPS
G99	F1	Hold Screen
G00	F2	Print Screen
G01	F3	Set-Up
G02	F4	F4
G03	F5	Break
G05	F6	F6
G06	F7	F7
G07	F8	F8
G08	F9	F9
G09	F10	F10
G11	F11	F11 (ESC)
G12	F12	F12 (BS)
G13	F13	F13 (LF)
G14	F14	F14
LED:	Hold Screen	Hold Screen
LED:	Lock	Lock
LED:	Compose	Compose
LED:	Wait	Wait
G20	F17	F17
G21	F18	F18
G22	F19	F19
G23	F20	Hyph Push Hyph Pull

6.5.3 Recommended Labeling For System Label Strips

The following labeling is recommended as a guideline to product implementors in designing System Label Strips, but is not required for conformance:

Key Id	Legend
G99	Hold Screen (or Hold Session)
G00	Print Screen (or Local Print)
G01	Set-Up
G02	F4
G03	Break
G05	Interrupt
G06	Resume
G07	Cancel
G08	Main Screen
G09	Exit
G11	F11 (ESC)
G12	F12 (BS)
G13	F13 (LF)
G14	Additional Options
LED:	Hold Screen (or Hold Session)
LED:	Lock
LED:	Compose
LED:	Wait
G20	F17
G21	F18
G22	F19
G23	F20

Notes

1. G00 is labelled "Local Print" on the VT330/VT340 since these terminals can support two sessions with windows on the screen at one time. Local Print in this case may print the logical display (virtual screen) for one session which is not the same as the physical screen.

2. G02 (F4) is labelled "Switch Session" on the VT330/VT340.

3. G02 (F4) is labelled "Data/Talk" on the VT200 series for use with an integral modem option.

6.6 KEYBOARD OPERATION AND STATE THAT AFFECTS USER INTERACTION

This section describes those functions and state that pertain to all keyboard areas and affect user interaction with the keyboard (but not keyboard encoding). Subsequent sections describe functions and state that pertain to particular keyboard areas or modify the codes transmitted by keystrokes.

The keyboard is sensitive to the following operating modes that affect user interaction:

- Keyboard Action Mode (KAM): locked or unlocked. When locked, all but the local function keys F1-F5 are disabled.

- Preference modes of Auto Repeat, Keyclick, Margin Bell, and Warning Bell. Choices are Enable, or Disable.

6.6.1 Keyboard Output Silo

The terminal shall maintain a silo for keyboard output. In the event that the keyboard produces characters faster than they can be transmitted by the terminal, the silo shall buffer at least 9 keystrokes. Note that function keys may produce up to five bytes per keystroke; while alpha-numeric input will produce only one byte. For the purpose of buffering, a compose sequence will be treated as a single keystroke and shall use only one silo position. Therefore function keys, composed sequences, and alpha-numeric input will each take one silo position per occurrence. (Guideline: the VT320 provides a 25 keystroke silo.)

6.6.2 Keyboard Action Mode

The keyboard shall lock when one of the following conditions occur:

- The keyboard output silo is full.
- Keyboard Action Mode (KAM) is placed in the set state by execution of a SM (SET MODE) control sequence.

When the keyboard is locked, the Wait indicator is turned on, and all keyboard keys except F1, F2, F3, F4 and F5, that is, the local function keys, are disabled. F5, break, may be disabled by other means and is unaffected by setting or resetting KAM. Entering Set-Up unlocks the keyboard for the time the terminal is in Set-Up, then upon exiting Set-Up the keyboard is locked again, unless the terminal user had explicitly executed a control function that unlocks the keyboard.

The keyboard will unlock when one of the following conditions occur and the keyboard output silo is not full:

1. The keyboard output silo has been partially emptied (while KAM is in the reset (unlocked) state).
2. Keyboard Action Mode (KAM) is placed in the reset state by execution of a RM (RESET MODE) control sequence.
3. The SELECT CONFORMANCE LEVEL (DECSCL) control sequence is executed (any level selected by the parameter).
4. The RESET TO INITIAL STATE (RIS) control function is executed.
5. The power up self-test is executed.
6. Entering Set-Up in Level 1.
7. Executing Clear Comm or Reset Terminal in Set-Up.
8. The SOFT TERMINAL RESET (DECSTR) control sequence is executed.

SET/RESET KEYBOARD ACTION MODE KAM

Levels: 1, 2

Purpose: Lock or unlock the entire keyboard.

Set Format:	CSI	2	h
	9/11	3/2	6/8
Reset Format:	CSI	2	l
	9/11	3/2	6/12

Description: The KAM control is used to set or reset the state of Keyboard Action Mode. When Keyboard Action Mode is reset (Unlocked), all keys on the keyboard transmit their defined codes or codes sequences. When Keyboard Action Mode is set (locked), the entire keyboard shall not transmit any codes and shall not keyclick, if enabled, until the state is reset.

State Affected:

```
KEYBOARD_ACTION_MODE:  (UNLOCKED,LOCKED);
```

Algorithm:

```
PROCEDURE SET_KEYBOARD_ACTION_MODE;
BEGIN
KEYBOARD_ACTION_MODE:= LOCKED;
END;


PROCEDURE RESET_KEYBOARD_ACTION_MODE;
BEGIN
KEYBOARD_ACTION_MODE:= UNLOCKED;
END;
```

Known Deviations:

This control is ignored by the VT100 and VT125.

6.6.3 Auto Repeat Mode

The terminal shall provide the ability to select whether the keyboard keys will automatically transmit codes at a periodic rate when held down [set state] or will transmit only one code each time they are depressed [reset state].

Auto-repeat can be disabled or set to a variable rate as a Set-Up feature. (Note: Provision of variable auto-repeat rates is an option, and is not required for conformance.) One of the rates provided shall be 30 keystrokes per second, continuous. Keys which can auto-repeat will start auto-repeating within ½ second of the time the initial key depression occurs. If the control sequence to turn auto-repeat off (DECARM) is received while an auto-repeat is in progress, the key will stop auto-repeating. If the control sequence to turn auto-repeat on is received when a key which had previously been auto-repeating is still held down, the key will immediately auto-repeat without delay. Except for the above, the effect of auto-repeating a key shall be the same as pushing the key a number of times. While any key is auto repeating, the affect of shift or control apply immediately. No new "start-up delay" results.

Note

The interval between the time the DECARM control is received and the time a key which was already auto-repeating actually stops repeating is UNDEFINED, and should not be depended upon by conforming software. For the purpose of testing compliance with the architecture, auto-repeat must stop within five seconds of receiving the DECARM control when the terminal is processing host output (not blocked by hold screen or alternate session).

SET/RESET AUTO REPEAT MODE DECARM

Levels: 1, 2

Purpose: Turn auto-repeating of keys on and off.

Set Format:	CSI	?	8	h
	9/11	3/15	3/8	6/8
Reset Format:	CSI	?	8	l
	9/11	3/15	3/8	6/12

Description: The DECARM control is used to set or reset the state of Auto Repeat Mode. When Auto Repeat Mode is reset (Repeat_Off), none of the keyboard keys will auto repeat as a result of being held down. When Auto Repeat Mode is set (Repeat_On) certain keys will repeat at a fixed rate as a result of being held down (see description above).

NOTE

Auto Repeat Mode is a User Preference Feature and should only be used by conforming software in response to explicit requests of the terminal user.

State Affected:

```
AUTO_REPEAT_MODE: (REPEAT_OFF,REPEAT_ON);
```

Algorithm:

```
PROCEDURE SET_AUTO_REPEAT_MODE;
BEGIN
AUTO_REPEAT_MODE:= REPEAT_ON;
END;


PROCEDURE RESET_AUTO_REPEAT_MODE;
BEGIN
AUTO_REPEAT_MODE:= REPEAT_OFF;
END;
```

Known Deviations: None

6.6.3.1 Keys Which Do Not Auto Repeat -

All keys that transmit bytes across the coding interface shall be auto-repeatable, with the exception of the Return and Escape key. Only keys that do not transmit immediately shall not auto-repeat. The keys that do not auto-repeat are:

1. Return Key
2. Escape Key
3. The Local Function Keys
4. Control key by itself (control characters auto-repeat)
5. Shift key by itself (shifted characters auto-repeat)
6. Lock
7. Compose
8. Non-Spacing Diacritical keys
9. Keys in an explicit or implied compose sequence

6.6.3.2 Auto Repeat Guidelines -

In order to give a constant repeat rate at each transmission speed, the speed of auto repeat is a function of the Host Transmit Speed. At speeds of 2400 Baud or above, all keys will auto repeat at 30 keystrokes per second.

The following table is provided as a guideline and is not required for conformance.

For the purpose of auto repeat, the keyboard is separated into three groups:

1. Group A - Main typing array
2. Group B - Cursor keys and numeric keypad keys
3. Group C - Top row function keys and editing keys

Every key in each group will auto repeat at the fixed rate set by the transmit speed, regardless of how many codes the key actually transmits.

Auto Repeat Rates

Host Port transmit Speed	Group A	Group B	Group C
2400 Baud or above	30 keys/sec	30 keys/sec	30 keys/sec
1200	30	30	24
600	30	20	12
300	30	10	6
150	13	6	6
110	10	6	6
75	6	6	6

These repeat rates assume that an 8-bit character is used, either 8-bit data with no parity, or 7-bit data plus a parity bit. Also for speeds of 300 baud or above 1 stop bit is assumed, for 150 baud or below, 2 stop bits are assumed. Other settings of these parameters will not change the repeat rate, however selecting 8-bits plus parity and/or 2 stop bits will, in some cases, cause a slight irregularity in the repeat rate as the keyboard transmit silo fills up. The "Limited Transmit" feature will not affect auto repeat rates, since all five codes can be transmitted at the limited speed of 150 characters per second. In local mode, keys will auto repeat at 30 keystrokes per second.

6.6.4 Visual Indicators

6.6.4.1 Hold Screen Indicator -

The Hold Screen LED will be lit whenever the screen image (or display of one or more sessions) is being held. It will be off whenever the screen is not being held. When the HOLD SCREEN (HOLD SESSION) key is pressed the LED shall toggle along with the function.

6.6.4.2 Lock Key Indicator -

The LOCK LED lights whenever the Shift Lock mode or Caps Lock mode is in effect. The LED will toggle along with the corresponding mode.

If a Shift Lock keyboard is selected in Set-Up, Shift Lock mode is turned on by pressing the LOCK key, and turned off by pressing either the LOCK key or a SHIFT key. If a Caps Lock keyboard is selected in Set-Up: Caps Lock mode is toggled on or off by pressing the LOCK key.

6.6.4.3 Compose Indicator -

The COMPOSE LED is turned on at the start of a compose. A compose is initiated either by typing the Compose Key or by typing a non-spacing diacritical mark. It remains on until one of the following occurs:

1. The character has been composed.
2. An error occurs.
3. The delete key is pressed.

Errors during compose terminate the compose and turn off the LED.

6.6.4.4 Keyboard Lock (Wait) Indicator -

The Wait LED lights when the keyboard is locked either because of data overrun (XOFF synchronization control code received from the host) or because of receipt of the control sequence which locks the keyboard (KAM). The LED is off whenever the keyboard is not locked. If the keyboard is locked during a compose, the compose is suspended until the keyboard is unlocked or until an action, such as enter Set-Up, is taken.

6.6.5 Audible Indicators

6.6.5.1 Warning Bell -

The Warning Bell will be sounded (if it is enabled in Set-Up) whenever one of the following occur:

a. A CTRL-G (BEL) is received by the terminal - ring when the character would have gone to the screen had it been printable.
b. When a compose error is made.
c. When the margin is reached (subject to Set-Up).

Closely spaced bell codes will be sounded individually at a rate of 4-5 per second. As bell characters arrive, they are stored in a bell buffer until they can be sounded. The bell buffer will have 10 entries. If the bell buffer fills, additional bell characters will be discarded until there is space in the bell buffer.

No control function is provided to enable/disable the Warning Bell from the host. It may be possible to change the volume of the Warning Bell, or disable it entirely, as a local Set-Up function (not mandatory).

6.6.5.2 Margin Bell -

The Margin Bell is set to RING ENABLED when the all of the following conditions are true:

1. Margin Bell is enabled in Set-Up ("Margin Bell").
2. A "normal keystroke" is typed (while not in SET-UP mode).

NOTE

In this context, a normal keystroke is any keystroke which does not generate one of the following characters as the first character: ESC, CSI, DCS. A COMPOSE is a "normal keystroke" (even though it is 2 or 3 keystrokes) since it results in one displayable character. Examples of normal keystrokes are "A", "=" and "COMPOSE e '" (<e'>), and (usually) the UDKs (i.e. Shift-F6).

The Margin Bell is rung under the following conditions:

a. The MARGIN BELL is RING ENABLED.
b. Any character is printed to the screen which causes the cursor to move from a column =< (RIGHT MARGIN - 8) to any column on the same line > (RIGHT MARGIN - 8). Note horizontal positioning controls do not ring the bell or affect whether the bell is ring enabled (these characters are not "printed" to the screen).

The Margin Bell is set to RING DISABLED when:

1. Margin Bell is disabled in Set-Up ("No Margin Bell").
2. The Margin Bell has been rung.
3. The cursor moves to a new line.
4. The terminal is in Set-Up

The MARGIN BELL conditions described above are dynamic. Because of this the following might occur:

With the cursor at column RIGHT MARGIN - 8 an 'A' (or any normal keystroke) is typed, thus setting the MARGIN BELL to RING ENABLED (assuming that Margin Bell is enabled in Set-Up).

A linefeed is now sent to the screen which will leave the cursor in column RIGHT MARGIN - 8 but on the next line. This will cause the the Margin Bell to be set to RING DISABLED.

The 'A' is now sent to the screen causing the cursor to move from RIGHT MARGIN - 8 to RIGHT MARGIN - 7, but the MARGIN BELL does not ring because it is RING DISABLED.

Had the 'A' been sent to the screen before the linefeed, the MARGIN BELL would have rung. It depends on the status of RING ENABLED/DISABLED at the instant the other condition for ringing the Margin Bell is satisfied.

No control function is provided to enable/disable the Margin Bell from the host. It may be possible to change the volume of the Margin Bell, or disable it entirely, as a local Set-Up function (not mandatory).

6.6.5.3 Keyclick -

Keyclick may be enabled or disabled in Set-Up. If Keyclick is enabled, the keyboard will "click" once for each depression of a key. If a key is auto-repeating, the click will occur once for each key sent, (that is, once for each alphabetic character and once for each control sequence from a function key). The keyclick shall be synchronized with the actual keystroke or auto-repeat rate. All keys which transmit codes or cause the terminal to take some immediate action click.

Set-Up, Hold Screen, Lock, and Compose all click since they invoke immediate actions (Lock and Compose change the input mode). The Shift key will click if exiting the Shift Lock state, otherwise it does not click. The Shift (except for the above noted exception) and the Control keys do not click since these keys do not cause any immediate action by themselves, but are intended to be pressed in conjunction with another key. Keys that form invalid control combinations do not click. Keys which are inoperative due to Level 1 operation or because the Host Port Environment Mode is 7-bits also do not click.

No control function is provided to affect the state of keyclick from the host. It may be possible to change the volume of the keyclick sound, or disable it entirely, as a local Set-Up function (not mandatory).

6.6.5.3.1 Interaction Of Keyclick And Bells - If a keyclick and a bell are generated by the same action, for example typing the second character in some invalid compose sequences, then either bell (Warning or Margin) takes precedence over the keyclick, and the keyclick is not generated.

6.7 KEYBOARD STATE AND OPERATING MODES THAT AFFECT KEYBOARD ENCODING

The keyboard is sensitive to the following operating modes that affect what codes are transmitted:

- Emulation mode: Primarily affects codes transmitted by function keys: One of VT52, VT100 or VT300.
- C1 Transmission mode: Affects how C1 controls are transmitted. One of "7-bit controls", or "8-bit controls".
- Character Set Mode: Primarily affects keyboard encoding. Choices are "7-bit Characters", or "8-bit Characters".
- User Preference Supplemental Set (UPSS): Determines keyboard supplemental set when 8-bit characters are enabled. Also establishes initial supplemental set for display (defaults to G2 on power-up), and allows one level of software indirection for designating supplemental character sets. Choices are "DEC Supplemental" or "ISO Latin-1".
- Keyboard Dialect: Primarily affects keyboard layout by specifying which country keyboard to use. When 7-bit Characters and Typewriter mode are in effect, the Keyboard Dialect also selects the corresponding National Replacement Character Set.
- Keyboard Usage Mode: Primarily affects keyboard layout. In 7-bit Characters mode, also affects character encoding. Choices are "Data Processing Keys" or "Typewriter Keys".
- Host line environment: Establishes frame size for character transmission. May restrict terminal to 7-bit characters. Choices are "7-bit host line", or "8-bit host line".
- Lock key mode: One of Caps Lock or Shift Lock.
- BackArrow key mode: One of Delete or Backspace (not mandatory).
- Set-Up feature to disable break. Choices are Break, or No Break. Disables generating break signal when Break Key alone is pressed. Does not affect Shift-Break, or Control-Break.
- Set-Up feature to disable Compose key (not mandatory). Does not disable auto-compose sequences which do not use the Compose key.
- Set-Up feature to remap shifted comma and period to send < and > (not mandatory).

- Set-Up feature to remap <> key to send `~ (not mandatory).

- Set-Up feature to remap `~ key to send ESC (not mandatory).

- Set-Up feature to remap KP, (keypad comma) to send SPACE (not mandatory).

- Cursor Key Mode: selects between having the four cursor (arrow) keys transmit ANSI standard cursor movement escape sequences, or Application Function Codes. See "Cursor Keys" section below.

- Numeric Keypad Mode: selects between having the auxiliary keypad keys transmit ASCII decimal digits (3/0 to 3/9) 'Numeric' mode, or application control functions 'Application Mode'. See "Numeric Keypad Keys" section below.

6.7.1 VT52 And VT100 (Level 1) Emulation Mode

In VT52 and VT100 Emulation modes, the following restrictions apply to the keyboard. Keys which are dead are completely ignored by the terminal. They do not click, transmit any code, or cause the terminal to perform any action. The keys are referred to by their names. See the "Named Key Map".

a. Only 7-bit character codes may be produced. This restricts DEC Multinational and ISO Latin-1 to their 7-bit ASCII subset.
b. The six "Edit Keys" E1, E2, E3, E4, E5, and E6 are dead.
c. Function Keys F6, F7, F8, F9, and F10 are dead.
d. Function Key F11 will transmit ESC. This key will not auto-repeat.
e. Function Key F12 will transmit BS
f. Function Key F13 will transmit LF
g. Function Keys F14, F15, F16, F17, F18, F19, and F20 are dead.
h. There will be no eight bit characters generated. Keys which would normally produce an 8-bit code will be dead (exception is British pound sign which transmits 10/3 in 8-bits, and shall transmit 2/3 for VT100 compatibility).
i. Compose will work only to produce 7-bit codes. Any attempt to compose an 8-bit code will result in the compose error recovery.
j. Eight bit control characters (C1 characters) cannot be generated.

6.7.2 VT200 (Level 2) And VT300 (Level 3) Emulation Mode

All function keys are live. The Function Keys, including F11, F12, and F13 all transmit control sequences. ESC, BS, and LF are not available on F11, F12, and F13. BS may be available if Backspace mode of Backarrow key is selected. If the host line environment is 8-bits, the full repertoire of currently selected character encodings is available. If the host line environment is 7-bits, there are restrictions:

a. Functions keys and other 8-bit controls are sent using ESC Fe sequences instead of CSI sequences, regardless of the C1 Transmission Mode.
b. Only the 7-bit ASCII subset of DEC Multinational or ISO Latin-1 may be generated. Keystrokes that would produce 8-bit codes are dead. Compose only works when used to produce 7-bit codes.

Local mode operation is never subject to the restrictions of the 7-bit host line environment.

6.7.3 C1 Transmission Mode

This mode selects how C1 controls are transmitted in Level 2 or Level 3. "7-bit Controls" forces all function keys and other C1 controls to be sent in their 7-bit format as ESC Fe sequences. "8-bit Controls" causes C1 controls to be sent in their 8-bit format (from columns 8 and 9 in the code table) when operating in Level 2 or Level 3, 8-bit Characters. If the host line environment is 7-bits, C1 controls are restricted to their 7-bit format regardless of the C1 Transmission mode.

The following control functions can also be used to set the C1 Transmission mode.

S7C1T - Select 7-bit C1 Transmission

ESC 2/0 4/6 (ESC SP F)

S8C1T - Select 8-bit C1 Transmission

ESC 2/0 4/7 (ESC SP G)

6.7.4 Character Set Mode

The Character Set Mode selects between "8-bit Characters" or "7-bit Characters" for keyboard character encoding.

When 8-bit Characters is selected, the keyboard will transmit 8-bit graphic characters as ASCII from GL and either DEC Supplemental or ISO Latin-1 supplemental from GR. The use of DEC Supplemental or ISO Latin-1 supplemental is determined by the User Preference Supplemental Set (UPSS). Note: If the host line environment is set to 7-bits, the keyboard is restricted to 7-bit Characters.

When 7-bit Characters is selected, the keyboard shall only generate 7-bit characters. All attempts to generate 8-bit codes when in 7-bits (such as COMPOSE sequences) will fail. The keyboard character set is determined by the Keyboard Usage Mode (Typewriter or Data Processing), and Keyboard Dialect. See below.

Explanatory Note

> The character set mode, 7-bit/8-bit Characters corresponds to National/Multinational Mode on the VT200 but with two simplifications. 8-bit Characters are temporarily disabled when in "VT100 mode", or a "7-bit host line" is selected. In practice, this means the terminal no longer supports ASCII with the Typewriter Keys layout except on the North American Keyboard. This makes sense. The typewriter layout has no inherent advantage when using ASCII, it simply has more dead keys. Confusing distinctions like VT100 Multinational mode have been eliminated.

SET/RESET CHARACTER SET MODE DECNRCM

Levels: 2x, 3x

Purpose: Select 7-bit or 8-bit Characters for keyboard encoding

Set Format:	CSI	?	4	2	h	(7-bit Characters)
	9/11	3/15	3/4	3/2	6/8	(National)
Reset Format:	CSI	?	4	2	l	(8-bit Characters)
	9/11	3/15	3/4	3/2	6/12	(Multinational)

Description: The DECNRCM control is used to select between 7-bit or 8-bit Character sets for keyboard encoding. Note: Setting this mode has the side effect of resetting the display character sets to their default state, i.e. the state at power-on or reset.

State Affected:

```
CHARACTER_SET_MODE:   (EIGHT_BITS, SEVEN_BITS);
R_CHARACTER_SET_MODE: (EIGHT_BITS, SEVEN_BITS);
  (* used to keep track of character set mode
     when a restrictive condition (7-bit Host Line,
     or VT100 mode) is in effect. *)
IN_USE_TABLE
DESIGNATED_GRAPHIC_SETS
```

Algorithm:

```
PROCEDURE SET_CHARACTER_SET_MODE;
BEGIN
CHARACTER_SET_MODE:= SEVEN_BITS;
(* keep track of mode if there are restrictions in effect *)
R_CHARACTER_SET_MODE:= SEVEN_BITS;
IF KEYBOARD_USAGE_MODE = TYPEWRITER THEN
  BEGIN
  IN_USE_TABLE.GL:= NRCS_LIST[KEYBOARD_DIALECT];
  IN_USE_TABLE.GR:= NRCS_LIST[KEYBOARD_DIALECT];;
  DESIGNATED_GRAPHIC_SETS[G0]:= NRCS_LIST[KEYBOARD_DIALECT];
  DESIGNATED_GRAPHIC_SETS[G1]:= NRCS_LIST[KEYBOARD_DIALECT];
  DESIGNATED_GRAPHIC_SETS[G2]:= NRCS_LIST[KEYBOARD_DIALECT];
  DESIGNATED_GRAPHIC_SETS[G3]:= NRCS_LIST[KEYBOARD_DIALECT];
  END;
ELSE
  (* keyboard usage mode is Data Processing *)
  BEGIN
  IN_USE_TABLE.GL:= ASCII_G;
  IN_USE_TABLE.GR:= ASCII_G:
  DESIGNATED_GRAPHIC_SETS[G0]:= ASCII_G;
  DESIGNATED_GRAPHIC_SETS[G1]:= ASCII_G;
  DESIGNATED_GRAPHIC_SETS[G2]:= ASCII_G;
```

```
    DESIGNATED_GRAPHIC_SETS[G3]:= ASCII_G;
    END;
  END;


  PROCEDURE RESET_CHARACTER_SET_MODE;
  BEGIN
  CASE CONFORMANCE_LEVEL OF
  (* keep track of mode if there are restrictions in effect *)
  R_CHARACTER_SET_MODE:= EIGHT_BITS;
  (* ignore if 7-bit host line or VT100 mode *)
  IF (ENVIRONMENT_TYPE <> SEVEN_BITS) AND
     (CONFORMANCE_LEVEL <> LEVEL_1) THEN
    BEGIN
    CHARACTER_SET_MODE:= EIGHT_BITS;
    IN_USE_TABLE.GL:= ASCII_G;
    IN_USE_TABLE.GR:= UPSS_G;
    DESIGNATED_GRAPHIC_SETS[G0]:= ASCII_G;
    DESIGNATED_GRAPHIC_SETS[G1]:= ASCII_G;
    DESIGNATED_GRAPHIC_SETS[G2]:= UPSS_G;
    DESIGNATED_GRAPHIC_SETS[G3]:= UPSS_G;
    END;
  END;


  Known Deviations:  None
```

6.7.5 Keyboard Usage Mode

Each version of the keyboard is oriented toward a particular language and/or country. The North American version, for example, has all 94 characters of ASCII on separate keys. Therefore, compose sequences are not needed to transmit ASCII characters, only to transmit Supplemental characters. Many of the keyboards have one or more keys which generate different characters depending on whether the keyboard is set to Data Processing mode or Typewriter mode (for data processing versus office use). The data processing legends appear to the right of the typewriter legends, if different. When in Data Processing mode, these keys generate the character codes for the data processing legends. When in Typewriter mode, these keys generate character codes for their typewriter legends.

When the terminal is in 7-bit Characters Mode, selecting Data Processing Keys also selects the ASCII character set instead of the National Replacement Character Set (NRCS) corresponding to the Keyboard Dialect, as would be the case if Typewriter keys were selected.

The keyboard usage mode is selectable from Set-Up, or by host control function (DECKBUM) in Level 3.

The tables later in this section list all the keys and characters produced for each mode. The following list summarizes the Data Processing characters by key position for each country keyboard. The most recent or recommended variation for each dialect is listed first. Note this list applies to the Keyboard Dialect (keyboard mapping tables) chosen in Set-Up, not the physical keyboards.

1. North American - none

2. British (LK201-EE US/UK)
   E03 has "#"

3. British (LK201-AE)
   E03 has "#"
   D11 has "\"

4. Flemish (Belgium)
   E06 has "["
   E07 has "]"
   C11 has "\"

5. Canadian (French)
   D11 has "]" and "["

6. Danish (2nd, LK201-ED)
   E00 has "@"
   E03 has "#"
   D11 has "}" and "]"
   C10 has "{" and "["
   C11 has "|" and "\"

7. Danish (1st, LK201-AD)
   E03 has "@"
   D11 has "]" and "["
   C11 has "#" and "\"

8. Finnish (3rd, LK201-NX)
   E03 has "#"
   D11 has "}" and "]"
   D12 has "~" and "^"
   C10 has "|" and "\"
   C11 has "{" and "["

9. Finnish (2nd, LK201-NF)
   E03 has "@"
   D11 has "]" and "["
   D12 has "#" and "\"

10. Finnish (1st, LK201-AF)
    E03 has "@"
    D11 has "]" and "["
    D12 has "#" and "\"

11. Austrian/German (2nd, LK201-NG)
    E11 has "|"
    D11 has "@" and "\"
    C10 has "}" and "{"
    C11 has "]" and "["

12. Austrian/German (1st, LK201-AG)
    D11 has "@" and "\"
    C11 has "]" and "["

13. Dutch (2nd, LK201-NH equivalent to LK201-EE) - none

14. Dutch (1st, LK201-AH)
    E02 has "["
    E03 has "]"
    E10 has "\"

15. Italian
    E01 has "@"
    E02 has "#"
    E09 has "["
    E10 has "]"
    C12 has "\"

16. Swiss (French)
    E04 has "@"
    D11 has "\"
    C10 has "["
    C11 has "]"
    C12 has "#"

17. Swiss (German)
    E04 has "@"
    D11 has "\"
    C10 has "["
    C11 has "]"
    C12 has "#"

18. Swedish (2nd, LK201-NM)
    E03 has "@"
    D11 has "]" and "["
    D12 has "#" and "\"
    C10 has "}" and "{"
    C11 has "|"

19. Swedish (1st, LK201-AM)
    E03 has "@"
    D11 has "]" and "["
    D12 has "#" and "\"

20. Norwegian (2nd, LK201-EN)
    D11 has "}" and "]"
    C10 has "|" and "\"
    C11 has "{" and "["

21. Norwegian (1st, LK201-AN)
    E03 has "@"
    D11 has "]" and "["
    C11 has "#" and "\"

22. Belgian/French
    E06 has "["
    E07 has "]"
    C11 has "\"

23. Spanish
    E00 has "@" and "\"
    E12 has "]" and "["

24. Portuguese - none

SET/RESET KEYBOARD USAGE MODE DECKBUM

Levels: 2x, 3

Purpose: Select Data Processing or Typewriter Keys mode.

Set Format:	CSI	?	6	8	h	(D.P.)
	9/11	3/15	3/6	3/8	6/8	
Reset Format:	CSI	?	6	8	l	(Typewriter)
	9/11	3/15	3/6	3/8	6/12	

Description: The DECKBUM control is used to select between Data Processing or Typewriter Keys mode.

Note:

> Keyboard Usage Mode is a User Preference Feature and should only be used by conforming software in response to explicit requests of the terminal user.

State Affected:

```
KEYBOARD_USAGE_MODE: (TYPEWRITER_KEYS,DATA_PROCESSING_KEYS);
```

Algorithm:

```
PROCEDURE SET_KEYBOARD_USAGE_MODE;
BEGIN
KEYBOARD_USAGE_MODE:= DATA_PROCESSING_KEYS;
END;

PROCEDURE RESET_KEYBOARD_USAGE_MODE;
BEGIN
KEYBOARD_USAGE_MODE:= TYPEWRITER_KEYS;
END;
```

Known Deviations: None

6.7.6 Keyboard Dialect

The Keyboard Dialect primarily effects the keyboard layout by specifying which country keyboard to use. When 7-bit Characters and Typewriter mode are in effect, the Keyboard Dialect also selects the corresponding National Replacement Character Set (new for Level 3).

When 7-bit Characters and Typewriter Keys mode are selected, the terminal generates characters from the current National Replacement Character Set (NRCS) determined by the Keyboard Dialect. This applies in all emulation modes: VT52, VT100 (Level 1), and VT200 (Level 2). NRCS are by definition 7-bit sets, thus only 7-bit codes will be generated. Additional compose sequences are available in this mode to generate characters that are in the current NRCS but not on the corresponding country keyboard.

Guideline: On the VT320, the national replacement character set for each keyboard is as follows:

1. North American - 7-bit Characters mode is not available. (ASCII is the left hand graphic set for both DEC Multinational and ISO Latin-1)
2. Flemish - French NRCS/AFNOR NF Z 62-010 [1973]
3. Canadian (French) - DEC French Canadian NRCS
4. British - British NRCS/BS 4730
5. Danish - Norwegian/Danish NRCS 3/NS 4551-1, DS 2089
6. Finnish - DEC Finnish NRCS
7. Austrian/German - German NRCS/DIN 66 003
8. Dutch - 7-bit Characters mode is not available (same as North American).
9. Italian - ISO Italian NRCS
10. Swiss (French) - DEC Swiss NRCS
11. Swiss (German) - DEC Swiss NRCS
12. Swedish - DEC Swedish NRCS
13. Norwegian - Norwegian/Danish NRCS 3/NS 4551-1, DS 2089
14. Belgian/French - French NRCS/AFNOR NF Z 62-010 [1973]
15. Spanish - ISO Spanish NRCS
16. Portuguese - DEC Portuguese NRCS

For more information on character sets and coding, refer to DEC STD 169 Digital Standard Coded Graphic Character Sets for Hardware and Software.

6.7.6.1 British Pound Sign <L=> -

The North American keyboard dialect differs from the British keyboard dialect only in the meaning of the Shift-3 (E03) key. A single keyboard model (LK201-EE) accomodates both of these keyboard dialects.

When "North American Keyboard" is selected, the Shift-3 key always sends 2/3 ("#" from ASCII character set).

When "British Keyboard" is selected, the setting of Character Set Mode, 8-bit Characters/7-bit Characters (DECNRCM) determines whether the British Pound Sign character is sent as 10/3 or 2/3. In the former case, from the DEC (or ISO) Supplemental character set, in the latter case from the UK national character set. When in 7-bit Characters mode, and the Keyboard Usage Mode is set to Data Processing Keys, the Shift-3 key sends 2/3 ("#" from ASCII character set).

The possible combinations are summarized in the chart below:

Keyboard Dialect	DECNRCM setting	Keyboard Usage Mode	Code sent by terminal	Character and set
British	8-bit Chars.	either	10/3	£ - DEC or ISO Supplemental
British	7-bit Chars.	Typewriter	2/3	£ - British NRC
British	7-bit Chars.	Data Proc.	2/3	# - ASCII
N. America	N/A *	either	2/3	# - ASCII

* when "North American" dialect is selected, 7-bit characters mode is not available.

The legend for the shift 3 (E03) key is #£.

6.8 CURSOR KEYS

The four cursor arrow keys are intended for use by the terminal user to indicate movement of the cursor symbol in the display. No action to move the cursor is performed by the terminal unless it is in Local Mode or Local Editing Mode. The cursor movement controls must be transmitted to the terminal on receipt of these sequences in order for a corresponding movement of the cursor symbol to occur.

6.8.1 Cursor Key Mode

The device shall provide four keys specifically designated for cursor movement as up, down, left and right. If the Cursor Key Mode is in the reset state, these keys will transmit the appropriate ANSI escape sequence codes (CSI A, CSI B, CSI D, CSI C). If the Cursor Key Mode is in the set state, these keys will transmit applications function codes (SS3 A, SS3 B, SS3 D, SS3 C).

Note that when Cursor Key Mode is in the reset state the control sequences transmitted by these keys are in fact the cursor movement control functions, and if echoed back to the terminal will cause cursor movement to occur.

6.8.2 Cursor Key Codes

The codes transmitted by the Cursor Keys are not effected in any way by the depression of the Shift or Lock keys.

In the following table, the first entry indicates the code sequence transmitted when the C1 Transmission Mode is in the reset state (SEVEN-BIT). The second entry indicates the code sequence transmitted when the C1 Transmission Mode is in the set state (EIGHT_BIT).

Codes Transmitted by Cursor Keypad Keys

Key	Legend	DECCKM reset	DECCKM set
C17	∧ (up arrow)		
		1/11 5/11 4/1 (ESC [A)	1/11 4/15 4/1 (ESC O A)
		9/11 4/1 (CSI A)	8/15 4/1 (SS3 A)
B17	∨ (down arrow)		
		1/11 5/11 4/2 (ESC [B)	1/11 4/15 4/2 (ESC O B)
		9/11 4/2 (CSI B)	8/15 4/2 (SS3 B)
B18	-> (right arrow)		
		1/11 5/11 4/3 (ESC [C)	1/11 4/15 4/3 (ESC O C)
		9/11 4/3 (CSI C)	8/15 4/3 (SS3 C)
B16	<- (left arrow)		
		1/11 5/11 4/4 (ESC [D)	1/11 4/15 4/4 (ESC O D)
		9/11 4/4 (CSI D)	8/15 4/4 (SS3 D)

SET/RESET CURSOR KEY MODE DECCKM

Levels: 1, 2

Purpose: Cause the Cursor keys to transmit cursor control functions or application function codes.

Set Format:	CSI	?	1	h
	9/11	3/15	3/1	6/8
Reset Format:	CSI	?	1	l
	9/11	3/15	3/1	6/12

Description: The DECCKM control is used to set or reset the state of Cursor Key Mode. When Cursor Key Mode is reset (Cursor), the Cursor keys will transmit ANSI standard cursor control sequences. When Cursor Key Mode is set (Ck_Application), the Cursor keys will transmit applications function codes similar to those transmitted by the Numeric Keypad when Keypad Mode is set (Kp_Application).

Note:

> In Level 1 operation, any interaction or dependency between Cursor Key Mode and Keypad Mode is UNDEFINED. In Level 2 operation, however, it is required that these modes be independent of one another (i.e., it is not necessary to have one set in order to set the other).

State Affected:

```
CURSOR_KEY_MODE: (CURSOR,CK_APPLICATION);
```

Algorithm:

```
PROCEDURE SET_CURSOR_KEY_MODE;
BEGIN
CURSOR_KEY_MODE:= CK_APPLICATION;
END;


PROCEDURE RESET_CURSOR_KEY_MODE;
BEGIN
CURSOR_KEY_MODE:= CURSOR;
END;
```

Known Deviations: None

6.9 NUMERIC KEYPAD KEYS

The codes transmitted by these keys are not effected in any way by the depression of the Control, Shift or Lock keys.

6.9.1 Keypad Application/Numeric Mode

There are two modes of operation for the Numeric Keypad keys. These modes are referred to as Keypad Application Mode and Keypad Numeric Mode. Since they are mutually exclusive (setting one mode causes the other to be reset) they can be considered as two states of the same mode.

6.9.1.1 Numeric Keypad Mode -

When Keypad Numeric Mode is set, the keys in the Numeric Keypad transmit ASCII codes for the numbers 0 to 9 (3/0 to 3/9 inclusive) and the ASCII codes for comma, minus/hyphen, period, and carriage return. This setting of the mode permits the terminal user to use the keypad for inputting certain numeric information without change to application software that normally expects input from the main key array only. The other keys on the keypad transmit the SINGLE SHIFT THREE (SS3) control code followed by a single ASCII character indicating the key which was depressed. See the Numeric Keypad code tables.

6.9.1.2 Application Keypad Mode -

When Keypad Application Mode is set, all keys on the keypad transmit the SINGLE SHIFT THREE (SS3) control code followed by a single ASCII character indicating the key which was depressed. Note that SS3 will be transmitted as a single character code (8/15) when C1 Transmission Mode is set to 8-bit and as a two character ESC Fe sequence (ESC O, 1/11 4/15) when set to 7-bit. This setting of the mode permits application software to be able to distinguish a keypad key from a main key array key, which is especially useful for applications that use the keypad as a set of function keys.

6.9.1.3 Numeric Keypad Key Codes -

In the following table, the first entry indicates the code sequence transmitted when C1 Transmission Mode is in the reset (SEVEN_BIT) state. The second entry indicates the code sequence transmitted when C1 Transmission Mode is in the set (EIGHT_BIT) state.

Codes Transmitted by Numeric Keypad Keys

Key	Legend	DECKPNM set			DECKPAM set			
A20	0	3/0	(0)		1/11	4/15	7/0	(ESC O p)
						8/15	7/0	(SS3 p)
B20	1	3/1	(1)		1/11	4/15	7/1	(ESC O q)
						8/15	7/1	(SS3 q)
B21	2	3/2	(2)		1/11	4/15	7/2	(ESC O r)
						8/15	7/2	(SS3 r)
B22	3	3/3	(3)		1/11	4/15	7/3	(ESC O s)
						8/15	7/3	(SS3 s)
C20	4	3/4	(4)		1/11	4/15	7/4	(ESC O t)
						8/15	7/4	(SS3 t)
C21	5	3/5	(5)		1/11	4/15	7/5	(ESC O u)
						8/15	7/5	(SS3 u)
C22	6	3/6	(6)		1/11	4/15	7/6	(ESC O v)
						8/15	7/6	(SS3 v)
D20	7	3/7	(7)		1/11	4/15	7/7	(ESC O w)
						8/15	7/7	(SS3 w)
D21	8	3/8	(8)		1/11	4/15	7/8	(ESC O x)
						8/15	7/8	(SS3 x)
D22	9	3/9	(9)		1/11	4/15	7/9	(ESC O y)
						8/15	7/9	(SS3 y)
C23	, (comma)	2/12	(,)		1/11	4/15	6/12	(ESC O l)
						8/15	6/12	(SS3 l)
D23	- (minus/hyphen)	2/13	(-)		1/11	4/15	6/13	(ESC O m)
						8/15	6/13	(SS3 m)
A22	. (period)	2/14	(.)		1/11	4/15	6/14	(ESC O n)
						8/15	6/14	(SS3 n)
A23	ENTER	same as Return			1/11	4/15	4/13	(ESC O M)
						8/15	4/13	(SS3 M)
E20	PF1	1/11	4/15	5/0	1/11	4/15	5/0	(ESC O P)
			8/15	5/0		8/15	5/0	(SS3 P)
E21	PF2	1/11	4/15	5/1	1/11	4/15	5/1	(ESC O Q)
			8/15	5/1		8/15	5/1	(SS3 Q)
E22	PF3	1/11	4/15	5/2	1/11	4/15	5/2	(ESC O R)
			8/15	5/2		8/15	5/2	(SS3 R)
E23	PF4	1/11	4/15	5/3	1/11	4/15	5/3	(ESC O S)
			8/15	5/3		8/15	5/3	(SS3 S)

6.9.2 Enter Key Operation

When Keypad Mode is in the Numeric state, depressing the Enter key causes either a Carriage Return (CR, 1/13) or a Carriage Return/Line Feed pair (CR LF, 0/13 0/10) to be transmitted, depending on the state of New Line Mode.

When Keypad Mode is in the Application state, depressing the Enter key causes an escape sequence (ESC O M, 1/11 4/15 4/13) to be transmitted. See the table of Numeric Keypad keys and their associated codes.

SET KEYPAD APPLICATION MODE DECKPAM

Levels: 1, 2

Purpose: Cause the auxiliary keypad keys to transmit control functions.

Format:

ESC	=
1/11	3/13

Description: The DECKPAM control places Keypad Mode into the set, or 'Application' state. In this state the keys of the Numeric Keypad on the terminal keyboard will transmit a SINGLE SHIFT THREE (SS3) code followed by a single graphic code indicating the key which was struck.

Notes:

1. SINGLE SHIFT THREE (SS3) is a C1 control code which is transmitted as 8/15 when the C1 Transmission Mode is set to EIGHT_BIT, and as 1/11 4/15 (ESC O) when set to SEVEN_BIT.

2. If the transmitted codes are echoed back to the terminal, or if the terminal is in local mode, the indicated character from the currently designated G3 set will be displayed; for example, when ASCII is designated as the G3 set, the PF1 key sequence would display as "P".

3. In Level 1 operation, any interaction or dependency between Cursor Key Mode and Keypad Mode is UNDEFINED. In Level 2 operation, however, it is required that these modes be independent of one another (that is, it is not necessary to have one set in order to set the other).

State Affected:

```
KEYPAD_MODE: (NUMERIC,KP_APPLICATION);
```

Algorithm:

```
PROCEDURE SET_KEYPAD_APPLICATION_MODE;
BEGIN
KEYPAD_MODE:= KP_APPLICATION;
END;
```

Known Deviations: None

SET KEYPAD NUMERIC MODE DECKPNM

Levels: 1, 2

Purpose: Cause the auxiliary keypad keys to transmit ASCII decimal digits (3/0 to 3/9).

Format: ESC >

1/11 3/14

Description: The DECKPNM control places Keypad Mode into the reset, or 'Numeric' state. In this state the keys of the Numeric Keypad on the terminal keyboard will transmit a combination of graphic and control codes.

Notes:

1. In Level 1 operation, any interaction or dependency between Cursor Key Mode and Keypad Mode is UNDEFINED. In Level 2 operation, however, it is required that these modes be independent of one another (i.e., it is not necessary to have one set in order to set the other).

State Affected:

```
KEYPAD_MODE: (NUMERIC,KP_APPLICATION);
```

Algorithm:

```
PROCEDURE SET_KEYPAD_NUMERIC_MODE;
BEGIN
KEYPAD_MODE:= NUMERIC;
END;
```

Known Deviations: None

SET/RESET NUMERIC KEYPAD MODE DECNKM

Levels: 2x, 3

Purpose: Select between numeric and application keypad mode. This set/reset mode sequence was added to simplify inquiring the state of the auxiliary keypad using Terminal State Interrogation (TSI) functions.

Set Format:	CSI	?	6	6	h	(Application)
	9/11	3/15	3/6	3/6	6/8	
Reset Format:	CSI	?	6	6	l	(Numeric)
	9/11	3/15	3/6	3/6	6/12	

Description: This Set/Reset mode sequence selects between having the auxiliary keypad keys transmit ASCII decimal digits (3/0 to 3/9) 'Numeric' mode, or application control functions 'Application Mode'. The set mode sequence is equivalent to DECKPAM, and the reset mode sequence is equivalent to DECKPNM.

State Affected:

```
KEYPAD_MODE: (NUMERIC,KP_APPLICATION);
```

Algorithm:

```
PROCEDURE SET_KEYPAD_APPLICATION_MODE;
BEGIN
KEYPAD_MODE:= KP_APPLICATION;
END;


PROCEDURE SET_KEYPAD_NUMERIC_MODE;
BEGIN
KEYPAD_MODE:= NUMERIC;
END;
```

Known Deviations: None

6.10 EDITING KEYPAD KEYS

These keys transmit designated control sequences which are intended for editing applications. The Editing Keypad Keys are inoperative when pressed in combination with either the Shift or Control keys.

In the following table, the first entry indicates the code sequence transmitted when C1 Transmission Mode is in the reset (SEVEN_BIT) state. The second entry indicates the code sequence transmitted when C1 Transmission Mode is in the set (EIGHT_BIT) state.

In Level 1 operation, the Editing Keypad keys are inoperative (transmit nothing, and do not keyclick). In Level 2 operation, the Editing Keypad keys transmit a control sequence with a single digit parameter indicating the key.

Codes Transmitted by Editing Keypad Keys

Key Id	Legend	Codes Generated (Level 2 only)				
E16	Find	1/11	5/11	3/1	7/14	(ESC [1 ~)
			9/11	3/1	7/14	(CSI 1 ~)
E17	Insert Here	1/11	5/11	3/2	7/14	(ESC [2 ~)
			9/11	3/2	7/14	(CSI 2 ~)
E18	Remove	1/11	5/11	3/3	7/14	(ESC [3 ~)
			9/11	3/3	7/14	(CSI 3 ~)
D16	Select	1/11	5/11	3/4	7/14	(ESC [4 ~)
			9/11	3/4	7/14	(CSI 4 ~)
D17	Prev Screen	1/11	5/11	3/5	7/14	(ESC [5 ~)
			9/11	3/5	7/14	(CSI 5 ~)
D18	Next Screen	1/11	5/11	3/6	7/14	(ESC [6 ~)
			9/11	3/6	7/14	(CSI 6 ~)

6.11 APPLICATION FUNCTION KEYS

Application Function Keys are all the keys in the upper row which are not explicitly designated as Local Function Keys (all keys but the left five keys in the row). These keys transmit designated control sequences which are intended for use by operating systems and applications programs. The Permanent and Removable (System and Application) Label Strips serve as legends for these keys. The specific meaning of these keys is not defined by the architecture, and will vary from product to product. However, the codes transmitted by these keys will not change due to their usage.

When the Application Function keys are depressed in combination with the Shift Key, they transmit User Defined Key (UDK) sequences whose definitions have been previously received. See the section "User Defined Keys Extension" for a complete description of this Level 2 function. Note that if the UDK extension is not supported, the Application Function Keys in combination with Shift will not transmit anything. The codes transmitted by the Application Function keys are not affected in any way by the depression of the Lock key. When the Application Function keys are pressed in combination with the Control key, they are inoperative (no code is transmitted).

In the following table, the first entry indicates the code sequence transmitted when C1 Transmission Mode is in the reset (SEVEN_BIT) state. The second entry indicates the code sequence transmitted when C1 Transmission Mode is in the set (EIGHT_BIT) state.

In Level 1 operation, only the G11, G12, and G13 keys are active as ESC, BS, and LF, respectively; the remaining Application Function Keys are inoperative (transmit nothing, and do not keyclick). In Level 1 operation G11, G12 and G13 transmit nothing and do not click if either the shift key or control key are depressed when either ESC, BS, or LF are also depressed. When the operating mode is set to Level 2, the Application Function Keys transmit a control sequence with a two-digit parameter indicating the key.

Codes Transmitted by Application Function Keys

Key	Permanent Label Strip	Level 1	Level 2	
G05	F6	-	1/11 5/11 3/1 3/7 7/14	(ESC [17 ~)
			9/11 3/1 3/7 7/14	(CSI 17 ~)
G06	F7	-	1/11 5/11 3/1 3/8 7/14	(ESC [18 ~)
			9/11 3/1 3/8 7/14	(CSI 18 ~)
G07	F8	-	1/11 5/11 3/1 3/9 7/14	(ESC [19 ~)
			9/11 3/1 3/9 7/14	(CSI 19 ~)
G08	F9	-	1/11 5/11 3/2 3/0 7/14	(ESC [20 ~)
			9/11 3/2 3/0 7/14	(CSI 20 ~)
G09	F10	-	1/11 5/11 3/2 3/1 7/14	(ESC [21 ~)
			9/11 3/2 3/1 7/14	(CSI 21 ~)
G11	F11 (ESC)	1/11	1/11 5/11 3/2 3/3 7/14	(ESC [23 ~)
			9/11 3/2 3/3 7/14	(CSI 23 ~)
G12	F12 (BS)	0/8	1/11 5/11 3/2 3/4 7/14	(ESC [24 ~)
			9/11 3/2 3/4 7/14	(CSI 24 ~)
G13	F13 (LF)	0/10	1/11 5/11 3/2 3/5 7/14	(ESC [25 ~)
			9/11 3/2 3/5 7/14	(CSI 25 ~)
G14	F14	-	1/11 5/11 3/2 3/6 7/14	(ESC [26 ~)
			9/11 3/2 3/6 7/14	(CSI 26 ~)
G15	(F15) HELP	-	1/11 5/11 3/2 3/8 7/14	(ESC [28 ~)
			9/11 3/2 3/8 7/14	(CSI 28 ~)
G16	(F16) DO	-	1/11 5/11 3/1 3/9 7/14	(ESC [29 ~)
			9/11 3/2 3/9 7/14	(CSI 29 ~)
G20	F17	-	1/11 5/11 3/3 3/1 7/14	(ESC [31 ~)
			9/11 3/3 3/1 7/14	(CSI 31 ~)
G21	F18	-	1/11 5/11 3/3 3/2 7/14	(ESC [32 ~)
			9/11 3/3 3/2 7/14	(CSI 32 ~)
G22	F19	-	1/11 5/11 3/3 3/3 7/14	(ESC [33 ~)
			9/11 3/3 3/3 7/14	(CSI 33 ~)
G23	F20	-	1/11 5/11 3/3 3/4 7/14	(ESC [34 ~)
			9/11 3/3 3/4 7/14	(CSI 34 ~)

6.12 LOCAL FUNCTION KEYS

The Local Function Keys are the left five keys in the Function Key Row, having positions G99, G00, G01, G02, and G03. The codes for these keys shall not be transmitted by any terminal products to a host system. Their use is reserved exclusively for product specific local functions.

The use of Local Function Keys shall be implementation defined and registered with the architecture. When a previously defined function is implemented in a product, it shall be implemented as defined, and in the defined key position (insofar as this is possible where multiple definitions do not produce conflict).

See Appendix D "Documented Exceptions" for a description of the designated codes to be used for the Local Function Keys when they are implemented internally to a product, such as a Personal Computer or Workstation.

6.12.1 Hold Screen Key Operation

When the currently selected communications protocol supports some form of flow control, typing this key will toggle the screen hold/no hold. When the toggle is set to "no hold", the Hold Screen LED will be off, and screen operation will be normal. When the toggle is set to "Hold", the Hold Screen LED will be on, and the screen will freeze. (The screen will not be updated further and no more scrolling will occur). Typing the Hold Screen key when the screen is held will allow scrolling to begin again.

The Hold Screen function should be implemented so that it operates regardless of whether the Hold Screen key is depressed alone, or in combination with the Shift or Control Keys. In terminals that implement the Multiple Sessions extension, such as the VT330/340, pressing Hold Screen in combination with the CTRL Key will hold all sessions.

If the currently selected communications protocol does not support the Hold Screen function, the Hold Screen key will be inoperative.

See DEC STD 070-12 Terminal Synchronization for information on the interaction between Hold Screen and communications flow control.

6.12.2 Print Screen Key Operation

For a complete description of the printer modes and data transfer operations, see DEC STD 070-7 Printer Port Extension.

a. PRINT alone prints a copy of the cell text data in the region of the character cell display defined by Print Extent Mode (either the entire character cell display, or the character cell display scrolling region) to the printer port.

b. SHIFT/PRINT prints a copy of the entire character cell display bitmap to the printer port (binary transfer).

c. CONTROL/PRINT toggles Auto Print Mode.

6.12.3 Set-Up Key Operation

Depression of the Set-Up Key, either alone or in combination with the Shift or Control Keys, causes the terminal to enter into a local mode of operation in which certain terminal state may be changed by the User. Keyboard operation and screen display in Set-Up Mode is beyond the scope of this architecture specification, as is the means by which the terminal returns to normal operation, and the state which gets restored.

6.12.4 Local Function Key F4

6.12.4.1 Switch Session Key Operation - The Switch Session Key (F4) is used by terminals with the Multiple Sessions extension (VT330/340) to switch between sessions.

a. Switch Session or Shift/Switch Session instructs the terminal to activate another session. Refer to the Session Management chapter.

b. CTRL/Switch Session instructs the terminal to select one of a number of presentation states. Refer to the Presentation Management chapter.

6.12.4.2 Data/Talk Key Operation - Terminals with an integral modem (option on VT220), may use F4 as a Data/Talk switch for the modem. Pressing F4 alone will toggle between the Data and Talk modes. The Data mode is used for data communications once a connection is established. Talk mode is used for dialing or normal conversation with the attached telephone.

6.12.4.3 F4 Operation When Not Used For Local Function - If not used for a local function, F4 shall be dead (shall not transmit anything to the host).

6.12.5 Break Key Operation

For a complete description of the communications functions, see DEC STD 070-12 Terminal Synchronization, DEC STD 070-4 Terminal Management, and DEC STD 070-3 Code Extension Layer.

1. BREAK alone transmits a break if break is enabled (an optional set-up feature may be provided to disable the break function).

2. SHIFT/BREAK performs a disconnect if disconnects are enabled.

3. CONTROL/BREAK sends the terminal answerback message to the host.

6.13 MAIN KEY ARRAY - SPECIAL KEYS AND FUNCTIONS

The following keys are "special keys" on the Main Key array. They perform special actions or are otherwise unique. They are each found on all versions of the keyboard.

Four of the keys do not cause bytes to be immediately transmitted across the coding interface; rather, they condition the effect of other keys at the same time and/or subsequently. These keys are not auto-repeatable. The keys are:

1. Control
2. Shift
3. Lock
4. Compose

The remaining four keys send a particular ASCII control character or SPACE. All but the Return key are auto-repeatable. They are:

	Normal	Control	Shift	Lock
1. Space Bar	2/0	0/0	2/0	2/0
2. Return	0/13*	0/13*	0/13*	0/13*
3. Tab	0/9	0/9	0/9	0/9
4. Delete	7/15	1/8	7/15	7/15

* When New Line Mode is set (New Line On), the two-byte sequence 0/13 0/10 is transmitted.

6.13.1 Control Key Operation

When the Control key is held down while another key is depressed which forms a legal control combination, there is one keyclick and the appropriate control code is transmitted. All of the 32 ASCII C0 control codes plus DELETE (7/15) can be generated by typing the A-Z keys or the 2-8 keys independent of the Shift key or Lock key. In addition to the 2-8 keys, the keyboard provides alternative forms for keying the C0 controls for backward compatibility with DEC STD 107 and ANSI X4.14 (now superseded by X4.23-1982). On non-US keyboards, these alternate forms are provided whenever the keys involved in these combinations are present.

Valid control combinations auto repeat the same as other repeatable keys. If Control is pressed while a key is repeating, and the key forms a valid control combination, the key will continue to repeat but transmit the control combination while the Control key is pressed. If the key does not form a valid control combination, the key will stop repeating while the Control key is pressed.

When the Control key is depressed, the Shift key is ignored. On European keyboards with the numbers in shifted positions, the Ctrl-number construction does not require the use of shift. This applies to the number keys only. In the case where the numeral legends are shifted, and the number forms a valid control code, the unshifted legend is ignored. If the number has no meaning with CTRL, then the lower character on the key (such as [ ) is used. Exception: Tilde (E00) on the North American keyboard is used even though it is the upper character.

Guideline: On the French keyboard, typing CTRL/E06 (unshifted legend "[", shifted legend "6") will generate the control code for CTRL-6. On the Italian Keyboard in DP mode, CTRL-9 is not defined, so the key sends CTRL-[(ESC) because that is the unshifted legend on the key, and the "[" character forms a valid control combination.

The valid control combinations are standard across all keyboards. Refer to the "Control Codes and Keystrokes" section later in this chapter.

Invalid control combinations (any not listed below) will result in no keyclick occurring and no code being transmitted.

Note 1: E1 to E6 (the editing keypad), F6 to F20 (the top row function keys) and the cursor keys are dead when used with the CTRL key.

Note 2: The CTRL key is ignored when used with the numeric keypad keys.

Note 3: When using some communications protocols, certain of the control characters may not be allowed, for example, ctrl/S and ctrl/Q (XON and XOFF). In this case the control characters are considered as Local Function Keys. Conforming software should not use ctrl/S and ctrl/Q because of their possible use in communications flow control. See DEC STD 070-12 Terminal Synchronization for a complete description of the use of these characters.

6.13.2 Shift Key Operation

In order to obtain a shifted character, the Shift key and the graphic character key must be depressed together. Shift only applies to keys which produce graphic characters on the main key array. The state of the Shift key is ignored when any other key on the main array is depressed, including the Space Bar, Return, Tab, Delete keys. The use of the Control key supersedes the Shift key, so that the same control characters are generated whether the key is also shifted or not.

Shift has no effect on the codes transmitted by the Numeric Keypad keys and the Cursor Keys. When Function keys are depressed in combination with the Shift key, they transmit User Defined Key (UDK) sequences whose definitions have been previously received. The Editing keypad keys are inoperative when pressed in combination with the Shift key.

6.13.3 Lock Key Operation

The LOCK key is used to enter Shift Lock Mode or Caps Lock Mode. If a Shift Lock keyboard is selected in Set-Up: Shift Lock mode is turned on by pressing the LOCK key, and turned off by pressing either the LOCK key or a SHIFT key. If a Caps Lock keyboard is selected in Set-Up: Caps Lock Mode is toggled on or off by pressing the LOCK key.

The LOCK key operates at all times regardless of any other keys being held down.

The terminal user may set Caps/Shift Lock Mode to either Caps Lock Operation or Shift Lock Operation as a Set-Up feature. The factory default is Caps Lock Operation to be compatible with the VT100.

6.13.3.1 Shift Lock Operation -

When Caps/Shift Lock Mode is set to Shift Lock Operation all normal characters are acted on as if the Shift key were being depressed at the same time as the keystroke. The normal characters are those characters which are listed in the tables of alpha-numeric keyboard maps. Not included are control characters, tab, function keys, delete, return, compose, enter, lock, and shift keys. Note: when Shift Lock Operation is in effect, unless an implied or explicit compose sequence is used, it is not possible to generate the upper case characters using:

a. The keys that have one lower case letter legend and another legend.

b. The keys that have two lower case letter legends.

6.13.3.2 Caps Lock Operation -

When Caps/Shift Lock Mode is set to Caps Lock Operation, keys whose shifted and unshifted characters are the upper and lower case form of a single alphabetic letter transmit the shifted (upper case) character (as if the Shift key were being held down). Keys affected by Caps Lock are those with upper case alphabetic legends on the corresponding keyboard. Keys with one or two lower case (small letter) legends are not effected by Caps Lock Operation.

6.13.4 SPACE Bar Operation

The Space Bar usually causes the SPACE code (SP 2/0) to be transmitted. It can also be used in some explicit and implied compose sequences in order to represent a free standing accent as the equivalent ASCII code for a spacing character. Ctrl-space will send a NUL.

6.13.5 Return Key Operation

Depressing the Return key causes either a Carriage Return (CR, 0/13) or a Carriage Return/Line Feed pair (CR LF, 0/13 0/10) to be transmitted, depending on the state of New Line Mode.

6.13.5.1 New Line Mode -

A conforming device shall provide a means of selecting the codes transmitted as a result of depressing the Return and Enter keys on the keyboard. Normally these keys will both transmit a Carriage Return character (0/13) [reset state = no_new_line]. When this mode is in the set state [set state = new_line] these keys will both generate a two-code sequence 1/13 1/10 (CR LF).

```
NEW_LINE_MODE: (NEW_LINE_OFF,NEW_LINE_ON);
```

Note that New Line Mode is a function of the Character Cell Display architecture since it also causes receipt of LF to be interpreted as if it were CR LF, and is fully defined in that specification.

Note also that New Line Mode does not effect the code transmitted when the M key is pressed while holding the Control key. ctrl/M always transmits just 0/13.

Deviation Note: In the VT100, VT101, VT102, VT125, VT131, and VT132, the M key in combination with the Control Key IS effected by New Line Mode, and sends the sequence 0/13 0/10 when New Line Mode is in the set (New Line On) state.

6.13.6 Tab Key Operation

Typing the Tab key causes a horizontal tabulation code (HT, 0/9) to be transmitted.

6.13.7 Delete Key Operation

The Delete key usually causes a Delete code (DEL, 7/15) to be generated. The exception is that during an explicit or implied compose sequence it aborts the compose sequence in progress, cancels compose mode, and no code is transmitted. The delete key still clicks if keyclick is enabled.

Guideline: VT300 series terminals provide a Set-Up option to allow the Backarrow key "<X]" to transmit Delete (7/15) or Backspace (0/8).

6.13.8 Compose Key Operation

The Compose key permits generation of additional characters not present on the keyboard as a sequence of three key strokes. Its operation is described in the section on Compose Operation since the sequences include graphic keys. An optional set-up feature may be provided to disable the compose key.

6.13.9 Non-Spacing Diacritical Keys

On most non-US keyboards, certain accents are provided as Non-spacing Diacritical Keys, depending on the language or languages of the country. These keys serve both to enter an implied compose mode and to indicate the accent to be used with the following letter key. The operation of Non-Spacing Diacritical keys is further described in the "Compose Operation" subsection below.

6.14 MAIN KEY ARRAY - GRAPHIC CHARACTER KEYS

This section describes the graphic characters produced by keys in the main key array. Graphic character keys are all keys on the main key array which transmit character codes in the range 2/1 to 7/14 inclusive, and 10/0 to 15/15 inclusive. The Space bar transmits 2/0 and is used along with graphic characters in compose sequences.

The tables in this section associate each alpha-numeric key (described by position on the Keyboard Position Map) with a unique character (by name) for shifted and unshifted modes. The actual code produced is a function of the character encoding.

Level 3 terminals, and Level 2 terminals with the 8-bit Architecture Extension, support two different 8-bit multinational character encodings. Level 2 and Level 3 terminals with the NRCS Extension also support a number of 7-bit country specific (National) character encodings. At any time, only one of these character encodings is in use. The user can choose one of the following keyboard encodings via Set-Up:

- DEC Multinational (factory default)
  SET-UP: 8-bit Characters, UPSS DEC Multinational
  Primary keyboard set is ASCII
  Supplemental keyboard set is DEC Supplemental

- ISO Latin Alphabet Nr 1
  SET-UP: 8-bit Characters, UPSS ISO Latin-1
  Primary keyboard set is ASCII
  Supplemental keyboard set is ISO Latin-1 supplemental

- 7-bit ASCII character set
  SET-UP: 7-bit Characters, Data Processing Keys
  Primary keyboard set is ASCII
  No supplemental characters available

- 7-bit National Replacement Character Set (NRCS)
  SET-UP: 7-bit Characters, Typewriter Keys
  Primary keyboard set is one of 12 NRCS
  No supplemental characters available

The selection of a particular 7-Bit NRCS is bound to the selection of the keyboard dialect (Guideline: The VT320 uses 12 NRC sets to support 16 keyboard dialects). In 7-bit Characters mode, choosing Data Processing Keys always uses ASCII as the primary keyboard set.

Deviation Note

Older Level 2 terminals did not always use ASCII as the primary keyboard set when Data Processing Keys was selected (prior to the VT220CR). This use of a 7-bit NRCS other than ASCII in 7-bit Characters Data Processing mode has been deprecated.

For information on character sets and coding, refer to DEC STD 169.

Some character codes are included in these tables for characters whose coding might otherwise be confusing. These codes are given in row/column notation. The character encoding should be clear from the context as described above (8-bit Characters mode is always given in ISO Latin Alphabet Nr. 1).

There are four tables for each Keyboard Dialect:

a. 8-bit Characters, Typewriter Keys
b. 8-bit Characters, Data Processing Keys
c. 7-bit Characters, Typewriter Keys (NRCS)
d. 7-bit Characters, Data Processing Keys (ASCII)

Some keyboard dialects have changed to accomodate more recent country requirements. The most recent or recommended tables are listed first for each dialect, followed optionally by superseded tables included for reference.

A character name enclosed in square brackets [] indicates that the key is a Non-Spacing Diacritical key which when combined with another key forms a two key implied compose sequence.

The following rules of precedence are applied:

1. The unshifted character is generated when neither Control, Shift, Shift-Lock, nor Caps-Lock are in effect.

2. When Caps-Lock is in effect, and the Shifted and Unshifted characters are the upper and lowercase form of a single alphabetic letter, the shifted character is generated.

3. When Shift is in effect, it overrides Caps-Lock (always generates the shifted character).

4. When Control is in effect, it overrides Shift, Shift-Lock, and Caps-Lock. Refer to the "Control Codes and Keystrokes" subsection later in this chapter.

6.14.1 North American Keyboard (LK201-EE US/UK, LK201-NA, LK201-AA)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	` Open quote	~ Tilde character
E01	1	! Exclamation point
E02	2	@ Commercial at
E03**	3	# Number sign
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	^ Circumflex character
E07	7	& Ampersand
E08	8	* Asterisk
E09	9	(Opening parenthesis
E10	0	) Closing parenthesis
E11	- Hyphen, Minus	_ Underline
E12	= Equals	+ Plus
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	[Opening bracket	{ Opening brace
D12	] Closing bracket	} Closing brace
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	; Semicolon	: Colon
C11	' Apostrophe	" Quotation marks
C12	\ Backslash	| Vertical line
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	, Comma
B09	. Period	. Period
B10	/ Slash	? Question mark

North American Keyboard, 8-bit Characters, Data Processing Keys

Equivalent to 8-bit Characters, Typewriter Keys

North American Keyboard, 7-bit Characters, Typewriter Keys

7-bit Characters mode is not available with the North American keyboard dialect, although certain conditions can restrict the keyboard to sending 7-bit characters. In this case, the layout is the same as for 8-bit Characters.

North American Keyboard, 7-bit Characters, Data Processing Keys

Equivalent to 7-bit Characters, Typewriter Keys

The LK201-NA is for use with the VT330/340 series terminals and has front legends on the editing and numeric keypad not found on the LK201-EE.

There are three legends on E03 (LK201-EE LK201-NA combined US/UK), the shifted code generated by this key is dependent upon which dialect, North American or British, is selected in set-up.

6.14.2 British Keyboard (LK201-EE US/UK)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	` Open quote	~ Tilde character
E01	1	! Exclamation point
E02	2	@ Commercial at
E03**	3	£ British pound (10/3)
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	^ Circumflex character
E07	7	& Ampersand
E08	8	* Asterisk
E09	9	(Opening parenthesis
E10	0	) Closing parenthesis
E11	- Hyphen, Minus	_ Underline
E12	= Equals	+ Plus
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	[Opening bracket	{ Opening brace
D12	] Closing bracket	} Closing brace
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	; Semicolon	: Colon
C11	' Apostrophe	" Quotation marks
C12	\ Backslash	| Vertical line
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	, Comma
B09	. Period	. Period
B10	/ Slash	? Question mark

British Keyboard, 8-bit Characters, Data Processing keys

Equivalent to 8-bit Characters, Typewriter keys

British Keyboard, 7-bit Characters, Typewriter Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	£ British pound (2/3)

British Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	# Number sign (2/3)

There are three legends on E03, the shifted code generated by this key is dependent upon which dialect, North American or British, is selected in set-up.

6.14.3 British Keyboard (LK201-AE)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	[Tilde mark]	° Degree sign
E01	1	! Exclamation point
E02	2	" Double quote
E03	3	£ British pound (10/3)
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	& Ampersand
E07	7	' Apostrophe
E08	8	(Open parenthesis
E09	9	(Close parenthesis
E10	0	= Equals
E11	- Hyphen, Minus	_ Underline
E12	½ one-half	¼ one-qrtr
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	@ Commercial at	§ Section sign
D12	] Close bracket	[Open bracket
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	; Semicolon	+ Plus
C11	: Colon	* Asterisk
C12	[Circumflex accent]	[Grave accent]
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	, Comma
B09	. Period	. Period
B10	/ Slash	? Question mark

British Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	# Number sign (2/3)
D11	@	\ Backslash

British Keyboard, 7-bit Characters, Typewriter Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	~ Tilde	Dead
E03	3	£ British pound (2/3)
C12	^ Circumflex Character	` Open quote

British Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	~ Tilde	Dead
E03	3	Dead
D11	@	\ Backslash
C12	^ Circumflex Character	` Open quote

6.14.4 Flemish Keyboard (LK201-AB)

8-bit Characters, Typewriter Keys

Position	Unshifted		Shifted
E00	[Grave accent]		[Tilde mark]
E01	& Ampersand		1
E02	é Small e with acute	(14/9)	2
E03	" Quotation marks		3
E04	' Apostrophe		4
E05	(Opening parenthesis		5
E06	§ Section sign	(10/7)	6
E07	è small e with grave	(14/8)	7
E08	! Exclamation point		8
E09	ç Small c with Cedilla	(14/7)	9
E10	à Small a with grave	(14/0)	0
E11	) Closing parenthesis		° Degree sign (11/0)
E12	- Hyphen, Minus		_ Underline
D01	a		A
D02	z		Z
D03	e		E
D04	r		R
D05	t		T
D06	y		Y
D07	u		U
D08	i		I
D09	o		O
D10	p		P
D11	[Circumflex accent]		[Dieresis]
D12	$ Dollar sign		* Asterisk
C01	q		Q
C02	s		S
C03	d		D
C04	f		F
C05	g		G
C06	h		H
C07	j		J
C08	k		K
C09	l		L
C10	m		M
C11	ù Small u with grave	(15/9)	% Percent sign
C12	# Number sign		@ Commercial at sign
B00	< Less Than		> Greater than
B01	w		W
B02	x		X
B03	c		C
B04	v		V
B05	b		B
B06	n		N
B07	, Comma		? Question mark
B08	; Semicolon		. Period
B09	: Colon		/ Slash
B10	= Equals		+ Plus

Flemish Keyboard, 8-bit Characters, Data Processing Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E06	[Opening bracket	6
E07	] Closing bracket	7
C11	\ Backslash	% Percent sign

Flemish Keyboard, 7-bit Characters, Typewriter Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted		Shifted
E00	[Grave accent]		Dead
E02	é Small e with acute	(7/11)	2
E06	§ Section sign	(5/13)	6
E07	è small e with grave	(7/13)	7
E09	ç Small c with Cedilla	(5/12)	9
E10	à Small a with grave	(4/0)	0
E11	) Closing parenthesis		° Degree sign (5/11)
C11	ù Small u with grave	(7/12)	% Percent sign
C12	Dead		Dead

Flemish Keyboard, 7-bit Characters, Data Processing Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	` Open quote	~ Tilde character
E02	Dead	2
E06	[Opening bracket	6
E07	] Closing bracket	7
E09	Dead	9
E10	Dead	0
E11	) Closing parenthesis	Dead
D11	^ Circumflex character	" Quotation marks
C11	\ Backslash	% Percent sign

6.14.5 Canadian (French) Keyboard (LK201-AC)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	[Tilde mark]	° Degree sign (11/0)
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	/ Slash
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	? Question mark
E07	7	& Ampersand
E08	8	* Asterisk
E09	9	(Opening parenthesis
E10	0	) Closing parenthesis
E11	- Hyphen, Minus	_ Underline
E12	= Equals	+ Plus
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	ç Small c w/Cedilla (14/7)	Ç Capital C w/Cedilla (12/7)
D12	# Number sign	@ Commercial at
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	; Semicolon	: Colon
C11	[Grave accent]	[Circumflex accent]
C12	\ Backslash	| Vertical line
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	' Apostrophe
B09	. Period	. Period
B10	é Small e with acute (14/9)	É Capital E with acute (12/9)

Canadian (French) Keyboard, 8-bit Characters, Data Processing Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
D11	] Closing bracket	[Opening bracket

Canadian (French) Keyboard, 7-bit Characters, Typewriter Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
D11	ç Small c w/Cedilla (5/12)	Dead
D12	# Number sign	Dead
C12	Dead	Dead
B10	é Small e with acute (7/11)	Dead

Canadian (French) Keyboard, 7-bit Characters, Data Processing Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	~ Tilde char	Dead
D11	] Closing bracket	[Opening bracket
C11	` Open Quote	^ Circumflex Char
B10	Dead	Dead

6.14.6 Danish Keyboard (2nd, LK201-ED)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	~ Tilde Char	§ Section sign (10/7)
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	£ Pound sign (10/3)
E04	4	$ Dollar sign
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	' Apostrophe	` Open Quote
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/5)	Å Capital A with ring (12/5)
D12	" Quotation marks	^ Circumflex Char
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	æ Small ae dipthong (14/6)	Æ Capital AE dipthong (12/6)
C11	ø Small o with slash (15/8)	Ø Capital O with slash(13/8)
C12	' Apostrophe	* Asterisk
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Danish Keyboard, 8-bit Characters, Data Processing Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	~ Tilde Char	@ Commercial at
E03	3	# Number sign
D11	} Closing brace	] Closing bracket
C10	{ Opening brace	[Opening bracket
C11	| Vertical bar	\ Back slash

Danish Keyboard, 7-bit Characters, Typewriter Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	~ Tilde Char	@ Commercial at
E03	3	# Number sign
D11	å Small a with ring (7/13)	Å Capital A with ring (5/13)
C10	æ Small ae dipthong (7/11)	Æ Capital AE dipthong (5/11)
C11	ø Small o with slash (7/12)	Ø Capital O with slash (5/12)

Danish Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	~ Tilde Char	@ Commercial at
E03	3	# Number sign
D11	} Closing brace	] Closing bracket
C10	{ Opening brace	[Opening bracket
C11	| Vertical bar	\ Back slash

6.14.7 Danish Keyboard (1st, LK201-AD)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	[Tilde accent]	° Degree sign (11/0)
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign (10/7)
E04	4	$ Dollar sign
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	[Acute accent]	[Grave accent]
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/5)	Å Capital A with ring (12/5)
D12	[Dieresis]	[Circumflex accent]
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	æ Small ae dipthong (14/6)	Æ Capital AE dipthong (12/6)
C11	ø Small o with slash (15/8)	Ø Capital O with slash(13/8)
C12	' Apostrophe	* Asterisk
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Danish Keyboard, 8-bit Characters, Data Processing Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	@ Commercial at
D11	] Closing bracket	[Opening bracket
C11	# Number sign	\ Back slash

Danish Keyboard, 7-bit Characters, Typewriter Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	[Ring mark]
E03	3	Dead
E12	Dead	Dead
D11	å Small a with ring (7/13)	Å Capital A with ring (5/13)
D12	[Dieresis]	Dead
C10	æ Small ae dipthong (7/11)	Æ Capital AE dipthong (5/11)
C11	ø Small o with slash (7/12)	Ø Capital O with slash (5/12)

Danish Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	[Ring mark]
E03	3	Dead
E12	Dead	Dead
D11	Dead	Dead
D12	[Dieresis]	Dead
C10	æ Small ae dipthong (7/11)	Æ Capital AE dipthong (5/11)
C11	# Number sign	Dead

6.14.8 Finnish Keyboard (3rd, LK201-NX)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	~ Tilde Char	@ Commercial at
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign (10/7)
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	^ Circumflex Char	` Open Quote
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/5)	Å Capital A with ring (12/5)
D12	ü small u w/umlaut (15/12)	Ü Capital U w/umlaut (13/12)
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ö small o with umlaut (15/6)	Ö Capital O w/umlaut (13/6)
C11	ä small a with umlaut (14/4)	Ä Capital A w/umlaut (12/4)
C12	' Apostrophe	* Asterisk
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Finish Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	# Number sign
D11	} Closing brace	] Closing bracket
D12	~ Tilde character	^ Circumflex character
C10	| Vertical line	\ Backslash
C11	{ Opening brace	[Opening bracket

Finnish Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	@ Commercial at
E03	3	# Number Sign
E12	Dead	Dead
D11	å Small a with ring (7/13)	A Capital A with ring (5/13)
D12	ü small u with umlaut (7/14)	Ü Capital U w/umlaut (5/14)
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

Finnish Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	# Number sign
D11	} Closing brace	] Closing bracket
D12	~ Tilde character	^ Circumflex character
C10	| vert bar	\ Back slash
C11	{ Opening brace	[Opening bracket

6.14.9 Finnish Keyboard (2nd, LK201-NF)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	[Tilde mark]	@ Commercial at
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign (10/7)
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	[Circumflex accent]	[Grave accent]
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/5)	Å Capital A with ring (12/5)
D12	ü small u w/umlaut (15/12)	Ü Capital U w/umlaut (13/12)
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ö small o with umlaut (15/6)	Ö Capital O w/umlaut (13/6)
C11	ä small a with umlaut (14/4)	Ä Capital A w/umlaut (12/4)
C12	' Apostrophe	* Asterisk
B00	< Less than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Finish Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	# Number sign
D11	} Closing brace	] Closing bracket
C10	| Vertical line	\ Backslash
C11	{ Opening brace	[Opening bracket

Finnish Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	@ Commercial at
E03	3	# Number sign
E12	Dead	Dead
D11	å Small a with ring (7/13)	A Capital A with ring (5/13)
D12	ü small u with umlaut (7/14)	Ü Capital U w/umlaut (5/14)
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

Finnish Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	~ Tilde	@ Commercial at
E03	3	# Number sign
E12	^ Circumflex	` Open Quote
D11	} Closing brace	] Closing bracket
D12	Dead	Dead
C10	| Vertical line	\ Backslash
C11	{ Opening brace	[Opening bracket

6.14.10 Finnish Keyboard (1st, LK201-AF)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	[Tilde mark]	° Degree sign
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign (10/7)
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	[Circumflex accent]	[Grave accent]
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/5)	Å Capital A with ring (12/5)
D12	ü small u w/umlaut (15/12)	Ü Capital U w/umlaut (13/12)
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ö small o with umlaut (15/6)	Ö Capital O w/umlaut (13/6)
C11	ä small a with umlaut (14/4)	Ä Capital A w/umlaut (12/4)
C12	' Apostrophe	* Asterisk
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Finish Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	@ Commercial at
D11	] Closing bracket	[Opening bracket
D12	# Number sign	\ Backslash

Finnish Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
E03	3	Dead
E12	Dead	Dead
D11	å Small a with ring (7/13)	A Capital A with ring (5/13)
D12	ü small u with umlaut (7/14)	Ü Capital U w/umlaut (5/14)
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

Finnish Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
E03	3	Dead
E12	Dead	Dead
D11	Dead	Dead
D12	# Number sign	Dead
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

6.14.11 Austrian/German Keyboard (2nd, LK201-NG)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	^ Tilde char	° Circumflex char
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign (10/7)
E04	4	$ Dollar sign
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	ß German small sharp s (13/15)	? Question mark
E12	´ Apostrophe	` Open Quote
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	z	Z
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	ü small u w/umlaut (15/12)	Ü Capital U w/umlaut (13/12)
D12	+ Plus	* Asterisk
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ö small o with umlaut (15/6)	Ö Capital O w/umlaut (13/6)
C11	ä small a with umlaut (14/4)	Ä Capital A w/umlaut (12/4)
C12	# Number sign	' Apostrophe
B00	< Less Than	> Greater than
B01	y	Y
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Austrian/German Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E11	| Vertical bar	? Question mark
D11	@ Commercial at	\ Backslash
C10	} Close Brace	{ Open brace
C11	] Closing bracket	[Opening bracket

Austrian/German Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	^ Circumflex Char
E03	3	§ Section sign (4/10)
E11	ß German small sharp s (7/14)	? Question mark
D11	ü small u with umlaut (7/13)	Ü Capital U w/umlaut (5/13)
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

Austrian/German Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	Dead
E11	| Vertical bar	? Question mark
D11	@ Commercial at	\ Backslash
C10	} Closing Brace	{ Opening Brace
C11	] Closing bracket	[Opening bracket

6.14.12 Austrian/German Keyboard (1st, LK201-AG)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	[Tilde mark]	[Circumflex accent]
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign (10/7)
E04	4	$ Dollar sign
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	ß German small sharp s (13/15)	? Question mark
E12	[Acute accent]	[Grave accent]
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	z	Z
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	ü small u w/umlaut (15/12)	Ü Capital U w/umlaut (13/12)
D12	+ Plus	* Asterisk
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ö small o with umlaut (15/6)	Ö Capital O w/umlaut (13/6)
C11	ä small a with umlaut (14/4)	Ä Capital A w/umlaut (12/4)
C12	# Number sign	' Apostrophe
B00	< Less than	> Greater than
B01	y	Y
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Austrian/German Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
D11	@ Commercial at	\ Backslash
C11	] Closing bracket	[Opening bracket

Austrian/German Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	^ Circumflex char
E03	3	§ Section sign (4/10)
E11	ß German small sharp s (7/14)	? Question mark
E12	Dead	' Open quote
D11	ü small u w/umlaut (7/13)	Ü Capital U w/umlaut (5/13)
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

Austrian/German Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	^ Circumflex char
E03	3	§ Section sign (4/10)
E11	ß German small sharp s (7/14)	? Question mark
E12	Dead	' Open quote
D11	Dead	Dead
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	Dead	Dead

6.14.13 Dutch Keyboard (2nd, LK201-NH)

The Dutch keyboard is equivalent to the North American keyboard.

6.14.14 Dutch Keyboard (1st, LK201-AH)

8-bit Characters, Typewriter Keys

Position	Unshifted	Shifted
E00	[Tilde mark]	° Degree sign
E01	1	¼ one-qrtr
E02	2	½ one-half
E03	3	£ British pound
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	# Number sign
E07	7	& Ampersand
E08	8	(Opening parenthesis
E09	9	(Closing parenthesis
E10	0	§ Section sign
E11	' Apostrophe	" Quotation marks
E12	/ Slash	: Colon
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	[Circumflex accent]	[Dieresis]
D12	* Asterisk	! Exclamation mark
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	- Hyphen, Minus	_ Underline
C11	[Grave accent]	[Acute accent]
C12	@ Commercial at	| Vertical line
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	? Question mark
B09	. Period	; semicolon
B10	+ Plus	= Equals

Dutch Keyboard, 8-bit Characters, Data Processing Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E02	2	[Opening bracket
E03	3	] Closing bracket
E10	0	\ Backslash

Dutch Keyboard, 7-bit Characters, Typewriter Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
E01	1	¼ one-qrtr (7/13)
E02	2	½ one-half (5/12)
E03	3	£ British pound (2/3)
E06	6	Dead
E10	0	Dead
D11	^ Circumflex Character	" Dieresis
C11	` Open quote	' Apostrophe
C12	Dead	| Vertical line

Dutch Keyboard, 7-bit Characters, Data Processing Keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
E01	1	¼ one-qrtr (7/13)
E02	2	Dead
E03	3	Dead
E06	6	Dead
E10	0	Dead
C11	` Open quote	' Apostrophe
C12	Dead	| Vertical line

6.14.15 Italian Keyboard (LK201-AI)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	[Grave accent]	[Tilde mark]
E01	£ British pound sign (10/3)	1
E02	é Small e with acute (14/9)	2
E03	" Quotation marks	3
E04	' Apostrophe	4
E05	(Opening parenthesis	5
E06	_ Underline	6
E07	è small e with grave (14/8)	7
E08	[Circumflex accent]	8
E09	ç Small c with Cedilla (14/7)	9
E10	à Small a with grave (14/0)	0
E11	) Closing parenthesis	° Degree sign (11/0)
E12	- Hyphen, Minus	+ Plus
D01	q	Q
D02	z	Z
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	ì Small i with grave (14/12)	= Equals
D12	$ Dollar sign	& Ampersand
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	m	M
C11	ù Small u with grave (15/19)	% Percent sign
C12	* Asterisk	§ Section sign (10/7)
B00	< Less Than	> Greater than
B01	w	W
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	, Comma	? Question mark
B08	; Semicolon	. Period
B09	: Colon	/ Slash
B10	ò Small o with grave (15/2)	! Exclamation point

Italian Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E01	@ Commercial at	1
E02	# Number sign	2
E09	[Opening bracket	9
E10	] Closing bracket	0
C12	* Asterisk	\ Backslash

Italian Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	[Grave accent]	Dead
E01	£ British pound sign (2/3)	1
E02	é Small e with acute (5/13)	2
E07	è small e with grave (7/13)	7
E09	ç Small c w/Cedilla (5/12)	9
E10	à Small a with grave (7/11)	0
E11	) Closing parenthesis	° Degree sign (5/11)
D11	ì Small i with grave (7/14)	= Equals
C11	ù Small u with grave (6/0)	% Percent sign
C12	* Asterisk	§ Section sign (4/0)
B09	: Colon	/ Slash
B10	ò Small o with grave (7/12)	! Exclamation point

Italian Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	' Open Quote	~ Tilde Char
E01	@ Commercial at	1
E02	# Number sign	2
E07	Dead	7
E08	^ Circumflex Character	8
E09	[Opening bracket	9
E10	] Closing bracket	0
E11	) Closing parenthesis	Dead
D11	Dead	= Equals
C11	Dead	% Percent sign
C12	* Asterisk	\ Backslash
B09	: Colon	/ Slash
B10	Dead	! Exclamation point

6.14.16 Swiss (French) Keyboard (LK201-AK)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	! Exclamation mark	° Degree sign (11/0)
E01	1	+ Plus
E02	2	" Quotation marks
E03	3	* Asterisk
E04	4	ç Small c w/Cedilla (14/7)
E05	5	% Percent sign
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	' Apostrophe	? Question mark
E12	[Circumflex accent]	[Grave accent]
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	z	Z
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	è small e with grave (14/8)	ü small u w/umlaut (15/12)
D12	[Dieresis]	[Tilde mark]
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	é Small e with acute (14/9)	ö small o w/umlaut (15/6)
C11	à Small a with grave (14/0)	ä small a w/umlaut (14/4)
C12	$ Dollar sign	£ U.K. Pound (10/3)
B00	< Less Than	> Greater than
B01	y	Y
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Swiss (French) Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E04	4	@ Commercial at
D11	è small e with grave (14/8)	\ Backslash
C10	é Small e with acute (14/9)	[Opening bracket
C11	à Small a with grave (14/0)	] Closing bracket
C12	$ Dollar sign	# Number sign

Swiss (French) Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	! Exclamation mark	Dead
E04	4	ç Small c w/Cedilla (5/12)
D11	è small e with grave (5/15)	ü small u with umlaut (7/13)
D12	[Dieresis]	Dead
C10	é Small e with acute (5/11)	ö small o w/umlaut (7/12)
C11	à Small a with grave (4/0)	ä small a w/umlaut (7/11)
C12	$ Dollar sign	Dead
B10	- Hyphen, Minus	Dead

Swiss (French) Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	! Exclamation mark	Dead
E04	4	@ Commercial at
E12	^ Circumflex Char	`Open Quote
D11	Dead	\ Backslash
D12	" Quotation marks	~ Tilde Char
C10	Dead	[Opening bracket
C11	Dead	] Closing bracket
C12	$ Dollar sign	# Number sign

6.14.17 Swiss (German) Keyboard (LK201-AL)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	! Exclamation mark	° Degree sign (11/0)
E01	1	+ Plus
E02	2	" Quotation marks
E03	3	* Asterisk
E04	4	ç Small c w/Cedilla (14/7)
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	' Apostrophe	? Question mark
E12	[Circumflex accent]	[Grave accent]
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	z	Z
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	ü small u w/umlaut (15/12)	è small e with grave (14/8)
D12	[Dieresis]	[Tilde mark]
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ö small o with umlaut (15/6)	é Small e with acute (14/9)
C11	ä small a with umlaut (14/4)	à Small a with grave (14/0)
C12	$ Dollar sign	£ British pound sign (10/3)
B00	< Less Than	> Greater than
B01	y	Y
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Swiss (German) Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E04	4	@ Commercial at
D11	ü small u w/umlaut (15/12)	\ Backslash
C10	ö small o with umlaut (15/6)	[Opening bracket
C11	ä small a with umlaut (14/4)	] Closing bracket
C12	$ Dollar sign	# Number sign

Swiss (German) Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	! Exclamation mark	Dead
E04	4	ç Small c w/Cedilla (5/12)
D11	ü small u with umlaut (7/13)	è small e with grave (5/15)
D12	[Dieresis]	Dead
C10	ö small o with umlaut (7/12)	é Small e with acute (5/11)
C11	ä small a with umlaut (7/11)	à Small a with grave (4/0)
C12	$ Dollar sign	Dead
B10	- Hyphen, Minus	Dead

Swiss (German) Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	! Exclamation mark	Dead
E04	4	@ Commercial at
E12	^ Circumflex Char	` Open Quote
D11	Dead	\ Backslash
D12	" Quotation marks	~ Tilde Char
C10	Dead	[Opening bracket
C11	Dead	] Closing bracket
C12	$ Dollar sign	# Number sign

6.14.18 Swedish Keyboard (2nd, LK201-NM)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	~ Tilde Char	° Degree sign (11/0)
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign (10/7)
E04	4	$ Dollar sign
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	^ Circumflex Char	` Open Quote
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/15)	Å Capital A with ring (12/5)
D12	ü small u with umlaut (15/12)	Ü Capital U w/Umlaut (13/12)
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ö small o with umlaut (15/6)	Ö Capital O w/umlaut (13/6)
C11	ä small a with umlaut (14/4)	Ä Capital A w/umlaut (12/4)
C12	' Apostrophe	* Asterisk
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Swedish Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	@ Commercial at
D11	] Closing bracket	[Opening bracket
D12	# Number sign	\ Backslash
C10	} Close brace	{ Open brace
C11	| Vertical line	| Vertical line

Swedish Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
E03	3	Dead
E12	Dead	Dead
D11	å Small a with ring (7/13)	Å Capital A with ring (5/13)
D12	ü small u with umlaut (7/14)	Ü Capital U w/Umlaut (5/14)
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

Swedish Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	~ Tilde Char	Dead
E03	3	@ Commercial at
D11	] Closing bracket	[Opening bracket
D12	# Number sign	\ Backslash
C10	} Close Brace	{ Open Brace
C11	| Vertical line	| Vertical line

6.14.19 Swedish Keyboard (1st, LK201-AM)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	[Tilde mark]	° Degree sign (11/0)
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign (10/7)
E04	4	$ Dollar sign
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	[Circumflex accent]	[Grave accent]
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/15)	Å Capital A with ring (12/5)
D12	ü small u with umlaut (15/12)	Ü Capital U w/Umlaut (13/12)
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ö small o with umlaut (15/6)	Ö Capital O w/umlaut (13/6)
C11	ä small a with umlaut (14/4)	Ä Capital A w/umlaut (12/4)
C12	' Apostrophe	* Asterisk
B00	< Less Than	> Greater than
B01	ż	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Swedish Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	@ Commercial at
D11	] Closing bracket	[Opening bracket
D12	# Number sign	\ Backslash

Swedish Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
E03	3	Dead
E12	Dead	Dead
D11	å Small a with ring (7/13)	Å Capital A with ring (5/13)
D12	ü small u with umlaut (7/14)	Ü Capital U w/umlaut (5/14)
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

Swedish Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
E03	3	Dead
E12	Dead	Dead
D11	Dead	Dead
D12	# Number sign	Dead
C10	ö small o with umlaut (7/12)	Ö Capital O w/umlaut (5/12)
C11	ä small a with umlaut (7/11)	Ä Capital A w/umlaut (5/11)

6.14.20 Norwegian Keyboard (2nd, LK201-EN)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	~ Tilde Char	@ Commercial at
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	# Number sign
E04	4	$ Dollar sign
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	' Apostrophe	` Open quote
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/5)	Å Capital A w/ring (12/5)
D12	" Quotation marks	^ Circumflex Character
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ø Small o with slash (15/8)	Ø Capital O w/slash (13/8)
C11	æ Small ae dipthong (14/6)	Æ Capital AE dipthong(12/6)
C12	' Apostrophe	* Asterisk
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Norwegian Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
D11	} Closing brace	] Closing bracket
C10	| Vertical bar	\ Backslash
C11	{ Opening brace	[Opening bracket

Norwegian Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
D11	å Small a with ring (7/13)	Å Capital A w/ring (5/13)
C10	ø Small o with slash (7/12)	Ø Capital O w/slash (5/12)
C11	æ Small ae dipthong (7/11)	Æ Capital AE dipthong (5/11)
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Norwegian Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
D11	} Closing brace	] Closing bracket
C10	| Vertical bar	\ Backslash
C11	{ Opening brace	[Opening bracket

6.14.21 Norwegian Keyboard (1st, LK201-AN)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	[Tilde mark]	° Degree sign
E01	1	! Exclamation mark
E02	2	" Quotation marks
E03	3	§ Section sign
E04	4	$ Dollar sign
E05	5	% Percent
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) Closing parenthesis
E10	0	= Equals
E11	+ Plus	? Question mark
E12	[Acute accent]	[Grave accent]
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	å Small a with ring (14/5)	Å Capital A w/ring (12/5)
D12	[Dieresis]	[Circumflex accent]
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ø Small o with slash (15/8)	Ø Capital O w/slash (13/8)
C11	æ Small ae dipthong (14/6)	Æ Capital AE dipthong(12/6)
C12	' Apostrophe	* Asterisk
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Norwegian Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E03	3	@ Commercial at
D11	] Closing bracket	[Opening bracket
C11	# Number sign	\ Backslash

Norwegian Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	[Ring mark]
E03	3	Dead
E12	Dead	Dead
D11	å Small a with ring (7/13)	Å Capital A w/ring (5/13)
D12	[Dieresis]	Dead
C10	ø Small o with slash (7/12)	Ø Capital O w/slash (5/12)
C11	æ Small ae dipthong (7/11)	Æ Capital AE dipthong(5/11)

Norwegian Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	[Ring mark]
E03	3	Dead
E12	Dead	Dead
D11	Dead	Dead
D12	[Dieresis]	Dead
C10	ø Small o with slash (7/12)	Ø Capital O w/slash (5/12)
C11	# Number sign	Dead

6.14.22 Belgian/French Keyboard (LK201-AP)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	[Grave accent]	[Tilde mark]
E01	& Ampersand	1
E02	é Small e with acute (14/9)	2
E03	" Quotation marks	3
E04	' Apostrophe	4
E05	(Opening parenthesis	5
E06	§ Section sign (10/7)	6
E07	è small e with grave (14/8)	7
E08	! Exclamation point	8
E09	ç Small c w/Cedilla (14/7)	9
E10	à Small a with grave (14/0)	0
E11	) Closing parenthesis	° Degree sign (11/0)
E12	- Hyphen, Minus	_ Underline
D01	a	A
D02	z	Z
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	[Circumflex accent]	[Dieresis]
D12	$ Dollar sign	* Asterisk
C01	q	Q
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	m	M
C11	ù Small u with grave (15/9)	% Percent sign
C12	# Number sign	@ Commercial at sign
B00	< Less Than	> Greater than
B01	w	W
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	, Comma	? Question mark
B08	; Semicolon	. Period
B09	: Colon	/ Slash
B10	= Equals	+ Plus

Belgian/French Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E06	[Opening bracket	6
E07	] Closing bracket	7
C11	\ Backslash	% Percent sign

Belgian/French Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	[Grave accent]	Dead
E02	é Small e with acute (7/11)	2
E06	§ Section sign (5/13)	6
E07	è small e with grave (7/13)	7
E09	ç Small c w/Cedilla (5/12)	9
E10	à Small a with grave (4/0)	0
E11	) Closing parenthesis	° Degree sign (5/11)
C11	ù Small u with grave (7/12)	% Percent sign
C12	Dead	Dead

Belgian/French Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	' Open Quote	~ Tilde Char
E02	Dead	2
E06	[Opening bracket	6
E07	] Closing bracket	7
E09	Dead	9
E10	Dead	0
E11	) Closing parenthesis	Dead
D11	^ Circumflex Char	" Quotation marks
C11	\ Backslash	% Percent sign

6.14.23 Spanish Keyboard (LK201-AS)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	¿ Inverted Question (11/15)	¡ Inverted exclamation (10/1)
E01	1	! Exclamation point
E02	2	" Quotation marks
E03	3	# Number sign
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	& Ampersand
E07	7	/ Slash
E08	8	(Opening parenthesis
E09	9	) closing parenthesis
E10	0	= Equals
E11	' Apostrophe	? Question mark
E12	º Masc. ordinal ind (11/10)	ª Fem. ordinal ind (10/10)
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	[Grave accent]	[Circumflex accent]
D12	+ Plus	* Asterisk
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ñ Small n with Tilde (15/1)	Ñ Capital N w/Tilde (13/1)
C11	[Acute accent]	[Dieresis]
C12	ç Small c w/Cedilla (14/7)	[Tilde mark]
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	- Hyphen, Minus	_ Underline

Spanish Keyboard, 8-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	@ Commercial at	\ Backslash
E12	] Closing bracket	[Opening bracket

Spanish Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	¿ Inverted Question (5/13)	¡ Inverted exclamation (5/11)
E03	3	Dead
E12	Dead	Dead
C10	ñ Small n w/Tilde (7/12)	Ñ Capital N w/Tilde (5/12)
C12	ç Small c w/Cedilla (7/13)	[Tilde mark]

Spanish Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	@ Commercial at	\ Backslash
E12	] Closing bracket	[Opening bracket
D11	` Open quote	^ Circumflex character
C10	Dead	Dead
C11	' Apostrophe	" Quotation marks
C12	Dead	~ Tilde character

6.14.24 Portuguese Keyboard (LK201-AV)

8-bit Characters, Typewriter keys

Position	Unshifted	Shifted
E00	\ Backslash	| Vertical line
E01	1	! Exclamation point
E02	2	@ Commercial at
E03	3	# Number sign
E04	4	$ Dollar sign
E05	5	% Percent sign
E06	6	" Quotations
E07	7	& Ampersand
E08	8	* Asterisk
E09	9	(Opening parenthesis
E10	0	) Closing parenthesis
E11	- Hyphen, Minus	_ Underline
E12	= Equals	+ Plus
D01	q	Q
D02	w	W
D03	e	E
D04	r	R
D05	t	T
D06	y	Y
D07	u	U
D08	i	I
D09	o	O
D10	p	P
D11	[Acute accent]	[Grave accent]
D12	] Closing bracket	} Closing brace
C01	a	A
C02	s	S
C03	d	D
C04	f	F
C05	g	G
C06	h	H
C07	j	J
C08	k	K
C09	l	L
C10	ç Small c w/cedilla (14/7)	Ç Capital C w/cedilla (12/7)
C11	[Tilde mark]	[Circumflex accent]
C12	[Opening bracket	{ Opening brace
B00	< Less Than	> Greater than
B01	z	Z
B02	x	X
B03	c	C
B04	v	V
B05	b	B
B06	n	N
B07	m	M
B08	, Comma	; Semicolon
B09	. Period	: Colon
B10	/ Slash	? Question mark

Portuguese Keyboard, 8-bit Characters, Data Processing keys

8-bit Characters Data Processing Keys are the same as 8-bit Characters Typewriter Keys.

Portuguese Keyboard, 7-bit Characters, Typewriter keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
E00	Dead	Dead
D12	Dead	Dead
C10	ç Small c w/Cedilla (7/12)	Ç Capital C w/Cedilla (5/12)
C12	Dead	Dead

Portuguese Keyboard, 7-bit Characters, Data Processing keys
(Where different from 8-bit Characters, Typewriter keys)

Position	Unshifted	Shifted
D11	' Apostrophe	` Open quote
C10	Dead	Dead
C11	~ Tilde character	^ Circumflex character

6.14.25 DECmate/WPS Main Key Array

The LK201-Bx Main Key Array is identical to LK201- Ax class, where x specifies the country, except that the Delete Key (E13) has the words Word and Char instead of Delete along with the "delete backward" symbol. In addition, the LK201-Bx main key array has the following gold legends on the front edge of the keys as shown below for the North American Version LK201-BA.

Id	Lower	Upper	Front edge
E00	` open quote	~ tilde	Halt
E01	1	! exclamation mark	
E02	2	@ Commercial at	
E03	3	# number sign	
E04	4	$ dollar sign	
E05	5	% percent	
E06	6	^ circumflex	
E07	7	& ampersand	
E08	8	* asterisk	
E09	9	(open paren	
E10	0	) close paren	
E11	- minus	_ underline	Print Hyph
E12	= equals	+ plus	Abbrv
D01	Q		Super Script
D02	W		
D03	E		
D04	R		Ruler
D05	T		Top Docmt
D06	Y		
D07	U		
D08	I		
D09	O		
D10	P		Page Marker
D11	[open bracket	{ open brace	Cmnd
D12	] close bracket	} close brace	
C01	A		Sub Script
C02	S		
C03	D		Dead Key
C04	F		File Docmt
C05	G		Get Docmt
C06	H		
C07	J		
C08	K		
C09	L		Libry
C10	; semicolon	: colon	
C11	' apostrophe	" double quote	Replc
C12	\ backslash	| vertical line	Date & Time
C13	Return		Para marker
B00	< less than	> greater than	
B01	Z		
B02	X		

LK201- BA Main Key Array Continued

Id	Lower	Upper	Front edge
B03	C		Center
B04	V		View
B05	B		Bot Docmt
B06	N		New Page
B07	M		Menu
B08	, comma	, comma	Srch
B09	. period	. period	Cont Srch
B10	/ slash	? question mark	Cont Srch & Sel

The LK201-PA is identical to the LK201-NA and the LK201-PE is identical to the LK201-RE except for the gold legends on the front edge. Note that the gold legends on the front edge of the LK201-Px are not identical to the LK201- Bx.

The difference between the LK201-RE and the LK201-NA is the front legends on the editing and numeric keypads. Therefore, the example below shows the LK201- Px, where x can be either {A, E}.

6.14.25.1 English WPS- LK201- {PA, PE} -

	Legend		
Id	Lower	Upper	Front Edge
E00	` open quote	~ tilde ch	
E01	1	! exclamation mark	
E02	2	@ Commercial at	
E03	3	# number sign	
E03**	3	£	
E04	4	$ dollar sign	
E05	5	% percent	
E06	6	^ circumflex	
E07	7	& ampersand	
E08	8	* asterisk	
E09	9	(open paren	
E10	0	) close paren	
E11	- minus	_ underline	Print Hyph
E12	= equals	+ plus	Abbrv
E13	<X]		Rub Line
D00	Tab		Indent Tab
D01	Q		Super Script
D02	W		Write Docmt
D03	E		
D04	R		Ruler
D05	T		Top Docmt
D06	Y		Footnote
D07	U		
D08	I		
D09	O		
D10	P		Page Marker
D11	[open bracket	{ open brace	Cmnd
D12	] close bracket	} close brace	Change Editor

** There are three legends on E03, the shifted code generated by this key is dependent upon which dialect, North American or British, is selected in set-up.

LK201- {PA, PE} Main Key Array Continued

Id	Lower	Upper	Front edge
C01	A		Sub Script
C02	S		Spell Check
C03	D		Dead Key
C04	F		File Docmt
C05	G		Get Docmt
C06	H		
C07	J		Lang Aids
C08	K		
C09	L		Libry
C10	; semicolon	: colon	Global Replc
C11	' apostrophe	" double quote	Replc
C12	\ backslash	| vertical line	Date & Time
C13	Return		Para Marker
B00	< less than	> greater than	
B01	Z		Status
B02	X		
B03	C		Center
B04	V		View
B05	B		Bot Docmt
B06	N		New Page
B07	M		Menu
B08	, comma	, comma	Srch
B09	. period	. period	Cont Srch
B10	/ slash	? question mark	Cont Srch & Sel

6.15 COMPOSE OPERATION

Compose is used to generate characters which cannot be typed directly from the keyboard. Since there are many keyboards, some characters will be producible both via compose and via direct keyboard input. Characters often generated by compose include many of the accented European characters, special symbols such as the Yen Sign, and other normal characters not available on the particular keyboard.

There are two forms of compose sequences:

1. Implied compose sequence - a two key sequence initiated by a Non-Spacing Diacritical key.

2. Explicit compose sequence - a three key sequence initiated by the Compose key.

Graphic character keys used during an implied or explicit compose sequences are not auto-repeatable. This prevents the terminal user from accidentally repeating a key during a compose sequence.

6.15.1 Compose With The Use Of The COMPOSE Key

All characters which are not directly available on the keyboard can be generated with the use of the COMPOSE key followed by a two character generating sequence called a compose sequence. Many composable characters have more than one compose sequence, and can therefore be composed in more than one way. To compose a character the user types the following 3 keystrokes:

1. COMPOSE
2. Character 1 (or Mark)*
3. Character 2 (or Mark)*

* Mark is used to mean a non-spacing diacritical mark or accent.

When the user types the COMPOSE key, the COMPOSE LED is turned on to indicate that a compose is in progress. When the compose ends either successfully, by the depression of the delete key, or with an error, then the COMPOSE LED is turned off. If the Character 1 and Character 2 pair form a valid compose sequence, then the compose is successful and the composed character is sent. If the Character 1 and Character 2 pair do not form a valid compose sequence then the Warning Bell rings once, if enabled in Set-up, to indicate a compose error, and no character is sent.

6.15.2 Compose With The Use Of Non-Spacing Diacritical Marks

Many characters which are not directly available on the keyboard can be generated with the use of the implied compose feature. Non-Spacing Diacritical keys do not independently transmit character codes, but shall be used in combination with other keys. When these marks are typed an "implied compose" begins with the non-spacing diacritical mark as the first half of the compose sequence. Such an implied compose consists of two keystrokes as follows:

1. Non-Spacing Diacritical Mark
2. Character

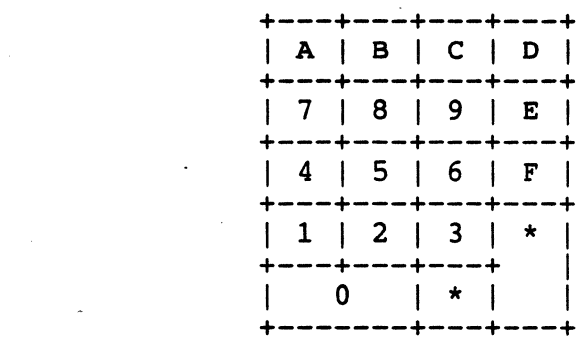
When the user types the non-spacing diacritical mark, the COMPOSE LED is turned on to indicate that a compose is in progress. When the compose ends either successfully, by depression of the delete key, or with an error, then the COMPOSE LED is turned off. If the non-spacing diacritical mark and character pair form a valid compose sequence, then the compose is successful and the composed character is sent. If the non-spacing diacritical mark and character pair do not form a valid compose sequence then the Warning Bell rings once, if enabled in Set-up, to indicate a compose error, and no character is sent.

6.15.3 Composing Arbitrary 8-bit Characters (VT330/340 Only, Not Mandatory)

When the user types the COMPOSE key, and follows that key with two keys from the numeric keypad, an arbitrary 8-bit character is generated based on which numeric keypad keys are pressed. The two keys pressed specify an 8-bit code in hexadecimal digits: the 0 through 9 keys specify the hexadecimal digits 0 through 9; the hexadecimal digits A through F are specified by the following keys:

Key Name	Hexadecimal Digit
PF1	A
PF2	B
PF3	C
PF4	D
Minus	E
Comma	F

The entire numeric keypad can then be thought of as a hexadecimal keypad:



* pressing either the Enter or Period keys will abort the compose sequence, and cause the function key to be sent to the host.

Notes:

1. Mixing a numeric keypad key and a main keypad key is not allowed, and will cause an error condition: the COMPOSE is aborted and the Warning Bell is rung.

2. Arbitrary 8-bit composition obeys the C1 transmission state. If 7-Bit Controls are enabled, and C1 control generated by using compose on the numeric keypad is sent in the 7-Bit ESC Fe form.

3. In VT52, VT100, and 4010/4014 modes, or in National character set mode, full 8-bit characters may not be generated by using compose on the numeric keypad. If the VT330/340 is in one of these modes, and the first key hit on the numeric keypad is 8, 9, PF1, PF2, PF3, PF4, Minus, or Comma, the COMPOSE is aborted and the Warning Bell is rung.

4. This function is not affected by the setting of DECNKM (Numeric Keypad mode).

6.15.4 Use Of The COMPOSE Key When A Compose Is Already In Progress

If the compose key is pressed while a compose sequence is in progress then the current compose sequence will be terminated and a new sequence will begin. The compose indicator remains on, the warning bell is not rung, no code is transmitted and a keyclick is generated if enabled.

6.15.5 Use Of The DELETE Key When A Compose Is Already In Progress

Typing the DELETE key will abort the compose immediately. No character is sent, the COMPOSE LED is turned off and the bell is not rung.

6.15.6 Keystrokes Which Abort Compose Immediately

A keystroke on any of the following keys aborts the compose in progress, does not ring the bell, sends no character codes, turns off the COMPOSE LED, and performs the local function normally performed by the keystroke:

- Hold Screen Key
- Print Screen Key
- Set-Up Key
- Switch Session Key
- Break Key

A Keystroke on any of the following keys (or key combinations) aborts the compose in progress, does not ring the bell, does not send the character code for any character entered earlier in the compose sequence, and sends the character code or codes normally transmitted by the keystroke which aborts the sequence:

- Keypad keys
- Function keys F6 through F20 including HELP and DO
- Tab key
- Return key
- Ctrl-A through Ctrl-Z and all other Ctrl-Key combinations which produce control codes
- Edit keypad keys
- ESC, BS and LF (F11, F12 and F13 in VT100 mode)

The Delete key aborts the compose, does not ring the bell, sends no character codes and turns off the COMPOSE LED.

6.15.7 Keystrokes Which Do Not Affect Compose Directly

The following keys do not affect compose directly but they may affect compose indirectly. Their actions are the normal keyboard actions:

- Shift
- Control (When held alone)

6.15.8 Order And Case Within Compose Sequences

The importance of the order and case (upper or lower) of the two characters used to form a compose sequence depends on the character being composed.

6.15.8.1 Order -

The sequences for characters which have an obvious order (e.g., the "OE" ligature or the "¼" fraction) may not be reversed, while those for other characters may be. When the Compose key is used to initiate the sequence, the two following keys may be depressed in either order, unless the table entry for the composite character says "in order". When a Non-Spacing Diacritical key is used to initiate the sequence (implied compose), the diacritical key precedes the other key in the sequence.

6.15.8.2 Case -

The sequences for generating alphabetic characters are case sensitive, while those for other symbols are not. In instances where two alphabetic keys are used in the sequence, the minimum conforming implementation requires that the sequences of both keys in upper case and both keys in lower case be provided. It is optional to also provide the sequences in which one key is upper case and the other lower case.

6.15.9 Composition Conventions

Mark is used to mean non-spacing diacritical mark. Some marks are also called accents. Each non-spacing diacritical mark has a character associated with it in the 8-bit Multinational character sets. Note that the mark, or accent, is not strictly the equivalent of its associated character.

Mark	Associated Character
Tilde mark	~ Tilde character
Acute accent	' Apostrophe
Circumflex accent	^ Circumflex character
Ring mark	° (degree sign) or asterisk
Dieresis or umlaut mark	" Double quote
Grave accent	` Open quote
Cedilla	, Comma

Grave accent, tilde, and circumflex appear on all keyboards Diaeresis/umlaut mark and acute accent appear on certain non-US keyboards and are always Non-Spacing Diacritical keys. To distinguish diaeresis/umlaut mark from quotation marks (") which both appear on some keyboards, the diaeresis/umlaut mark is shown as two raised dots and quotation marks are short vertical lines on the key legends. Similarly, the acute accent is slanted and the apostrophe is shown vertical on the key legends.

6.15.9.1 Compose Sequence Error Conditions -

There are three different classes of compose sequence errors that may occur.

The first class consists of dead keys which do not transmit any code. These include invalid control combinations, such as ctrl-9, and any other keys that are dead in the current operating environment. Dead keys are ignored within a compose sequence.

The second class consists of sequences prematurely terminated by an illegal character. Illegal characters are all valid control combinations and all keys not from the main array of the keyboard. This includes Tab, Return, the Cursor Keys, keys from the Editing Keypad, keys from the Numeric Keypad, and the function keys (including the Local Function Keys). Such compose sequences are terminated as soon as the illegal character/key is typed. The compose led is turned off, and compose mode is exited. The offending character/key is considered to be outside of the sequence, and is handled normally (that is, a keyclick is generated, if enabled, and the code for the offending character/key is transmitted). The presence or absence of a keyclick, when enabled, indicates to the terminal user whether anything has been transmitted to the host or not.

The third class of compose errors consists of sequences containing legal characters from the main array of the keyboard but which do not form a valid compose sequence. Such compose sequences are terminated after the second character is typed. The procedure involves sounding the error bell (if it is enabled), turning off the Compose LED, and exiting compose mode. No code is transmitted by the terminal and nothing is displayed on the video display for the invalid compose sequence. The audible sound for a compose failure may be distinguishable from the sound produced by the receipt of the BEL control character and may be separately disabled via Set-up as an extension to Level 2.

6.15.9.2 Syntax Of Compose Sequences -

The following syntax diagrams show all possible forms of valid and invalid explicit and implied compose sequences. All keys are divided into the following disjoint equivalence classes:

Symbol	Meaning
C	Compose Key
d	Non-Spacing Diacritical key
a	valid first key after Compose key
b	valid second key after Compose key or first key after a Non-Spacing Diacritical key
x	an invalid first key after Compose key
y	an invalid second key after Compose key or first key after a Non-Spacing Diacritical key
F	any function key or control character on the main key array or any other key outside of the main key array (including Local Function Keys)
D	Delete key
c	is the resulting composed one-byte character

Syntax	Action		
	Transmit	LED	bell
Valid:			
C a b	c	off	no
C a d	c	off	no
C d b	c	off	no
d b	c	off	no
Corrected by terminal user:			
C a D	nothing	off	no
C d D	nothing	off	no
C x D	nothing	off	no
C D	nothing	off	no
d D	nothing	off	no
Started over by terminal user:			
C a C	nothing	on	no
C d C	nothing	on	no
C x C	nothing	on	no
C C	nothing	on	no
d C	nothing	on	no
errors:			
C a y	nothing	off	yes
C x y	nothing	off	yes
C x b	nothing	off	yes
C x d	nothing	off	yes
C d y	nothing	off	yes
C a F	F	off	no
C d F	F	off	no
C x F	F	off	no
C F	F	off	no
d F	F	off	no
d d	nothing	off	yes
d y	nothing	off	yes

6.15.9.2.1 Compose Sequence Processing - The following algorithm defines the sequence of events which shall occur during the processing of an explicit or implied compose sequence. The algorithm is entered by the depression of the Compose key or a Non-Spacing Diacritical key. The Pascal coding which follows provides a detailed specification of the compose sequence logic.

STEP 1: Process the first keystroke after the Compose key or process the Non-Spacing Diacritical keystroke. If the keystroke is:

(1) Any graphic character key, Space Bar, or Non-Spacing Diacritical key: proceed to STEP 2.
(2) A control character or any key not on the main key array: turn off the compose indicator, and exit compose mode. Then keyclick (if enabled) and transmit the code or sequence for the key just pressed.
(3) Delete: turn off the compose indicator and exit compose mode. Keyclick (if enabled). Note that the error bell is not sounded and nothing is transmitted.
(4) Compose: restart the compose sequence, keyclick (if enabled), and transmit nothing to the host until the new sequence is completed successfully.

STEP 2: Process the second key after the Compose key or the key following a Non-Spacing Diacritical key. If the keystroke is:

(1) A valid graphic character key or Non-Spacing Diacritical key (as defined in the tables) and if the preceding key is also valid: turn off the compose indicator, keyclick (if enabled), and transmit the corresponding DEC Multinational character code.
(2) An invalid graphic character key or Non-Spacing Diacritical key, or the preceding key was invalid: ring the error bell, do not key click, turn off the compose indicator, and exit compose mode.
(3) A control character or any key not on the main key array: turn off the compose indicator, and exit compose mode. Then keyclick (if enabled) and transmit the code or sequence for the key just pressed.
(4) Delete: turn off the compose indicator and exit compose mode. Keyclick (if enabled). Note that the error bell is not sounded.
(5) Compose: restart the compose sequence, keyclick (if enabled), and transmit nothing until the new sequence is completed successfully.

```
PROCEDURE PROCESS_COMPOSE_SEQUENCE (KEY: KEY_CODE);
(*
  This procedure is called whenever the Compose key or a Dead
  Diacritical keystroke occurs and the terminal is not already
  in compose mode.

  The key code of the initiating key is passed in the call.
  The procedure calls the following external routines, which
  should be self-explanatory:

          key := GET_NEXT_KEY (KEY_CODE);
          TURN_ON_COMPOSE_LED;
          TURN_OFF_COMPOSE_LED;
          RING_ERROR_BELL;
          GENERATE_KEY_CLICK;
          TRANSMIT_COMPOSED_CHAR (KEY_1, KEY_2: KEY_CODE);
          TRANSMIT_CONTROL_FUNCTION (KEY_1);

  It also calls the function

    VALID_COMPOSE_SEQUENCE(KEY_1, KEY_2: KEY_CODE,
            SUPPLEMENTAL_SET: VALID_SUPPLEMENTAL_SETS): BOOLEAN;

  which returns True if the two keys form a valid compose
  sequence as defined in the tables in this section.
*)

CONST     COMPOSE = (* the key code representing the Compose key *)

          DELETE = (* the key code representing the Delete key *)

TYPE      KEY_CODE = 0..255;

          SPACE_GRAPHIC_DIACRITICAL_SET = SET OF (* The key codes
            representing SPACE and graphic characters from 2/0
            to 7/14 inclusive, 10/1 to 15/14 inclusive, and the
            Non-Spacing Diacritical characters *);

VAR       KEY_1, KEY_2: KEY_CODE;
          IN_COMPOSE_MODE: BOOLEAN;
```

```
BEGIN
TURN_ON_COMPOSE_INDICATOR;
KEY_1 := KEY;
IN_COMPOSE_MODE:= TRUE;    (* set loop flag, clear to exit loop *)
WHILE IN_COMPOSE_MODE DO   (* restart on Compose key *)
  BEGIN
  WHILE KEY_1 = COMPOSE DO
    BEGIN
    GENERATE_KEY_CLICK;
    KEY_1 := GET_NEXT_KEY;
    END;
  CASE KEY_1 OF

    SPACE_GRAPHICAL_DIACRITICAL_SET:  (* have KEY_1, go get KEY_2 *)
      GENERATE_KEY_CLICK;

    DELETE:
      BEGIN
      GENERATE_KEY_CLICK;
      IN_COMPOSE_MODE := FALSE;
      END

    OTHERWISE:
      BEGIN
      IN_COMPOSE_MODE := FALSE;
      GENERATE_KEY_CLICK;
      TRANSMIT_CONTROL_FUNCTION (KEY_1);
      END
    END;          (* end case on first key of pair *)
```

```
  IF IN_COMPOSE_MODE THEN
    (* Process second key of explicit or implied compose pair *)
    BEGIN
    KEY_2 := GET_NEXT_KEY;
    CASE KEY_2 OF

      SPACE_GRAPHIC_DIACRITICAL_SET:
         IF VALID_COMPOSE_SEQUENCE (KEY_1, KEY_2) THEN
           BEGIN          (* valid pair *)
           GENERATE_KEY_CLICK;
           TRANSMIT_COMPOSED_CHAR (KEY_1, KEY_2);
           IN_COMPOSE_MODE := FALSE;
           END
         ELSE             (* invalid pair *)
           ERROR_PROCEDURE;

      DELETE:             (* Delete *)
         BEGIN
         GENERATE_KEY_CLICK;
         IN_COMPOSE_MODE := FALSE;
         END;

      COMPOSE:            (* Compose *)
        GENERATE_KEY_CLICK;

      OTHERWISE: (* not Compose, SPACE, graphic, diacrit. or DEL *)
         BEGIN
        IN_COMPOSE_MODE := FALSE;
         GENERATE_KEY_CLICK;
         TRANSMIT_CONTROL_FUNCTION (KEY_2);
         END
      END;        (* end case *)
    END; (* end case on second key of pair *)
  END;            (* end loop that Compose Key starts over *)
  TURN_OFF_LED;
END;              (* end procedure PROCESS_COMPOSE_SEQUENCE *)

PROCEDURE ERROR_PROCEDURE;
BEGIN
RING_ERROR_BELL;
IN_COMPOSE_MODE := FALSE;
END;
```

6.15.10 8-bit Characters Mode Valid Compose Sequences

6.15.10.1 Compose Sequences For Characters Without Diacritical Marks -

Note when the Compose key is used to initiate the sequence, the two following keys may be depressed in either order, unless the table entry explicitly says "in order".

CHARACTER	SEQUENCES			
#	+ +			
@	a a	A A	a A	
[	((			
\	/ /	/ <		
]	))			
{	(-			
|	/ ^			
}	) -			
<NBSP> No Break Space	SP SP			
<!!> ¡	! !			
<c/> Cent	c /	C /	c |	C |
<L=> Pound Sign	L =	l =	L -	l -
<Xo> General currency	X 0	X O	X o	
	x 0	x O	x o	
<Y=> Yen	Y -	y -	Y =	y =
<||> Broken vertical bar	| |	! ^		
<So> Section Sign	S !	S 0	S O	S o
	s !	s 0	s O	s o
<CO> Copyright	C O	C o	C 0	
	c O	c o	c 0	
<_a> Feminine ordinal	_ a	_ A		
<<<> Angle quotes left	< <			
<-,> Logical not sign	- , (in order)			
<SHY> Soft hyphen	- -			
<RO> Reg. Trademark	R O	R o	R 0	
	r O	r o	r 0	

<-^> Macron	_ ^	- ^
<+-> Plus-minus	+ -	
<2^> Superscript 2	2 ^	
<3^> Superscript 3	3 ^	
</u> Mu, Micro sign	/ u	/ U (in order)
<!P> Paragraph	! P	! p
<.^> Middle dot	. ^	
<1^> Superscript 1	1 ^	
<_o> Masculine ordinal	_ o	_ O
<>>> Angle quote right	> >	
<14> Quarter	1 4 (in order)	
<12> Half	1 2 (in order)	
<34> Three Quarters	3 4 (in order)	
<??> Inverted ?	? ?	
<AE> AE ligature	A E (in order)	
<-D> Cap Icelandic Eth	D -	
<OE> OE ligature	O E (in order)	
<xx> times sign	x x	XX xX
<O/> O slash	O /	
<TH> Icelandic Thorn	T H (in order)	
<ss> ß	s s	
<ae> ae ligature	a e (in order)	
<-d> Sm Icelandic Eth	d -	
<oe> oe ligature	o e (in order)	
<-:> division sign	- :	
<o/> o slash	o /	
<th> Sm Icelandic Thorn	t h (in order)	
<0^> °	0 ^	
	SP ° (space degree)	
	SP * (space asterisk)	
~ tilde character	SP ~ (space tilde-mark)	
` open quote	SP ` (space open-quote)	
	space grave-accent	
' apostrophe	SP ' (space apostrophe)	
	space acute-accent	
<''> acute accent	' ' (apostrophe apostrophe)	
	acute-accent acute-accent	
^ circumflex character	SP ^ (space circumflex-accent)	
" double quote	SP " (space double-quote)	
	space dieresis	
<"^> dieresis	" " (double-quote double-quote)	
	dieresis dieresis	
<,,> cedilla	, , (comma comma)	

6.15.10.2 Compose Sequences For Characters With Diacritical Marks For composition of the following composite characters, either the mark or the associated character(s) may be used in the compose sequence. The compose sequence consists of one of the following:

- Mark, Basic Character
- COMPOSE, Associated Character, Basic Character
- COMPOSE, Basic Character, Associated Character
- COMPOSE, Mark, Basic Character
- COMPOSE, Basic Character, Mark

Where the basic character is the alphabetic character without the accent or mark. For example the basic character in <U"> is U.

Mark	Associated Character	Composite Characters
Grave Accent	` single quote	<A`><a`><E`><e`><I`><i`><O`><o`> <U`><u`>
Acute Accent	' apostrophe	<A'><a'><E'><e'><I'><i'><O'><o'> <U'><u'><Y'><y'>
Circumflex Accent	^ circumflex	<A^><a^><E^><e^><I^><i^><O^><o^> <U^><u^>
Tilde Mark	~ tilde	<A~><a~><N~><n~><O~><o~>
Dieresis or Umlaut Mark	" quotations	<A"><a"><E"><e"><I"><i"><O"><o"> <U"><u"><Y"><y">
Ring Mark	* asterisk or degree	<A*><a*>
Cedilla	, comma	<C,><c,>

6.15.10.3 Non-Spacing Diacritical Marks By Keyboard -

See the position keyboard map for details. The following is a list of non-spacing characters which do an implied compose. The list is by keyboard. The most recent or recommended variation for each dialect is listed first. Note: This table applies to keyboard maps (as specified in Set-Up), not physical keyboards.

1. North American (LK201-EE US/UK or LK201-AA) - none

2. British (LK201-EE US/UK) - none

3. British (LK201-AE)
   E00 Tilde Mark
   C12 Circumflex Mark
   C12 Grave Accent (shifted)

4. Flemish (LK201-AB)
   E00 Grave Accent
   E00 Tilde Mark (shifted)
   D11 Circumflex Accent
   D11 Dieresis (shifted)

5. Canadian (French) (LK201-AC)
   E00 Tilde Mark
   C11 Grave Accent
   C11 Circumflex Accent (shifted)

6. Danish (2nd, LK201-ED) - none

7. Danish (1st, LK201-AD)
   E00 Tilde Mark
   E12 Acute Accent
   E12 Grave Accent (shifted)
   D12 Dieresis
   D12 Circumflex Accent (shifted)

8. Finnish (3rd, LK201-NX) - none

9. Finnish (1st, LK201-AF)
   E00 Tilde Mark
   E12 Circumflex Accent
   E12 Grave Accent (shifted)

10. Austrian/German (2nd, LK201-NG) - none

11. Austrian/German (1st, LK201-AG)
    E00 Tilde Mark
    E00 Circumflex Accent (shifted)
    E12 Acute Accent
    E12 Grave Accent (shifted)

12. Dutch (2nd, LK201-NH) - none

13. Dutch (1st, LK201-AH)
    E00 Tilde Mark
    D11 Circumflex Accent
    D11 Dieresis (shifted)
    C11 Grave Accent
    C11 Acute Accent (shifted)

14. Italian (LK201-AI)
    E00 Grave Accent
    E00 Tilde Mark (shifted)
    E08 Circumflex Accent

15. Swiss (French) (LK201-AK)
    E12 Circumflex Accent
    E12 Grave Accent (shifted)
    D12 Dieresis
    D12 Tilde Mark (shifted)

16. Swiss (German) (LK201-AL)
    E12 Circumflex Accent
    E12 Grave Accent (shifted)
    D12 Dieresis
    D12 Tilde Mark (shifted)

17. Swedish (2nd, LK201-NM) - none

18. Swedish (1st, LK201-AM)
    E00 Tilde Mark
    E12 Circumflex Accent
    E12 Grave Accent (shifted)

19. Norwegian (2nd, LK201-EN) - none

20. Norwegian (1st, LK201-AN)
    E00 Tilde Mark
    E12 Acute Accent
    E12 Grave Accent (shifted)
    D12 Dieresis
    D12 Circumflex Accent (shifted)

21. Belgian/French (LK201-AP)
    E00 Grave Accent
    E00 Tilde Mark (shifted)
    D11 Circumflex Accent
    D11 Dieresis (shifted)

22. Spanish (LK201-AS)
    D11 Grave Accent
    D11 Circumflex Accent (shifted)
    C11 Acute Accent
    C11 Dieresis (shifted)
    C12 Tilde Mark (shifted)

23. Portuguese (LK201-AV)
    C11 Tilde Mark
    C11 Circumflex Accent (shifted)
    D11 Acute Accent
    D11 Grave Accent (shifted)

6.15.10.4 Valid Compose Sequences And The Supplemental Character Sets -

Depending on the Supplemental character set chosen in Set-Up (either DEC Supplemental or ISO Latin-1 Supplemental), there are some changes in the compose sequences available to create 8-bit characters, as a direct result of the lack of some characters in either of the character sets.

The following table specifies which characters are not available in each of the supplemental character sets. Compose sequences which choose the characters listed cannot be composed when that supplemental character set is chosen through Set-Up:

DEC Supplemental (not available)

160 <NBSP> no break space
166 <||> broken vertical bar
168 <"^> dieresis
172 <-,> logical not
173 <SHY> soft (syllable) hyphen
174 <RO> registered trademark sign
175 <-^> macron
180 <''> acute accent
184 <,,> cedilla
190 <34> fraction three quarters
208 <-D> capital Icelandic Eth
215 <xx> times sign
221 <Y'> capital Y with acute accent
222 <TH> capital Icelandic thorn
240 <-d> small Icelandic Eth
247 <-:> division sign
253 <y'> small y with acute accent
254 <th> small Icelandic thorn

ISO Latin-1 Supplemental (not available)

215 <OE> capital OE ligature
221 <Y"> capital Y with dieresis
247 <oe> small oe ligature

There are also two characters in different positions between the ISO Latin-1 Supplemental and DEC Supplemental character sets. For these two character sets, the appropriate compose sequence is present when either set is invoked. Only the code transmitted to the host changes:

Character Name	DEC Supplemental Coding	ISO Latin-1 Supplemental Coding
<OX> general currency	168	164
<y"> small y w/dieresis	253	255

Deviation Note

On the VT320, VT330, and VT340, there is one set of compose sequences whose result changes depending on whether the ISO Latin-1 Supplemental or DEC Supplemental character set is chosen in Set-Up:

Compose Sequences	DEC Supplemental Result	ISO Latin-1 Result
<dieresis> <space> <space> <dieresis>	34 - " quotation marks	168 - <"^> dieresis

This difference has been deprecated. "<dieresis> <space>" should produce double quote in both DEC Supplemental and ISO Latin-1.

6.15.11 7-bit Characters Mode Valid Compose Sequences

6.15.11.1 Typewriter Keys Valid Compose Sequences By Keyboard (7-bit Characters) -

The following details all of the valid COMPOSE sequences for each keyboard when the terminal is in 7-bit Characters mode, and Typewriter Keys are selected. A separate selection of compose sequences is given for Data Processing Keys, regardless of Keyboard type, because when Data Processing keys is selected in 7-bit Characters mode, the ASCII character set, and not the associated NRCS, is used in G0. No COMPOSE sequences other than those listed for any one keyboard are legal when that keyboard is selected and is in 7-bit Characters mode. Note: This table applies to keyboard maps as specified in Set-Up, not physical keyboards.

1. North American (LK201-EE US/UK)

   7-bit Characters mode is not available with the North American keyboard.

2. British (LK201-EE US/UK) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
<L=> UK pound	- L L - - l l - = L L = = l l =
' apostrophe	space apostrophe apostrophe space
@ commercial at	a a A A A a a A
[opening bracket	((
\ backslash	/ / / < < /
] closing bracket	))
^ circumflex character	space ^ (circumflex) ^ (circumflex) space
` grave accent	space ` (grave accent) ` (grave accent) space
{ opening brace	(- - (
| vertical bar	/ ^ ^ /
} closing brace	) - -)
~ tilde	space ~ (tilde) ~ (tilde) space

3. Flemish (LK201-AB) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
<L=> UK pound	- L L - - l l - = L L = = l l =
' apostrophe	space apostrophe apostrophe space
<a`> small a with grave	` (grave accent) a a ` (grave accent)
<0^> °	0 ^ ^ 0 space degree degree space space * * space
<c,> small c with cedilla	c , , c
<So> §	! s s ! ! S S ! o s s o o S S o O s s O O S S O 0 s s 0 0 S S 0
` grave accent	space ` (grave accent) ` (grave accent) space
<e'> small e with acute	apostrophe e e apostrophe
<u`> small u with grave	` (grave accent) u u ` (grave accent)
<e`> small e with grave	` (grave accent) e e ` (grave accent)
^ circumflex	space ^ ^ space

4. French (Canada) (LK201-AC) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
# pound (number) sign	+ +
' apostrophe	space apostrophe apostrophe space
<a`> small a with grave	grave a a grave
<a^> small a with circumflex	circumflex-accent a a circumflex-accent
<c,> small c with cedilla	c , , c
<e^> small e with circumflex	circumflex-accent e e circumflex-accent
<i^> small i with circumflex	circumflex-accent i i circumflex-accent
<o^> small o with circumflex	circumflex-accent o o circumflex-accent
<e'> small e with acute	apostrophe e e apostrophe
<u`> small u with grave	grave u u grave
<e`> small e with grave	grave e e grave
<u^> small u with circumflex	circumflex-accent u u circumflex-accent

5. Danish (2nd, LK201-ED) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
# pound (number) sign	+ +
' apostrophe	space apostrophe apostrophe space
@ commercial at	a a A A A a a A
<AE> capital Æ	A E
<O/> capital O with slash	O / / O
<A*> capital A with ring	A asterisk asterisk A
^ circumflex	space ^ ^ space
` grave accent	space ` (grave accent) ` (grave accent) space
<ae> small æ	a e
<o/> small o with slash	o / / o
<a*> small a with ring	a asterisk asterisk a
~ tilde	space ~ (tilde) ~ (tilde) space

6. Finnish (3rd, LK201-NX) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
# pound (number) sign	+ +
' apostrophe	space apostrophe apostrophe space
@ commercial at	a a A A A a a A
<A"> capital A with umlaut	quotations A A quotations
<O"> capital O with umlaut	quotations O O quotations
<A*> capital A with ring	A asterisk asterisk A A <0^> <0^> A
<U"> capital U with umlaut	quotations U U quotations
<e'> small e with acute	apostrophe e e apostrophe
<a"> small a with umlaut	quotations a a quotations
<o"> small o with umlaut	quotations o o quotations
<a*> small a with ring	a asterisk asterisk a a <0^> <0^> a
<u"> small u with umlaut	quotations u u quotations

7. Austrian/German (2nd, LK201-NG) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
# pound (number) sign	+ +
' apostrophe	space apostrophe apostrophe space
<So> §	! s s ! ! S S ! o s s o o S S o O s s O O S S O 0 s s 0 0 S S 0
<A"> capital A with umlaut	quotations A A quotations
<O"> capital O with umlaut	quotations O O quotations
<U"> capital U with umlaut	quotations U U quotations
^ circumflex character	space ^ (circumflex) ^ (circumflex) space
` grave accent	space ` (grave accent) ` (grave accent) space
<a"> small a with umlaut	quotations a a quotations
<o"> small o with umlaut	quotations o o quotations
<u"> small u with umlaut	quotations u u quotations
<ss> ß	s s

8. Dutch (2nd, LK201-NH) - Typewriter Keys

7-bit Characters mode is not available with the Dutch Keyboard (same as North American).

9. Italian - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
<L=> UK pound	- L L - - l l - = L L = = l l =
' apostrophe	space apostrophe apostrophe space
<So> §	! s s ! ! S S ! o s s o o S S o O s s O O S S O 0 s s 0 0 S S 0
<0^> °	0 ^ ^ 0 space * * space
<c,> small c with cedilla	c , , c
<e'> small e with acute	apostrophe e e apostrophe
^ circumflex character	space ^ (circumflex) ^ (circumflex) space
<u`> small u with grave	grave-accent u u grave-accent
<a`> small a with grave	grave-accent a a grave-accent
<o`> small o with grave	grave-accent o o grave-accent
<e`> small e with grave	grave-accent e e grave-accent
<i`> small i with grave	grave-accent i i grave-accent

10. Swiss (French) (LK201-AK) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
<u`> small u with grave	grave-accent u u grave-accent
' apostrophe	space apostrophe apostrophe space
<a`> small a with grave	grave-accent a a grave-accent
<e'> small e with acute	apostrophe e e apostrophe
<c,> small c with cedilla	c , , c
<e^> small e with circumflex	circumflex-accent e e circumflex-accent
<i^> small i with circumflex	circumflex-accent i i circumflex-accent
<e`> small e with grave	grave-accent e e grave-accent
<o^> small o with circumflex	circumflex-accent o o circumflex-accent
<a"> small a with umlaut	umlaut-mark a a umlaut-mark
<o"> small o with umlaut	umlaut-mark o o umlaut-mark
<u"> small u with umlaut	umlaut-mark u u umlaut-mark
<u^> small u with circumflex	circumflex-accent u u circumflex-accent

11. Swiss (German) (LK201-AL) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
<u`> small u with grave	grave-accent u u grave-accent
' apostrophe	space apostrophe apostrophe space
<a`> small a with grave	grave-accent a a grave-accent
<e'> small e with acute	apostrophe e e apostrophe
<c,> small c with cedilla	c , , c
<e^> small e with circumflex	circumflex-accent e e circumflex-accent
<i^> small i with circumflex	circumflex-accent i i circumflex-accent
<e`> small e with grave	grave-accent e e grave-accent
<o^> small o with circumflex	circumflex-accent o o circumflex-accent
<a"> small a with umlaut	umlaut-mark a a umlaut-mark
<o"> small o with umlaut	umlaut-mark o o umlaut-mark
<u"> small u with umlaut	umlaut-mark u u umlaut-mark
<u^> small u with circumflex	circumflex-accent u u circumflex-accent

12. Swedish (2nd, LK201-NM) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
# pound (number) sign	+ +
' apostrophe	space apostrophe apostrophe space
<E'> capital E with acute	apostrophe E E apostrophe
<A"> capital A with umlaut	quotations A A quotations
<O"> capital O with umlaut	quotations O O quotations
<A*> capital A with ring	A asterisk asterisk A A <0^> <0^> A
<U"> capital U with umlaut	quotations U U quotations
<e'> small e with acute	apostrophe e e apostrophe
<a"> small a with umlaut	quotations a a quotations
<o"> small o with umlaut	quotations o o quotations
<a*> small a with ring	a asterisk asterisk a a <0^> <0^> a
<u"> small u with umlaut	quotations u u quotations

13. Norwegian - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
# pound (number) sign	+ +
' apostrophe	space apostrophe apostrophe space
@ commercial at	a a A A A a a A
<AE> capital Æ	A E
<O/> capital O with slash	O / / O
<A*> capital A with ring	A asterisk asterisk A
^ circumflex	space ^ ^ space
` grave accent	space ` (grave accent) ` (grave accent) space
<ae> small æ	a e
<o/> small o with slash	o / / o
<a*> small a with ring	a asterisk asterisk a
~ tilde	space ~ (tilde) ~ (tilde) space

14. French (Belgium) (LK201-AP) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
<L=> UK pound	- L L - - l l - = L L = = l l =
' apostrophe	space apostrophe apostrophe space
<a'> small a with grave	' (grave accent) a a ' (grave accent)
<0^> °	0 ^ ^ 0 space degree degree space space * * space
<c,> small c with cedilla	c , , c
<So> §	! s s ! ! S S ! o s s o o S S o O s s O O S S O 0 s s 0 0 S S 0
` grave accent	space ` (grave accent) ` (grave accent) space
<e'> small e with acute	apostrophe e e apostrophe
<u`> small u with grave	` (grave accent) u u ` (grave accent)
<e`> small e with grave	` (grave accent) e e ` (grave accent)
^ circumflex	space ^ ^ space

15. Spanish (LK201-AS) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
<L=> UK pound	- L L - - 1 1 - = L L = = 1 1 =
' apostrophe	space apostrophe apostrophe space
<So> §	! s s ! ! S S ! o s s o o S S o O s s O O S S O 0 s s 0 0 S S 0
<!!> inverted !	! !
<N~> capital N with tilde	~ (tilde) N N ~ (tilde)
<??> inverted ?	? ?
^ circumflex character	space ^ (circumflex) ^ (circumflex) space
` grave accent	space ` (grave accent) ` (grave accent) space
<0^> °	0 ^ ^ 0 space * * space
<n~> small n with tilde	~ (tilde) n n ~ (tilde)
<c,> small c with cedilla ~ tilde	c , , c space ~ (tilde) ~ (tilde) space

16. Portuguese (LK201-AV) - Typewriter Keys

Character	Sequences
" quotations	space quotations quotations space
' apostrophe	space apostrophe apostrophe space
@ commercial at	a a A A A a a A
<A~> capital A with tilde	A ~ (tilde) ~ (tilde) A
<C,> capital C with cedilla	C , , C
<O~> capital O with tilde	O ~ (tilde) ~ (tilde) O
^ circumflex character	space ^ (circumflex) ^ (circumflex) space
` grave accent	space ` (grave accent) ` (grave accent) space
<a~> small a with tilde	a ~ (tilde) ~ (tilde) a
<c,> small c with cedilla	c , , c
<o~> small o with tilde	o ~ (tilde) ~ (tilde) o
~ tilde character	space ~ (tilde) ~ (tilde) space

6.15.11.2 Typewriter Keys Non-Spacing Diacritical Marks By Keyboard (7-bit Characters) -

The following lists all the non-spacing diacritical marks while the terminal is in 7-bit Characters mode, with Typewriter Keys selected. Note: This table applies to keyboard maps as specified in Set-Up, not physical keyboards.

1. North American (LK201-EE US/UK or LK201-AA) - not applicable.

2. British (LK201-EE US/UK) - none

3. British (LK201-AE) - none

4. Flemish (LK201-AB)
   E00 Grave Accent
   D11 Circumflex Accent
   D11 Dieresis (shifted)

5. Canadian (French) (LK201-AC)
   C11 Grave Accent
   C11 Circumflex Accent (shifted)

6. Danish (2nd, LK201-ED) - none

7. Danish (1st, LK201-AD)
   E00 Ring Mark (shifted)
   D12 Dieresis

8. Finnish (3rd, LK201-NX) - none

9. Finnish (1st, LK201-AF) - none

10. Austrian/German (2nd, LK201-NG) - none

11. Austrian/German (1st, LK201-AG) - none

12. Dutch (2nd, LK201-NH) - not applicable

13. Dutch (1st, LK201-AH) - none

14. Italian (LK201-AI)
    E00 Grave Accent
    E08 Circumflex Accent

15. Swiss (French) (LK201-AK)
    E12 Circumflex Accent
    E12 Grave Accent (shifted)
    D12 Dieresis

16. Swiss (German) (LK201-AL)
    E12 Circumflex Accent
    E12 Grave Accent (shifted)
    D12 Dieresis

17. Swedish (2nd, LK201-NM) - none

18. Swedish (1st, LK201-AM) - none

19. Norwegian (2nd, LK201-EN) - none

20. Norwegian (1st, LK201-AN)
    E00 Ring Mark (shifted)
    D12 Dieresis

21. Belgian/French (LK201-AP)
    E00 Grave Accent
    D11 Circumflex Accent
    D11 Dieresis (shifted)

22. Spanish (LK201-AS)
    D11 Grave Accent
    D11 Circumflex Accent (shifted)
    C11 Acute Accent
    C11 Dieresis (shifted)
    C12 Tilde Mark (shifted)

23. Portuguese (LK201-AV)
    C11 Tilde Mark
    C11 Circumflex Accent (shifted)
    D11 Acute Accent
    D11 Grave Accent (shifted)

6.15.11.3 Data Processing Keys Valid Compose Sequences -

The following compose sequences are valid when the terminal is in 7-bit Characters mode, with Data Processing keys selected, regardless of the keyboard selected.

When in 7-bit Characters Mode, with Data Processing keys selected, the ASCII character set is used, instead of the applicable National Replacement Character set. Therefore, compose sequences to generate the special characters and symbols in the several NRCS are not available when Data Processing Keys is selected.

All Keyboards - Data Processing Keys

Character	Sequences
" quotations	space quotations quotations space
# pound (number) sign	+ +
' apostrophe	space apostrophe apostrophe space
@ commercial at	a a A A A a a A
[opening bracket	((
\ backslash	/ / / < < /
] closing bracket	))
^ circumflex character	space ^ (circumflex) ^ (circumflex) space
` grave accent	space ` (grave accent) ` (grave accent) space
{ opening brace	(- - (
| vertical bar	/ ^ ^ /
} closing brace	) - -)
~ tilde	space ~ (tilde) ~ (tilde) space

6.15.11.4 Data Processing Keys, Non-Spacing Diacritical Marks -

There are no non-spacing diacritic keys in 7-bit Characters mode when Data Processing Keys are selected, because no such key is required to produce a character from the ASCII character set.

6.16 CONTROL CODES AND KEYSTROKES

The following codes will be transmitted when the control key is depressed simultaneously with another key:

Legend	HEX Code Transmitted	Name
Ctrl-space	00	NUL
Ctrl-2	00	NUL
Ctrl-A	01	SOH
Ctrl-B	02	STX
Ctrl-C	03	ETX
Ctrl-D	04	EOT
Ctrl-E	05	ENQ
Ctrl-F	06	ACK
Ctrl-G	07	BEL
Ctrl-H	08	BS
Ctrl-I	09	HT
Ctrl-J	0A	LF
Ctrl-K	0B	VT
Ctrl-L	0C	FF
Ctrl-M	0D	CR
Ctrl-N	0E	SO
Ctrl-O	0F	SI
Ctrl-P	10	DLE
Ctrl-Q	11	DC1
(Only if XOFF handling is disabled)		
Ctrl-R	12	DC2
Ctrl-S	13	DC3
(Only if XOFF handling is disabled)		
Ctrl-T	14	DC4
Ctrl-U	15	NAK
Ctrl-V	16	SYN
Ctrl-W	17	ETB
Ctrl-X	18	CAN
Ctrl-Y	19	EM
Ctrl-Z	1A	SUB
Ctrl-3	1B	ESC
Ctrl-[	1B	ESC
Ctrl-4	1C	FS
Ctrl-\	1C	FS
Ctrl-5	1D	GS
Ctrl-]	1D	GS
Ctrl-6	1E	RS
Ctrl-~	1E	RS
Ctrl-7	1F	US
Ctrl-?	1F	US
Ctrl-8	7F	DEL
Ctrl-Tab	09	HT
Ctrl-Enter	(same as ENTER alone)	
Ctrl-Return	(same as RETURN alone)	
Ctrl-Delete	18	CAN

Notes:

1. The Shift key is ignored when pressed in combination with Control.

2. On the number keys (E01-E10), the control-numeral combination takes precedence over the other legend (if the control-numeral combination is valid) regardless of the shift state.

3. On the North American keyboard, the control-tilde combination is used for the Open Quote/Tilde key (Tilde is the shifted legend on E00, Control Open_quote is not a valid control combination).

6.17 SUMMARY OF MODES

The following table summarizes the modes that affect the keyboard and/or character transmission. Those modes that do not have a symbol are local Set-Up modes only and are not setable from the host. Those modes that do not have an LED are indicated as blank.

Name of Mode	Symbol/LED	Reset Name	Set Name
Host Port Environment *		Seven_Bit	Eight_Bit
Keyboard Usage ***	DECKBUM****	Typewriter	Data Processing
Conformance Level	DECSCL	Level_1	Level_2 Level_3
Hold Screen **	Hold Screen	Hold_Off	Hold_On
Lock Key **	Lock	Unlocked	Locked
Caps/Shift Lock		Caps_Lock	Shift_Lock
Compose * and **	Compose	Compose_Off	Compose_On
Keyboard Action	KAM/Wait	Unlocked	Locked
Auto Repeat ***	DECARM	Repeat_Off	Repeat_On
Keyclick *		Keyclick_Off	Keyclick_On
Cursor Key	DECCKM	Cursor	Ck_Application
Keypad	DECKPAM/DECKPNM	Numeric	Kp_Application
	DECNKM	Numeric	Kp_Application
New Line	LNM	New_Line_Off	New_Line_On
C1 Transmission	S7C1T/S8C1T	Seven_Bit	Eight_Bit
Character Set Mode	DECNRCM	8-bit_Char's	7-bit_Characters
UPSS***	DECAUPSS	DEC-MCS	ISO Latin-1
Keyboard Dialect*			
Delete Key*		Delete	Backspace
Compose Key*		Enable	Disable
Break Key*		Enable	Disable

Notes:

* - Set-Up mode only (no coded representation)

** - Activated by terminal user by typing keys (no coded representation)

*** - Terminal user convenience feature - not to be issued by software except in response to the wishes of the terminal user

**** - Extension to Level 2, required in Level 3

6.18 CHANGE HISTORY

6.18.1 Revision 0.0 To 0.1

All sections:

1. Renumbered sections so as to put related information together. Organized material according to physical areas of the keyboard.

2. Added change bars to any change that could effect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

Section 3 - Terminology:

3. Added terminology section.

Section 4 - Physical Keyboard Description:

4. Clarified that Lock key is not effected by typing Shift key.

5. Clarified that Compose indicator also comes on for implied compose.

6. Clarified that XON and XOFF do NOT set Hold Screen Indicator, only Hold Screen Key does.

Section 5 - Conformance Requirements:

7. Made conformance requirement specific, including keys, control functions related to the keyboard and local Set-Up features.

8. Indicated that Caps lock Operation is the factory default mode (not Shift Lock Operation) in order to be compatible with the VT100.

9. Indicated that disabling the error bell is a Level 2 extension.

10. Require GR Transmission Mode for Level 2.

Section 7 - Functions and State for all keyboard areas:

11. Changed so that all keys that send codes also auto-repeat.

12. Indicated that keys in an implied or explicit compose sequence do not auto-repeat.

13. Indicated that all keys, including Compose and Lock keys, keyclick, except Shift and Control keys.

14. Indicated that keys that are inoperative when GR Transmission Mode is NO_GR, do not keyclick.

15. Corrected the coding for KAM parameter value.

16. Indicated that KAM disabled keyclick too.

17. SELECT CONFORMANCE LEVEL (DECSCL) unlocks the keyboard.

Section 8 - Cursor Keys:

18. Corrected coding for DECCKM.

19. Clarified that Cursor Keys depend only on Cursor Key Mode and are independent of Keypad Application/Numeric Mode.

Section 9 - Keypad:

20. Enter key always auto-repeats, independent of keypad mode.

Section 11 - Application Function Key Row:

21. Added internal documentation about Bill Hefner allocating application function keys between operating system and application programs to prevent conflict and increase compatibility.

22. Set-Up mode restores terminal modes and screen when exited.

23. Hold Screen does not update the screen (not just prevents scrolling).

Section 13 - Main Key Array:

24. Caps lock operation transmits upper case of lower case alphabetic even when it is on the same key with another symbol.

25. Indicated that Non-Spacing Diacritical keys behave the same after implied and explicit compose sequences have started.

26. Switched [with] on Canadian (French), German/Austrian, and Spanish data processing keyboards so that they are like all other keyboards that have them on the same key ([is in shifted position).

27. Removed Ctrl-[from Canadian (French) Data processing keyboard to generate ESC. Now like all other non-US keyboards that only use numbers with ctrl to get NUL, ESC, FS, GS, RS, US, and DEL.

28. Clarified that Compose indicator goes on when Non-Spacing Diacritical key pressed too.

29. Removed implementor option to transmit compose key strokes for invalid compose sequences.

30. When GR Transmission Mode is NO_GR, 7-bit explicit compose sequences are still required, but Non-Spacing Diacritical keys are immediately inoperative.

31. Indicated that a distinguishable bell sound on compose errors is an extension to Levels 1 and 2.

32. Indicated that the error bell can be separately disabled by the terminal user as an extension to Levels 1 and 2.

33. Clarified that error bell is echoed if a compose is ended by a function key, even though the function key is transmitted (and key clicked).

34. Compose key does keyclick. Lock, Shift, and Control don't key click.

35. Compose error doesn't keyclick on last key stroke.

36. Non-Spacing Diacritical keys keyclick, but do not auto-repeat.

6.18.2 Revision 0.1 To 0.2

All sections:

1. Removed Rev 0.1 change bars. Added change bars to any change that could effect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

2. In the coded format sections, removed most of the 7-bit notation since the only difference is CSI (9/11) in 8-bits versus ESC [(1/11 5/11) in 7-bits. This is not a specification change, only a presentation change.

3. In the coded format sections, added the 8-bit column/row decimal notation used in ANSI and ISO standards immediately below the graphic character notation, e.g., immediately below CSI 2 h is 9/11 3/2 6/8.

Section 3 - Terminology:

4. Added concept of "permanent label strip" and "removable label strip" of which the latter can be "system label strip" or an "application label strip".

5. Changed all references of "foreign keyboards" to "non-US keyboards".

Section 4 - Physical Keyboard Description:

6. Added concept of "product class" of keyboards for LK201-Ax (standard keyboard), LK201-Bx (DECmate II), LK201-Cx (VAXworkstation), LK201-Dx (Generic Video), LK201-Ex (Generic Hardcopy). The DEC Standard keyboard is intended to be used on most video terminals sold with DEC systems, except DECmate II and VAX Workstations. Generic is intended to be used on non-DEC systems or by OEMs that have their own software.

7. VAXworkstation has "Symbol" instead of "Compose" an implements it as a shift key, rather than a modal prefix key.

8. Added DECmate II and EDT keypad legends.

9. DECmate II Delete key has Word and Char instead so can delete previous character and previous word (holding Control Key down).

10. Added the Permanent Label Strips for each of the five above product classes and changed the Function row to always be blank (to agree with what we are shipping).

11. Added the Professional System Label Strip and RAINBOW 100 System Label Strip.

12. Changed the names of keyboard versions to be country with language in parentheses (only when needed). Removed all alternate country names. Art Len and Martin Hall agreed with this simplification.

13. Introduced the concept of "usage mode" to distinguish typewriter layout from data processing layout, rather than calling them different versions, since both modes are labeled on the same version of the keyboard.

14. Described keys not present on most non-U.S. typewriter modes (11 characters) and data processing modes (5).

15. Removed GR Transmission Mode (setable from host) and replaced it with the Host Port Environment Mode (which is only a Set-Up feature and which is required for Level 2 conformance but is optional for Level 1 conformance).

Section 5 - Conformance

16. Removed Requirement to implement Keyclick Mode as a Set-Up feature. It is an implementor option. Section 11 - Application Function Key row:

17. Corrected Level 1 coding for F12 and F13 keys (from 1/8 and 1/10 to 0/8 and 0/10) to agree with ASCII for BS and LF.

Section 13 - Main Key Array:

18. Put versions in increasing alphabetical order by version number.

19. When in Shift Lock Operation (as opposed to Caps Lock Operation), the Shift key rather than the Lock key is pressed again to leave the locked state.

20. Clarified that New Line Mode does not effect ctrl/M. ctrl/M always transmits just 0/13.

21. Corrected the coding of the Q keys (D01 and C01) to agree with ASCII (7/2 changed to 7/1 and 5/2 changed to 5/1).

22. Corrected Denmark LK201-xD in Data processing mode key C10 from O w/slash (15/8, 13/8) to Æ (14/6, 12/6).

23. Corrected Italy LK201-xI (both modes) to show that E08 circumflex is a Non-Spacing Diacritical key that starts an implied compose sequence.

24. Spain LK201-xS (typewriter mode) changed E00 upper from invert question mark (11/15) to invert exclamation mark (10/1).

25. Clarified that keyclick indicates whether something was transmitted to the host or not, even in compose errors.

26. Indicated additional compose error combinations in the table.

Section 14 - Summary of modes

27. Added this section for easy reference.

6.18.3 Revision 0.2 To AX10

1. Reorganized first five sections of the document.

2. Removed all references to the Dx and Ex classes of keyboard, and modified all legend tables to reflect this change.

3. Removed descriptions of Application Label Strips pending software strategy on label strip distribution control, and added a note to the System Label Strip table indicating that it is not required for conformance.

4. Added Japan (Katakana) Version LK201-xJ keyboard.

5. Corrected an error in the description of Keyboard Action Mode which referred to the Lock LED instead of the Wait LED.

6. Added notes to the description of Auto-Repeat to soften the conformance requirements, making it clear that variable repeat rates are not a required function, and that the delay in turning off auto-repeat when the DECARM control is received is UNDEFINED.

7. Added codes for Local Function Keys used in combination with Shift and Control, and for Compose Key when not implemented as a local function.

8. Added a deviation note to the description for New Line Mode to indicate that previous terminals caused New Line Mode to effect the operation of Control-M.

9. Corrected the code transmitted by Control-Q in all of the keyboard tables (it was 1/2, and should be 1/1).

10. Corrected the tables for the Denmark Version LK201-xD keyboard (both modes) which had errors on key ids C10 and C11.

11. Added a note that the Local Function Keys will terminate a Compose Sequence.

12. Added a note to the Compose Sequence tables to indicate that order is significant only where specifically indicated in the tables.

13. Removed the error bell from the Compose error recovery for keys which terminate the Compose sequence and transmit their normal codes, such as control keys and Function keys.

14. Removed the error bell from the definition of inoperative keys.

15. Removed the restriction on the use of Non-Spacing Diacritical Keys in Level 1 operation to allow valid 7-bit Compose sequences to be introduced by Non-Spacing Diacritical Keys.

16. Removed Hold Screen interactions with communications functions, particularly XON/XOFF, and added references to the section "Code Extension Layer".

17. Added a note that some control characters may be used as Local Function Keys when required by the communications protocol.

18. Changed the implementation of Caps Lock. Only keys with upper case characters on the legends will be effected by Caps Lock. Keys with one or more lower case diacritical characters will not be effected. All keyboard code tables were changed to reflect this.

19. Added a note that for Compose sequences consisting of two alphabetic characters, it is required to implement the sequences with both characters upper case and both lower case. It is optional to also implement the sequences with one character upper case and the other lower case.

20. Removed Local Function Key codes to the appendix "Documented Exceptions".

21. Changed the description of Lock Key to indicate that it works the same in both Shift Lock and Caps Lock Mode.

22. Added notes the Cursor Key Mode and Keypad Mode to indicate that any inter-dependencies are UNDEFINED in Level 1 operation, but they must be independent in Level 2 operation.

6.18.4 Revision AX10 To AX11

All sections:

1. Removed all revision AX10 change bars. Added change bars to any change that could effect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

2. Removed all references to echoing the error bell as part of the error recovery for dead keys.

Section 5 - Physical Keyboard Description

3. Removed note on VAXstation Keyboard Lock Indicator.

Section 6 - Control Functions And State

4. Added ESC to the list of keys which do not auto repeat.

5. Clarified rules defining which keys click. All keys which transmit codes, or cause the terminal to take some immediate action click.

Section 9 - Editing Keypad Keys

6. Removed statement that the codes transmitted by the Editing keypad keys are not effected by Control, Shift, or Lock. Added statement that these keys are inoperative when pressed in combination with Shift or Control.

Section 10 - Application Function Keys

7. Removed statement that the codes transmitted by the Application Function Keys are not effected by Control. Added statement that these keys are inoperative when pressed in combination with Control.

Section 12 - Main Key Array

8. Removed reference to keyclick behavior in section on special keys and functions.

9. Removed statement that the codes transmitted by Space Bar and Delete are independent of Control, Shift, and or Lock keys. Fixed table to show Control Delete sends 1/8.

10. Changed section on Control key operation to indicate that alternate forms for keying C0 controls are available on non-US keyboards whenever the keys involved in these combinations are present.

11. Added paragraph documenting behavior of auto repeat when Control is pressed.

12. Added statement that pressing Shift does not turn off lock in section on Lock Key Operation.

13. Added statement that delete key clicks when used to cancel a compose sequence.

14. Added paragraph documenting how British £ is handled in 7-bit mode on the British keyboard.

15. Corrected description of compose sequence error conditions and recovery. Dead keys are ignored while in a compose sequence. Illegal compose characters cancel the compose sequence but do not sound the error bell. Pressing Compose within a compose sequence clicks if keyclick is enabled.

16. Added new compose sequences for back slash, §, and °.

6.18.5 Revision AX11 To AX12

All sections:

1. Removed all revision AX11 change bars. Added change bars to any change that could effect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.

2. Removed all references to LK201-Cx

3. Replaced the term Dead Diacritical Mark with Non-Spacing Diacritical Mark to reflect current usage.

4. Re-organized keyboard features into categories by whether they affect coding or user interaction to group functionally related features together.

5. Updated entire standard to reflect latest country keyboards, 8-bit Architecture (ISO Latin-1), and NRCS support.

Section - 1 Introduction

6. Added subsection on keyboard product classes to explain model designations (LK201-xx).

7. Added explanation of Keyboard Versions and Dialects to show which keyboards are described in this chapter, and which are the most recent (recommended) and superseded keyboards for each country market (Keyboard Dialect).

Section - 2 Conformance Requirements

8. Added Level 3 conformance requirements.

9. Added description of Level 3 operation.

10. Expanded Keyboard Character Encoding section to cover "8-bit Architecture" and "NRCS Extension".

Section - 3 Referenced Standards and Related Publications

11. Updated referenced standards to include document order number and full name of standard.

12. Added reference to ^&LK201 Functional Specification\&.

13. Added information on how to order cited documents.

Section - 4 Terminology

14. Added definitions for "Keyboard Usage Mode" and "Keyboard Dialect".

Section - 5 Physical Keyboard Description

15. Documented front legends for Editing and Numeric Keypads for LK201- {NA, PA} (VT330/VT340 Local Edit Mode legends).

16. Described system legend strip variations for G99 as "Hold Session", G00 as "Local Print", and G02 as "Switch Session" for multiple session terminals.

17. Described system legend strip variations for G02 as "F4" or "Data/Talk".

Section - 6 Keyboard Operation and State that Affects User Interaction

18. Clarified section on Keyboard Output Silo. Added guideline "The VT320 provides a 25 keystroke silo".

19. Extended list of conditions that can reset Keyboard Action Mode to include Entering Set-Up in Level 1, Clear Comm, and Terminal Reset.

20. Added sentence that control and shift keys do not cause a new delay before auto-repeating new code (if any).

21. Broke out information on keys not effected by auto repeat mode, created a new section. Added additional keys that were not part of the original list.

22. Created section on auto repeat guidelines to show timing considerations for actual terminals.

23. Defined an upper limit on the time between when a DECARM control is received and the time a key which was already auto-repeating stops repeating when terminal is processing host output (to allow testing for conformance).

24. Clarified section on Visual Indicators.

25. Expanded section on Audible Indicators to include Warning Bell and Margin Bell. Added paragraph describing interaction of keyclick and bells.

Section - 7 Keyboard State and Operating Modes that Affect Keyboard Encoding

26. Added summary of modes that affect keyboard encoding at beginning of section. Added new modes: BackArrow Key mode, Disable Compose mode. Added new Set-Up options: Remap shifted ,. to <>, Remap <> to '~, Remap '~ to ESC, Remap KP, to SPACE.

27. Documented Terminal Emulation mode not previously covered (VT52 and VT100, VT200 and VT300).

28. Added description of "Character Set Mode" to cover NRCS Extension. Made NRCS support more intuitive by organizing as 7-bit/8-bit Characters instead of National/Multinational. Added algorithm for setting/resetting character set mode.

29. Greatly expanded explanation of Keyboard Usage Mode. Added summary of Data Processing characters for each keyboard dialect. Added description of new DECKBUM control function.

30. Greatly expanded explanation of Keyboard Dialect (selection of country keyboards) to cover NRCS Extension and latest country keyboard layouts. Added list showing NRC set to be used for each keyboard dialect.

31. Added detailed explanation of how Combined US/UK keyboard works with North American and British keyboard dialects.

32. Added description of User Preference Supplemental Set (UPSS) to cover 8-bit Architecture Extension.

Section - 9 Numeric Keypad Keys

33. Added description of DECNKM control function.

Section - 12 Local Function Keys

34. Expanded explanation of Hold Screen function to include "Hold Session" on multi-session terminals. Referenced new Terminal Synchronization chapter for explanation of Hold Screen interaction with flow control.

35. Included section on Local Function key F4 as used with the VT330/340 (Switch Session), and VT220 (Data/Talk).

36. Expanded explanation of Break Key operation to include Disable Break option.

Section - 13 Main Key Array - Special Keys and Functions

37. Clarified example of the ctrl-number construct.

38. Referenced new section on Control Codes and Keystrokes.

39. Added notes on control key operation with keys from the editing keypad and function keys.

40. Expanded explanation of Lock Key operation.

41. Added description of DEL/BS option for Backarrow key on VT300 series terminals.

42. Added description of "Hold Session" for multiple session terminals (VT330/VT340).

43. Expanded description of "Data/Talk" key as used for VT220 integral modem.

44. Noted F4 is dead if not used for a local function.

45. Added explanation of Disable Compose option.

Section - 14 Main Key Array - Graphic Character Keys

46. Clarified keyboard character set encoding.

47. Explained precedence rules for unshifted, caps-lock, shift-lock, shift, and control.

48. Removed most coding (where obvious) from tables and referenced DEC STD 169 on character sets.

49. Updated graphic character tables to include latest keyboards, and support for NRCS Extension.

50. Re-organized subsection to provide tables for 8-bit typewriter and Data Processing keys, as well as 7-bit typewriter and Data Processing keys for each Keyboard Dialect and country keyboard in one place.

51. Listed complete table only once per keyboard dialect, then listed differences to simplify understanding the affect of each keyboard mode.

52. Each table fits on one page and is clearly identified (Keyboard Dialect, 8-bit/7-bit Characters, Typewriter/Data_Processing, layout version within dialect, which physical keyboards table is intended for).

53. Added table for main key array for the LK201- {PA, NA} in the DECMate/WPS section.

Section - 15 Compose Operation

54. Expanded section on Compose Operation.

55. Provided tables for valid compose sequences in 8-bit and 7-bit Characters Mode (NRCS Extension) with and without non-spacing diacriticals.

56. Added compose sequences for 8-bit Architecture Extension (ISO Latin-1) as implemented in VT300 family.

57. Added section on choice of supplemental sets and compose operation.

58. Eliminated the one compose sequence that produces a different character in ISO Latin-1 versus DEC Supplemental. COMPOSE <dieresis> <space> should produce 'double quote' in both DEC Supplemental and ISO Latin-1 for backward compatibility and consistency with acute accent. Noted VT320, VT330, and VT340 as deviations.

59. Simplified table of compose sequences for 8-bit characters without non-spacing diacriticals by listing each sequence in only one order. Added note that order can be reversed unless table entry explicitly says "in order". Added phrase "in order" to every order sensitive entry.

60. Added VT330/VT340 numeric keypad (hexidecimal) compose feature.

61. Added "Switch Session" key to list of keys which abort compose immediately.

62. Added table summarizing non-spacing diacriticals for each keyboard dialect.

Section - 16 Control Codes and Keystrokes

63. Put control codes into one table in new section.

Section - 17 Summary of Modes

64. Updated summary of modes.

Section Index

VIDEO SYSTEMS REFERENCE MANUAL

Printer Port Extension

Document Identifier: A-DS-EL00070-07-0 Rev AX12, 28-Apr-1987

ABSTRACT: This section describes the characteristics of the Printer Port extension to the Level 2 and Level 3 Character Cell Display service class. It includes both the control functions required to cause transmission from the terminal to the printer, and the syntax of the coded interchange at the printer port. Note: The Level 1 Printer Port Extension consists of a sub-set of the functions described in this standard. At the present time there is no separate specification of the Level 1 required functions.

APPLICABILITY: Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria" (EL00070-01).

STATUS: FOR REVIEW ONLY

The material contained within this document is assumed to define mandatory standards unless it is clearly marked as: (a) not mandatory; or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements.

TITLE: VIDEO SYSTEMS - Printer Port Extension

DOCUMENT IDENTIFIER: A-DS-EL00070-07-0 Rev. AX12, 28-Apr-1987

REVISION HISTORY: Revision AX11 18-Mar-1985
Revision AX12 28-Apr-1987

FILES: User Documentation EL070S7.mem
Internal Documentation EL070S7.rno
EL070S7.rnt
EL070S7.rnx
ZZFALLBACK_DEC_CRM.rno
ZZFALLBACK_DEC_SPECIAL.rno
ZZFALLBACK_DEC_SUPPLEMENTAL.rno
ZZFALLBACK_DEC_TECHNICAL.rno
ZZFALLBACK_ISO_SUPPLEMENTAL.rno

Document Management Group: Terminal Interface Architecture
Responsible Department: Video Architecture Group
Responsible Individual: Peter Sichel
Authors: PAS

APPROVAL: This document prepared by the TBU Architecture Group has been reviewed and approved by the TIA Review Board for use throughout Digital.

******* STATUS: For Review Only ********

Peter Conklin, Technical Director,
Terminals Business Unit

******* STATUS: For Review Only ********

Eric Williams, Standards Process Manager

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, VIDEO::TERMARCH

Copies of this document can be ordered from Standards and Methods Control, LJO1/I2, DTN 226-2484, JOKUR::SMC

Please provide your name, badge number, cost center, mailstop, and ENET node.

CONTENTS

7.1 INTRODUCTION

The use of a Printer Port forms an extension to the Level 1, Level 2, and Level 3 Character Cell Display service classes. A conforming device may provide an auxiliary communications port which can be used to support a printing device. A means shall be provided to transfer data from the display to the printer port, as well as to relay data received at the host port directly to the printer port. The device must transfer data from the display, including the character codes, character renditions, and character sets, on a per-line, per-scrolling region, per-screen, and per-page basis.

The Printer Port uses serial asynchronous communications with independently setable communication parameters. Level 3 requires a fully bi-directional printer port to allow printer to host communication in addition to local printing (not mandatory for Level 1 or Level 2).

7.2 TERMINOLOGY

Auto Print Mode - A printer port control which, when set, causes the contents of the Active Line to be transmitted to the printer port whenever the Active Position is moved off the Active Line as a result of a Line Feed, Form Feed or Vertical Tab command, or autowrap.

Fallback Transmission - A substitute transmission used when a character cannot be transmitted in a particular mode; for instance, when in NATIONAL and LINE DRAWING mode and a character from the DEC Supplemental set is encountered.

Printer Controller Mode - A printer port control that causes all data received at the host port to be transferred directly to the printer port without interpretation by the display terminal.

Printer To Host Mode - A printer port control that causes all data received at the printer port (other than XON/XOFF flow control) to be relayed to the host.

Print Extent Mode - A printer port control used to select the area of the display to be transmitted to the printer port when a Print Page operation is performed.

Print Form Feed Mode - A printer port control which, when set, transmits a Form Feed (FF) control character to the printer at the conclusion of each Print Page or Print Screen operation.

Printer Port - An auxiliary communication port that supports a printing device.

Print Line - A printer control that causes the line containing the Active Position to be transmitted to the printer port and the print function to be executed.

Print Page - A printer control that causes the page containing the Active Position to be transmitted to the printer port and the print function to be executed. The extent of the area which is to be transmitted is determined by the setting of Print Extent Mode. "Print Page" was previously called "Print Screen" in Level 1 and Level 2. Page refers to the logical display as seen by the host, and may be larger or smaller than the physical screen.

Print Screen - A printer control that causes the visible screen image to be transmitted to the printer port and the print function to be executed.

Virtual Terminal - Consists of all the context (state information) of a real terminal but is able to share physical resources (such as the keyboard, display, and comm lines) with other virtual terminals. This allows a single physical device to appear as more than one terminal to a host computer.

Session - A connection between a single virtual terminal and a host computer.

7.3 STATE DESCRIPTIONS

7.3.1 Printer Port Flow Control

The Printer Port uses two flow control mechanisms:

1. XON/XOFF
2. Data Set Ready (DSR)

7.3.1.1 XON/XOFF

When XON/XOFF flow control is enabled, XON and XOFF are interpreted as flow control characters and are not buffered in the printer port input buffer or transmitted to the host.

The terminal recognizes XOFF (1/3) as a request from the printer device to suspend the transmission of characters. When this character is received by the terminal, transmission of all characters (other than XON or XOFF) to the printer is suspended. When the printer port is in this state, it is said to be in the XOFF state.

The terminal recognizes XON (1/1) as a request from the printer device to resume the transmission of characters, if any. When

this character is received by the terminal, the XOFF state of the printer is cleared and transmission of characters in the printer port output buffer is resumed. If the line was not XOFF-ed, this character is ignored.

When Printer To Host Communication is enabled (bidirectional printer port only), the terminal transmits an XOFF to the printer device if a character other than XON or XOFF is received from that device, and (1) the number of characters in the input buffer reaches the first XOFF point for the first time since the last XON was sent; (2) the number of characters in the input buffer reaches the second XOFF point for the first time since the last XON was sent; or (3) the input buffer is full. If the input buffer is full, the received character is discarded.

The terminal transmits an XON to the printer device when the printer port input buffer empties to the XON point, and the last flow control sent was XOFF.

When XON/XOFF flow control is disabled at both the printer port and the host port, characters XON and XOFF may be sent to the printer in Printer Controller mode, and are buffered and transmitted to the host when Printer To Host communication is enabled.

```
PRINTER_PORT_FLOW_CONTROL: (XOFF, NO_XOFF);
```

7.3.1.2 Data Set Ready (DSR)

The terminal polls DSR on the printer port before transmitting each character. If DSR is not being asserted by the printer device, transmission of the character is suspended until DSR is asserted by the printer device. DSR takes precedence over the XON/XOFF state.

All print commands will be ignored unless the terminal has seen Data Set Ready (DSR) since the last power-up, or Reset to Initial State (RIS). If at any time the printer sets DSR low (for example to indicate paper out), printing is suspended. When DSR is again set high, printing can resume.

7.3.1.3 Data Terminal Ready (DTR)

If DTR is not being used for hardware flow control, the terminal asserts DTR on the printer port whenever the terminal is turned on and not in self-test.

Conforming Devices may support the use DTR for hardware flow control. The following description is intended as a guideline.

When hardware flow control is selected, DTR is asserted any time the terminal is not in self-test, and the terminal's printer port input buffer is below the first XOFF point. DTR is de-asserted when the printer port input buffer is filled up to or beyond the first XOFF point. If the input buffer is full, received characters are discarded until there is again room in the input buffer.

When hardware flow control is used, only a few characters of buffer space are required to prevent loss of data. The first XOFF point is suggested as the flow control threshold because it is often adjustable allowing the user to tailor the buffer size depending the interaction requirements.

7.3.2 Printer Port Status

The printer port can be in one of three states depending on whether the terminal believes there is a printer available, and whether the printer is ready to accept data. The printer port status may be determined from the host using a Device Status Report control sequence.

If DSR (Data Set Ready) has not been asserted on the printer port since the last power-up or Reset to Initial State (RIS), the terminal assumes there is no printer. The device status report (response from terminal to host) will indicate "Printer not available". In this state, all print commands which require the printer port are ignored.

If the printer port has seen DSR since power-up, but it is no longer asserted on the printer port, the device status report will indicate "Printer not ready". In this state, print operations shall be queued. Updates that would affect the data to be printed are suspended until the data has been transferred to the printer port. If the terminal's host port input buffer fills, XOFF will be sent to the host (if enabled).

If the printer port sees DSR, the device status report will indicate "Printer ready". In this state, print operations can proceed immediately. Note updates that would affect the data to be printed are still suspended until the data has been transferred to the printer port. The printer port may provide an output buffer to minimize interruption of display updates.

```
PRINTER_STATUS:
    (PRINTER_READY, PRINTER_NOT_READY, PRINTER_NOT_AVAILABLE);
```

7.3.3 Printer Port Status On Virtual Terminals

When Virtual Terminals are supported (Session Management Extension), the terminal will report two additional printer port states to indicate the printer port is in use by another session.

If the Data Set Ready signal at the printer port is asserted, but an alternate session is currently sending information to the printer, the device status report will indicate "Printer Busy." This is functionally equivalent to the "Printer Not Ready" condition above.

If the printer port is explicitly assigned to another session so that the current session may not access the printer port, the device status report will indicate "Printer Not Assigned." This is functionally equivalent to the "Printer Not Available" condition above.

7.3.4 Printer Port Communications Environment

The printer port can be set for either an 8-bit or 7-bit communications line environment independently of the Host Port. In the 7-bit environment, all data will be transmitted as 7-bit characters. In the 8-bit environment, each character will be transmitted as an 8-bit character. If the host port is set to 8-bits and the printer port is set to 7-bits, C1 control characters will be converted to ESC Fe sequences at the printer port, and GR characters will have their eighth bit stripped.

```
PRINTER_PORT_ENVIRONMENT: (EIGHT_BIT, SEVEN_BIT);
```

7.3.5 Printer To Host Communications

Printer To Host Mode relays all input from the printer port other than XON/XOFF flow control back to the host (enabling session). Printer To Host mode may be selected from the host using the Media Copy (MC) control function.

Character input from the printer port is interleaved with input from the keyboard as it is received. Arbitration between keyboard and auxiliary device input is the responsibility of the application program and user. The terminal only provides arbitration to the keystroke, control function, or character level.

Characters associated with a single keystroke, or control function are sent as contiguous bytes. When the terminal sees an ESC, CSI, DCS, OSC, PM, or APC character, the escape sequence, control sequence, or control string, will be buffered until a terminator is received for the sequence, or until the input buffer is full.

Notes:

1. Printer To Host Mode functions in Printer Controller Mode as well as other print modes.

2. When available, Printer To Host Mode works at all operating levels (level 1, level 2, and level 3). 8-bit graphic characters may be relayed from the printer port even when the conformance level is set to Level 1.

3. If the host port is set to 7-bits, C1 control characters will be converted to ESC Fe sequences at the host port before re-transmission to the host, and GR characters will have their eighth bit stripped.

```
PRINTER_TO_HOST_MODE: (PRINTER_TO_HOST, NO_PRINTER_TO_HOST);
```

7.3.6 Printer Controller Mode

When Printer Controller Mode is set, all data received at the host port is transferred directly to the printer port without interpretation by the display terminal. The only exceptions to this rule are the communications control characters XON and XOFF (DC1 and DC3), NUL, and the control sequences which cause the terminal to enter and exit this mode.

XON and XOFF are used by the communications software for flow control on the line between the terminal and the host, and independently for flow control on the line between the terminal and the printer. The terminal must intercept the Media Copy control sequences from the host which turn Printer Controller Mode on and off, and must execute the sequence which turns Printer Controller Mode off. These control sequences are not passed to the printer.

Characters from the keyboard continue to be sent to the host. The video display is not updated. When in Printer Controller Mode no local printer functions will work. For example: Print Page does not work.

In going from an 8-bit host port to a 7-bit printer port, C1 control characters will be translated to ESC Fe sequences, and GR characters will have the eighth bit stripped (i.e., no fallback translation will be performed for 8-bit graphic characters in Printer Controller Mode).

Deviation Note

On VT220, VT240, and VT241, when the host port was configured for a 7-bit environment, and the

printer port was configured for 8-bits, the terminal would translate 7-bit C1 controls to their 8-bit equivalents. New terminals shall NOT perform this translation.

Printer Controller mode can be selected from the host using the Media Copy (MC) control function.

When Control Representation Mode (CRM) is selected, Printer Controller Mode is temporarily disabled.

Upon exiting from Printer Controller mode, the state of Auto Print mode is UNDEFINED. That is, when Printer Controller mode is exited, Auto Print Mode may not be in the same state it was when Printer Controller Mode was entered.

```
PRINTER_CONTROLLER_MODE: (PRINTER_CONTROLLER_OFF,
                          PRINTER_CONTROLLER_ON);
```

Guideline

The VT220 sets Auto Print mode off (normal print mode) upon exit from Printer Controller mode.

7.3.6.1 Printer Controller Mode On Virtual Terminals (guideline)

If the Printer is "assigned" to another session (virtual terminal), attempts to invoke Printer Controller Mode will be ignored. This is equivalent to the "Printer not available" state, but the Device Status Report will indicate "printer not assigned".

When the printer assignment is "shared" and the printer is in use by another session, this is equivalent to the "printer busy state". The next action which would cause data to be transferred to the printer will cause this session to wait until the printer is available. If the input buffer for this session fills while waiting, the terminal will send XOFF (or other flow control) to the host. Entering Printer Controller Mode forces the printer to appear busy to other sessions.

7.3.7 Auto Print Mode

When Auto Print Mode is set, any Line Feed (LF), Vertical Tab (VT), Form Feed (FF) control, or the execution of Auto Wrap if this mode is enabled, will cause the active line to be transferred from the display to the printer port. The line will be terminated by a Carriage Return (CR), followed by the same character that initiated the print operation (LF, VT, FF, or LF if Auto Wrap).

Auto Print mode can be set or cleared by the host using the Media Copy (MC) control function.

```
AUTO_PRINT_MODE: (AUTO_PRINT_OFF,AUTO_PRINT_ON);
```

7.3.7.1 Auto Print Mode On Virtual Terminals (guideline)

If the Printer is "assigned" to another session, attempts to set Auto Print mode will be ignored. If the printer assignment is "shared", and the printer is in use by another session (busy), the next action which would cause data to be transferred to the printer will cause this session to wait until the printer is available. Entering Auto Print Mode forces the printer to appear busy to other sessions.

Although Auto Print Mode can keep the printer "busy" indefinitely, it is not "assigned" to the session which invokes Auto Print Mode. This is to avoid a race condition in which the host could not predict whether a print operation will be queued or ignored.

7.3.8 Print Form Feed Mode

Print Form Feed Mode is used to determine whether a Form Feed (FF) control character shall be transmitted to the printer at the conclusion of each Print Page or Print Screen operation. In the set state (Print_FF) a Form Feed shall be sent. In the reset state (No_Print_FF) no Form Feed is sent.

```
PRINT_FORM_FEED_MODE: (PRINT_FF_OFF,PRINT_FF_ON);
```

7.3.9 Print Extent Mode

Print Extent Mode is used to determine the area of the page to be transmitted to the printer when a Print Page operation is performed. In the set state (Complete_Page) the entire contents of the current page is transmitted. In the reset state (Scrolling_Region) only the area of the page between the Top Margin and Bottom Margin of the scrolling region (inclusive of the margins) is transmitted.

```
PRINT_EXTENT_MODE: (PRINT_SCROLLING_REGION,PRINT_DISPLAY);
```

7.3.10 Local Controller Mode (not Mandatory)

When both Local Mode and Controller Mode are selected, the terminal is in "Local Controller Mode" (Local Mode is only selectable by the user under local control). In Local Controller Mode, keyboard characters are sent directly to the printer port.

Note that in regular Local mode the keyboard is aimed at the screen, while in regular Controller mode the keyboard is aimed at the host. When "Local Controller Mode" is exited to Printer Controller Mode, the characters accumulated in the receive silo are transmitted to the printer. When "Local Controller Mode" is exited to Local Mode, the terminal just redirects the keyboard from the printer port to the screen (and parser). Local echo is disabled while in Local Controller Mode. CRM mode is disabled when in local controller mode.

Local Controller mode is used to allow keyboard output to a printer. In local controller mode, the setting of printer to host mode is ignored and characters received from the printer are ignored (except XON and XOFF if XOFF support is enabled).

7.3.11 Printer Style

Three Printer Style modes are available to match the range of character sets the printer is capable of supporting. These are: National Only; National + Line Drawing; and All Characters. These modes are described in the next section on TRANSMITTING PRINT DATA.

```
PRINTER_STYLE:
    (NATIONAL_ONLY, NATIONAL_PLUS_LINE_DRAWING, ALL_CHARACTERS);
```

7.4 TRANSMITTING PRINT DATA

Each time a print of screen data is requested, three things happen. First, the printer is "initialized" to a known or assumed character set state, either via assumption, or via designating escape sequences and/or locking shift sequences. Next, the characters are transmitted. This may require additional designating escape sequences and/or shifts. Finally, the transmission is ended by putting the printer into a known state.

Since the host may access the printer during auto-print mode, the complete initialization / transmission / closing sequence is used for each line printed.

7.4.0.1 Print National Only (7-bit Or 8-bit Printer Port Environment)

ASSUMPTIONS: This mode is included for backward support of "dumb" printers which do not understand designating escape sequences, or which do not support any character set other than a National Replacement Character Set (NRCS). The terminal and printer have already been configured to use the same NRC Set. The terminal

determines which NRC Set to use based on the keyboard dialect (ASCII is the NRC Set for North America). Refer to the National Replacement Character Set section below.

INITIALIZATION: No escape sequences or shifts are sent. It is assumed that the printer is ready to print NRCS characters as GL.

TRANSMISSION: The characters are sent. No escape sequences or shifts are sent. If a character which is not in the selected NRCS is encountered, fallback is used. If 8-bits is set, the most significant bit is always zero. Characters with the "invisible" character attribute are transmitted as spaces. All other Display Attributes are ignored.

CLOSING: No closing sequences are sent.

FALLBACK: Characters which have an NRCS equivalent (have an identical dot matrix representation to an NRCS character) are sent as the NRCS equivalent. Otherwise fallback is used (a single unsupported character maps to one or more supported characters). Refer to the fallback section and tables later in this chapter.

7.4.0.2 Print National + Line Drawing (7-bit Or 8-bit Printer Port Environment)

ASSUMPTIONS: The printer supports ASCII, the VT100 line drawing set (DEC Special Graphics), the current NRCS (when in national mode), designating escape sequences for G0 and G1, and LS0/LS1 (SO/SI). It does not support 8-bit controls. The terminal determines which NRCS to use based on the keyboard dialect (ASCII is the NRCS for North America). Refer to the National Replacement Character Set section below.

INITIALIZATION: The printer is initialized to be sure the desired NRCS is designated as G0, and DEC Special Graphics (line drawing) is designated as G1. G0 is invoked. The sequences sent are:

```
ESC ( B     (Designate ASCII to G0, note other NRC Sets
             require a different designating sequence)
ESC ) 0     (Designate DEC Special Graphics to G1)
LS0         (Invoke G0 to GL)
```

TRANSMISSION: The characters are sent, using LS0 and LS1 to choose between NRCS and DEC Special Graphics. If a character is in both sets (e.g., uppercase letters), it is sent as NRCS. If a character is in ASCII but not in the current NRCS, ASCII is designated as G1 and invoked as needed. The next line drawing character encountered will cause the DEC Special Graphics set to again be designated as G1. If a character which is not in the current NRCS, ASCII, or DEC Special Graphics set is encountered, fallback is used. In an 8-bit environment the most significant

bit is always set to zero.

In this mode the display attributes are also sent. Refer to the Display Attributes section below.

CLOSING: LS0 is sent, if necessary. If ASCII was the last set designated as G1, DEC Special Graphics (line drawing) is re-designated. This leaves the printer with NRCS as G0, DEC Special Graphics as G1, and G0 invoked to GL.

FALLBACK: If the character has an equivalent (the same dot matrix representation) as a character in either the NRCS, DEC Special Graphics, or ASCII sets, then the equivalent is used. Otherwise fallback is used (a single unsupported character maps to one or more supported characters). Refer to the fallback section and tables later in this chapter.

7.4.0.3 Print All Characters (7-bit Printer Port Environment)

ASSUMPTIONS: The printer supports all the character sets in use by the terminal, designating escape sequences for G0, G1, G2, G3, and character set shifts LS0, LS1, SS2 and SS3. If the User Preference Supplemental set (UPSS) extension is used, the printer has already been configured with the same UPSS. If a DRCS is used, the host has already down line loaded the printer's character set(s) to match the video terminal's (in Printer Controller Mode, if necessary). Printers with changeable font cartridges must support all of the above character sets simultaneously. No font selection commands will be used.

INITIALIZATION: The printer is initialized to correspond with the terminal. The character sets currently designated as G0, G1, G2, and G3 in the terminal are designated as G0, G1, G2, and G3 in the printer respectively. G0 is invoked to GL. The sequences sent are:

```
ESC ( Dscs      (Designate a 94 character set as G0)
ESC ) Dscs      (Designate a 94 character set as G1)
ESC * Dscs      (Designate a 94 character set as G2)
ESC + Dscs      (Designate a 94 character set as G3)
LS0             (Invoke G0 to GL)
```

Where "Dscs" consists of zero, one, or two intermediate characters and a final character for the character set being designated.

If a 96 character set is to be designated, the first intermediate character will change as follows:

```
ESC - Dscs      (Designate a 96 character set as G1)
ESC . Dscs      (Designate a 96 character set as G2)
```

ESC / Dscs (Designate a 96 character set as G3)

If one of the character sets is the UPSS, it is designated as the UPSS, not the specific character set chosen.

TRANSMISSION: The characters are sent, using LS0 and LS1 to choose between the character sets in G0 and G1. If a character is in the set designated as G2, SS2 is used to temporarily invoke G2 into GL. If a character is in the set designated as G3, SS3 is used to temporarily invoke G3 into GL. If a character from an alternate character set is encountered the alternate set is designated as G3 and SS3 is used to send it.

The priority order for sending characters contained in more than one set is ASCII, UPSS, DEC Supplemental, ISO Supplemental, DEC Technical, and DEC Special Graphics.

In this mode the display attributes are also sent. Refer to the Display Attributes section below.

CLOSING: LS0 is sent, if necessary. If an alternate set was the last set designated as G3, G3 is re-designated. This leaves the printer with G0 invoked as GL; G0, G1, G2, and G3 the same as the terminal.

FALLBACK: When Print All Characters is selected, fallback is only provided for CRM characters. CRM characters do not belong to a character set. Refer to the fallback section and tables later in this chapter.

7.4.0.4 Print All Characters (8-bit Printer Port Environment)

ASSUMPTIONS: The printer supports all the character sets in use by the terminal, designating escape sequences for G0, G1, G2, G3, and character set shifts LS0, LS1, LS2R and LS3R. If the User Preference Supplemental Set (UPSS) extension is used, the printer has already been configured with the same UPSS. If a DRCS is used, the host has already down line loaded the printer's character set(s) to match the video terminal's (in Printer Controller Mode, if necessary). Printers with changeable font cartridges must support all of the above character sets simultaneously. No font selection commands will be used.

INITIALIZATION: The printer is initialized to correspond with the terminal. The character sets currently designated as G0, G1, G2, and G3 in the terminal are designated as G0, G1, G2, and G3 in the printer respectively. G0 is invoked to GL, and G2 is invoked to GR. The sequences sent are:

ESC (Dscs (Designate a 94 character set as G0)

```
ESC ) Dscs      (Designate a 94 character set as G1)
ESC * Dscs      (Designate a 94 character set as G2)
ESC + Dscs      (Designate a 94 character set as G3)
LS0             (Invoke G0 to GL)
LS2R            (Invoke G2 to GR)
```

Where "Dscs" consists of zero, one, or two intermediate characters and a final character for the character set being designated.

If a 96 character set is to be designated, the first intermediate character will change as follows:

```
ESC - Dscs      (Designate a 96 character set as G1)
ESC . Dscs      (Designate a 96 character set as G2)
ESC / Dscs      (Designate a 96 character set as G3)
```

If one of the character sets is the UPSS, it is designated as the UPS, not the specific character set chosen.

TRANSMISSION: The characters are sent using LS0/LS1 to choose between the character sets in G0 and G1, and LS2R/LS3R to choose between the character sets in G2 and G3. If a character from an alternate character set is encountered, the alternate set is designated as G3, invoked by LS3R, and then sent.

The priority order for sending characters contained in more than one set is ASCII, UPSS, DEC Supplemental, ISO Supplemental, DEC Technical, and DEC Special Graphics.

In this mode the display attributes are also sent. Refer to the Display Attributes section below.

CLOSING: LS0 and LS2R are sent if necessary. If an alternate set was the last set designated as G3, G3 is re-designated. This leaves the printer with G0 invoked as GL and G2 invoke as GR; G0, G1, G2, and G3 the same as the terminal.

FALLBACK: When Print All Characters is selected, fallback is only provided for CRM characters. CRM characters do not belong to a character set. Refer to the fallback section and tables later in this chapter.

Documented Exception

The "All Characters" mode in the VT200 family designated ASCII as G0, Line Drawing as G1, DEC Multinational as G2, and the DRCS as G3 if needed. In the VT300 family, possible support for the UPSS (8-bit architecture), DEC Technical, and DEC Publishing meant VT200 "All Characters" mode

really printed multinational characters. In the VT300, "All Characters" mode was changed to print all characters more efficiently. Some terminals may provide a "Multinational" mode similar to "All Characters" on the VT200 to support fallbacks for other character sets (DEC Technical for example).

7.4.1 National Replacement Character Sets

There are 12 NRC Sets currently defined for 16 keyboard dialects. The National Replacement Character set for each keyboard dialect is as follows:

Keyboard Dialect	NRC Set	Designating Characters
North American	ASCII	B (4/2)
Flemish	French NRC Set	R (5/2)
Canadian (French)	French Canadian NRC Set	9 (3/9)
British	U.K. NRC Set	A (4/1)
Danish	Norwegian/Danish NRC Set	` (6/0)
Finnish	Finnish NRC Set	5 (3/5)
Austrian/German	German NRC Set	K (4/11)
Dutch	ASCII	B (4/2)
Italian	Italian NRC Set	Y (5/9)
Swiss (French)	Swiss NRC Set	= (3/13)
Swiss (German)	Swiss NRC Set	= (3/13)
Swedish	Swedish NRC Set	7 (3/7)
Norwegian	Norwegian/Danish NRC Set	` (6/0)
Belgian/French	French NRC Set	R (5/2)
Spanish	Spanish NRC Set	Z (5/10)
Portuguese	Portuguese NRC Set	% 6 (2/5 3/6)

The National Replacement Character Sets are available only when the appropriate keyboard is selected in the terminal.

Deviation Note

VT200 family devices support the Dutch NRC set (designating character "4" (3/4)).

Guideline: LA120 non-conforming sequences: Finland "C" (4/3); Sweden "H" (4/8); French Canadian "Q" (5/1); Norway/Denmark "6" (3/6) or "E" (4/5).

7.4.2 Display Attributes

When the Printer Style is "National + Line Drawing", or "Print All Characters", the display attributes are also sent. The attributes are of two kinds: line attributes and character attributes. The line attributes will be initialized at the beginning of each line as follows:

Single Width lines -	ESC # 5
Double Width lines -	ESC # 6
Top half of Double Height -	ESC # 3
Bottom half of Double Height -	ESC # 4

The character attributes will be initialized at the start of each print operation as follows:

ESC [0 m

Every time the character attributes change, the appropriate control sequence setting the character attributes will be sent. The sequences are of the form:

ESC [0 ; Ps ; Ps ; Ps m

With the following parameters:

0 all attributes off
1 for bold character attribute
4 for underscore character attribute
5 for blink character attribute
7 for reverse video character attribute

At the end of each print (line or page), the following closing sequences will be sent as appropriate:

ESC [0 m

Note: characters with the "invisible" character attribute (SGR parameter 8) are always sent to the printer as spaces.

7.4.3 Trailing Spaces

Trailing spaces are spaces which come after all printable characters on a line. Note that in general trailing spaces themselves are not printed. In "National Only" trailing spaces are not printable and therefore are not sent to the printer. In "National + Line Drawing" and "All Characters" some spaces have video character attributes, and are therefore printable. In this context, spaces with the normal video character attribute which follow all printable characters (including spaces with video attributes on), are not sent to the printer.

7.5 FALLBACK PRESENTATION OF GRAPHIC CHARACTERS

When the Printer Style specifies that the device attached to the Printer Port is not capable of presenting graphic characters which can be presented on the terminal, the terminal attempts to display a reasonable approximation of that character on the printer for the Printer Port function operations, and in Auto-Print mode. These approximations, or "Fallbacks", are not necessarily unique but are intended to allow the reader of the result to be able to reconstruct the original text using context to resolve conflicts.

NOTE

These fallbacks are not performed in Printer Controller Mode or in Printer to Host operations.

7.5.1 NRC Fallbacks

No NRC fallbacks are required. There are no NRCS characters which are not in DEC Multinational or ISO Latin Alphabet Nr 1 (see the corresponding fallback sections).

Deviation Note

In older devices, two National Replacement Characters in the Dutch NRC set are not available in DEC-MCS, or ISO Latin-1. If the Printer Style is set to National Only or National + Line Drawing, and one of these characters occur in the data stream and is not available in the current national set, fallback is used as described below.

Character		Fallback Presentation
ij	ij vowel (Dutch NRCs 5/11)	ij
florin	florin (Dutch NRCs 7/12)	f

7.5.2 DEC Special Graphics Fallbacks

If the Printer Style is set to "Print National Only", the terminal will represent all characters from the DEC Special Graphic (VT100 Line Drawing) character set as one or two character sequences from the ASCII character set which approximate the desired graphic. These characters have been chosen so that diagrams or presentations that use the mosaic line combinations will align properly and present a reasonable appearance.

Character		Fallback Presentation
<**>	diamond	*
<##>	filled block	#
<HT>	HT symbol	HT
<FF>	FF symbol	FF
<CR>	CR symbol	CR
<LF>	LF symbol	LF
<+->	plus/minus sign	+
<0^>	degree sign	o
<NL>	NL symbol	NL
<VT>	VT symbol	VT
<_|>	lower right corner	+
<^|>	upper right corner	+
<|^>	upper left corner	+
<|_>	lower left corner	+
<+>	intersection	+
<^^>	scan 1	~
<^->	scan 3	=
<-->	scan 5	-
<-_>	scan 7	=
<__>	scan 9	_
<|->	intersect from right	+
<-|>	intersect from left	+
<_L>	intersect from top	+
<T>	intersect from bottom	+
<|>	vertical bar	|
<<=>	less than or equal to	<=
<>=>	greater than or equal to	>=
<][>	Pi symbol	Pi
</=>	not-equal to	<>
<L=>	pound sign	#
<.^>	centered dot	.

7.5.3 DEC Supplemental Fallbacks

If the Printer Style is set to "Print National Only", or "Print National + Line Drawing" the terminal will represent all characters from the DEC Supplemental character set as a sequence of one or more characters from the ASCII character set which approximate the desired graphic as described in the fallback table below.

Character			Fallback Presentation
<!!>	SP03	inverted exclamation mark	!
<c/>	SC04	cent sign	c
<L=>	SC02	pound sign	#
<Y=>	SC05	yen sign	Y
<So>	SM24	section sign	Sc
<Xo>	SC01	general currency sign	O
<CO>	SM52	copyright sign	(C)
<_a>	SM21	feminine ordinal	a
<<<>	SP17	angle quotation mark left	<<
<0^>	SM19	degree sign	o
<+->	SA02	plus/minus sign	+
<2^>	NS02	superscript 2	2
<3^>	NS03	superscript 3	3
</u>	SM17	micro sign	u
<!P>	SM25	paragraph sign	Pr
<.^>	SM26	middle dot	.
<1^>	NS01	superscript 1	1
<_o>	SM20	masculine ordinal	o
<>>>	SP18	angle quotation mark right	>>
<14>	NF04	fraction one quarter	1/4
<12>	NF01	fraction one half	1/2
<??>	SP16	inverted question mark	?
<a'>	LA11	small a with acute accent	a
<A'>	LA12	capital A with acute accent	A
<a`>	LA13	small a with grave accent	a
<A`>	LA14	capital A with grave accent	A
<a^>	LA15	small a with circumflex	a
<A^>	LA16	capital A with circumflex	A
<a">	LA17	small a with dieresis or umlaut	a
<A">	LA18	capital A with dieresis or umlaut	A
<a~>	LA19	small a with tilde	a
<A~>	LA20	capital A with tilde	A
<a*>	LA27	small a with ring	a
<A*>	LA28	capital A with ring	A
<ae>	LA51	small ae diphthong	ae
<AE>	LA52	capital AE diphthong	AE
<c,>	LC41	small c with cedilla	c
<C,>	LC42	capital C with cedilla	C
<e'>	LE11	small e with acute accent	e
<E'>	LE12	capital E with acute accent	E
<e`>	LE13	small e with grave accent	e
<E`>	LE14	capital E with grave accent	E
<e^>	LE15	small e with circumflex	e
<E^>	LE16	capital E with circumflex	E
<e">	LE17	small e with dieresis or umlaut	e
<E">	LE18	capital E with dieresis or umlaut	E
<i'>	LI11	small i with acute accent	i
<I'>	LI12	capital I with acute accent	I
<i`>	LI13	small i with grave accent	i
<I`>	LI14	capital I with grave accent	I
<i^>	LI15	small i with circumflex	i

<I^>	LI16 capital I with circumflex	I
<i">	LI17 small i with dieresis	i
<I">	LI18 capital I with dieresis	I
<n~>	LN11 small n with tilde	n
<N~>	LN12 capital N with tilde	N
<o'>	LO11 small o with acute accent	o
<O'>	LO12 capital O with acute accent	O
<o`>	LO13 small o with grave accent	o
<O`>	LO14 capital O with grave accent	O
<o^>	LO15 small o with circumflex	o
<O^>	LO16 capital O with circumflex	O
<o">	LO17 small o with dieresis or umlaut mark	o
<O">	LO18 capital O with dieresis or umlaut mark	O
<o~>	LO19 small o with tilde	o
<O~>	LO20 capital O with tilde	O
<oe>	LO51 small oe ligature	oe
<OE>	LO52 capital OE ligature	OE
<o/>	LO61 small o with slash	o
<O/>	LO62 capital O with slash	O
<ss>	LS61 German small sharp s	ss
<u'>	LU11 small u with acute accent	u
<U'>	LU12 capital U with acute accent	U
<u`>	LU13 small u with grave	u
<U`>	LU14 capital U with grave	U
<u^>	LU15 small u with circumflex	u
<U^>	LU16 capital U with circumflex	U
<u">	LU17 small u with dieresis or umlaut	u
<U">	LU17 capital U with dieresis or umlaut	U
<y">	LY17 small y with dieresis	y
<Y">	LY18 capital Y with dieresis	Y

7.5.4 ISO Latin-1 Supplemental Fallbacks

If the Printer Style is set to "Print National Only", or "Print National + Line Drawing", the terminal will represent all characters from the ISO Latin-1 supplemental character set as a sequence of one or more characters from the ASCII character set which approximate the desired graphic as described in the fallback table below.

Character			Fallback Presentation
<NBSP>	SP??	no break space	<space>
<!!>	SP03	inverted exclamation mark	!
<c/>	SC04	cent sign	c
<L=>	SC02	pound sign	#
<Xo>	SC01	general currency sign	O
<Y=>	SC05	yen sign	Y
<||>	SM??	broken bar	|
<So>	SM24	section sign	Sc
<"^>	SC01	diaeresis	"
<CO>	SM52	copyright sign	(C)
<_a>	SM21	feminine ordinal	a
<<<>	SP17	angle quotation mark left	<<
<-,>	SM??	logical not sign	~ (tilde)
<SHY>	SM18	soft hyphen	- (dash)
<RO>	SM53	registered trade mark sign	(R)
<-^>	SP31	macron, overline	- (dash)
<0^>	SM19	degree sign	o
<+->	SA02	plus/minus sign	+
<2^>	NS02	superscript 2	2
<3^>	NS03	superscript 3	3
<''>	SD11	acute accent	' (apostrophe)
</u>	SM17	micro sign	u
<!P>	SM25	paragraph sign	Pr
<.^>	SM26	middle dot	.
<,,>	SD41	cedilla	, (comma)
<1^>	NS01	superscript 1	1
<_o>	SM20	masculine ordinal	o
<>>>	SP18	angle quotation mark right	>>
<14>	NF04	fraction one quarter	1/4
<12>	NF01	fraction one half	1/2
<34>	NF05	fraction three quarters	3/4
<??>	SP16	inverted question mark	?
<A`>	LA14	capital A with grave accent	A
<A'>	LA12	capital A with acute accent	A
<A^>	LA16	capital A with circumflex	A
<A">	LA18	capital A with diaeresis or umlaut	A
<A~>	LA20	capital A with tilde	A
<A*>	LA28	capital A with ring	A
<AE>	LA52	capital AE diphthong	AE
<C,>	LC42	capital C with cedilla	C
<E`>	LE14	capital E with grave accent	E
<E'>	LE12	capital E with acute accent	E
<E^>	LE16	capital E with circumflex	E
<E">	LE18	capital E with diaeresis or umlaut	E
<I`>	LI14	capital I with grave accent	I
<I'>	LI12	capital I with acute accent	I
<I^>	LI16	capital I with circumflex	I
<I">	LI18	capital I with diaeresis	I
<-D>	LD62	capital Icelandic letter Eth	D
<N~>	LN12	capital N with tilde	N
<O`>	LO14	capital O with grave accent	O

<O'>	LO12	capital O with acute accent	O
<O^>	LO16	capital O with circumflex	O
<O~>	LO20	capital O with tilde	O
<O">	LO18	capital O with diaeresis or umlaut	O
<xx>		times sign	x
<O/>	LO62	capital O with slash	O
<U`>	LU14	capital U with grave	U
<U'>	LU12	capital U with acute accent	U
<U^>	LU16	capital U with circumflex	U
<U">	LU17	capital U with diaeresis or umlaut	U
<Y'>	LY12	capital y with acute accent	Y
<TH>	LT64	capital Icelandic letter Thorn	Th
<ss>	LS61	German small sharp s	ss
<a`>	LA13	small a with grave accent	a
<a'>	LA11	small a with acute accent	a
<a^>	LA15	small a with circumflex	a
<a~>	LA19	small a with tilde	a
<a">	LA17	small a with diaeresis or umlaut	a
<ae>	LA51	small ae diphthong	ae
<a*>	LA27	small a with ring	a
<c,>	LC41	small c with cedilla	c
<e`>	LE13	small e with grave accent	e
<e'>	LE11	small e with acute accent	e
<e^>	LE15	small e with circumflex	e
<e">	LE17	small e with diaeresis or umlaut	e
<i`>	LI13	small i with grave accent	i
<i'>	LI11	small i with acute accent	i
<i^>	LI15	small i with circumflex	i
<i">	LI17	small i with diaeresis	i
<-d>	LD63	small Icelandic letter Eth	d
<n~>	LN11	small n with tilde	n
<o`>	LO13	small o with grave accent	o
<o'>	LO11	small o with acute accent	o
<o^>	LO15	small o with circumflex	o
<o~>	LO19	small o with tilde	o
<o">	LO17	small o with diaeresis or umlaut	o
<-:>		division sign	-
<o/>	LO61	small o with slash	o
<u`>	LU13	small u with grave	u
<u'>	LU11	small u with acute accent	u
<u^>	LU15	small u with circumflex	u
<u">	LU17	small u with diaeresis or umlaut	u
<y'>	LY11	small y with acute accent	y
<th>	LT63	small Icelandic letter Thorn	th
<y">	LY17	small y with diaeresis	y

7.5.5 DEC Technical Character Set Fallbacks

If the Printer Style is set to anything other than "Print All Characters", the terminal will represent all characters from a ROM resident (not DRCS) DEC Technical Character set as one or two

character sequences from the ASCII character set which approximate the desired graphic. These characters have been chosen so that diagrams or presentations that use the mosaic line combinations will approximately represent the displayed characters and present a reasonable appearance.

Character		Fallback Presentation
<-v>	left radical	-v
<|->	top left radical	|-
<Horiz>	horizontal connector	- (minus)
<TopI>	top integral	(
<BotI>	Bottom integral	)
<Vert>	vertical connector	| (vertical line)
<Top[>	top left square bracket	[
<Bot[>	bottom left square bracket	[
<Top]>	top right square bracket	]
<Bot]>	bottom right square bracket	]
<Top(>	top left parenthesis	(
<Bot(>	bottom left parenthesis	(
<Top)>	top right parenthesis	)
<Bot)>	bottom right parenthesis	)
<{{>	left middle curly brace	{
<}}>	right middle curly brace	}
<TopS>	top left summation	>
<BotS>	bottom left summation	<
<\\>	top vertical summation connector	\ (backslash)
<//>	bottom vertical summation connector	/ (slash)
<TopRS>	top right summation	none
<BotRS>	bottom right summation	none
<>S>	right middle summation	>
<<=>	less than or equal	<
<=/>	not equal	=
<>=>	greater than or equal	>
<Integ>	integral	S
<:.>	therefore	:.(colon,period)
<var>	variation, proportional to	a
<oO>	infinity	oO
<-:>	division, divided by	:
<DELTA>	capital delta, triangle	D
<nabla>	nabla, del	v
<PHI>	capital phi	F
<GAMMA>	capital gamma	G
<~~>	is approximate to	~ (tilde)
<~->	similar or equal to	~ (tilde)
<THETA>	capital theta	J
<xx>	times, cross product	x
<LAMBD>	capital lambda	L
<<=>>	if and only if	<=>
<=>>	implies	=>
<=->	is identical to	=

<PI>	capital pi, product	P
<PSI>	capital psi	Q
<SIGMA>	capital sigma, summation	S
<Rad>	radical	V~
<OMEGA>	capital omega, Ohm sign	W
<XI>	capital xi	X
<UPSIL>	capital upsilon	Y
<cc>	is included in	c
<inclu>	includes	none
<inter>	intersection	^ (circumflex)
<union>	union	v
<L&>	logical and	& (ampersand)
<L|>	logical or	| (vertical line)
<L~>	logical not	~ (tilde)
<alpha>	small alpha	a
<beta>	small beta	b
<chi>	small chi	c
<delta>	small delta	d
<epsil>	small epsilon	e
<phi>	small phi	f
<gamma>	small gamma	g
<eta>	small eta	h
<iota>	small iota	i
<theta>	small theta	j
<kappa>	small kappa	k
<lambd>	small lambda	l
<nu>	small nu	n
<pd>	partial derivative	none
<pi>	small pi	p
<psi>	small psi	q
<rho>	small rho	r
<sigma>	small sigma	s
<tau>	small tau	t
<func>	function	f
<omega>	small omega	w
<xi>	small xi	x
<upsil>	small upsilon	y
<zeta>	small zeta	z
<<->	left arrow	<-
<^|>	up arrow	^
<->>	right arrow	->
<v|>	down arrow	v

NOTE

Wherever the word "none" appears in the above chart, the fallback of last resort, the SPACE character, shall be sent to the printer port as the fallback presentation for that character.

7.5.6 DRCS Fallbacks

If the Printer Style is set to anything other than "Print All Characters", the SPACE character (2/0) is printed in place of any character from a DRCS, except when the designating string of the DRCS matches a ROM-based character set that may be printed. If the DRCS is substituting for a ROM-based character set, the appropriate character from the ROM-based character set is used (if it can be printed without fallback).

Deviation Note

> The printer fallback character on the VT200 series was the underscore ("_"). This was changed to space to simplify manual editing of documents containing fallback characters.

7.5.7 User Preference Supplemental Set Fallbacks

The terminal must know the User Preference Supplemental Set even though the host may not. If the UPSS is DEC or ISO Supplemental, the terminal will use the corresponding fallback table previously given. UPSS fallback is only used when the Printer Port is set to Print National Only, or Print National + Line Drawing.

7.5.8 Control Representation Mode Fallbacks (not Mandatory)

Control Representation characters are presented to the printer as a pair of capital ASCII letters or digits representing the name of the C0 or C1 control character. If the printer supports the underscore graphic rendition and "Print National + Line Drawing", or "Print All Characters" is selected, the CRM fallback combinations are also underlined to distinguish them from normal text.

Control Character		Fallback Presentation
0/0	Null	NU
0/1	Start of Header	SH
0/2	Start Transmission	SX
0/3	End of Text	EX
0/4	End of Transmission	ET
0/5	Enquiry	EQ
0/6	Acknowledge	AK
0/7	Bell	BL
0/8	Backspace	BS
0/9	Horizontal Tab	HT
0/10	Line Feed	LF
0/11	Vertical Tab	VT

0/12	Form Feed	FF
0/13	Carriage Return	CR
0/14	Shift Out (Locking Shift 1)	SO
0/15	Shift In (Locking Shift 0)	SI
1/0	Data Link Escape	DL
1/1	Device Control 1	D1
1/2	Device Control 2	D2
1/3	Device Control 3	D3
1/4	Device Control 4	D4
1/5	Negative Acknowledge	NK
1/6	Synchronous Idle	SY
1/7	End of Transmission Block	EB
1/8	Cancel	CN
1/9	End of Medium	EM
1/10	Substitute	SB
1/11	Escape	EC
1/12	File Separator	FS
1/13	Group Separator	GS
1/14	Record Separator	RS
1/15	Unit Separator	US
8/0		80
8/1		81
8/2		82
8/3		83
8/4	Index	IN
8/5	Next Line	NL
8/6	Start of Selected Area	SS
8/7	End of Selected Area	ES
8/8	Horizontal Tabulation Set	HS
8/9	Horizontal Tab with Justification	HJ
8/10	Vertical Tabulation Set	VS
8/11	Partial Line Down	PD
8/12	Partial Line Up	PU
8/13	Reverse Index	RI
8/14	Single Shift 2	S2
8/15	Single Shift 3	S3
9/0	Device Control String	DC
9/1	Private Use 1	P1
9/2	Private Use 2	P2
9/3	Set Transmit State	SE
9/4	Cancel Character	CC
9/5	Message Waiting	MW
9/6	Start of Protected Area	SP
9/7	End of Protected Area	EP
9/8		98
9/9		99
9/10		9A
9/11	Control Sequence Introducer	CS
9/12	String Terminator	ST
9/13	Operating System Command	OS
9/14	Privacy Message	PM
9/15	Application Program Command	AP

7.6 PRINT OPERATIONS

7.6.1 Print Page Or Scrolling Region

The text for the current page of the current session or just the lines in the current page which are in the active scrolling region are transferred to the printer. Which occurs depends on the setting of Printer Extent Mode (DECPEX). This function is invoked by the host through the Media Copy (MC) control function, or by the user pressing the LOCAL PRINT key.

7.6.2 Print Screen (not Mandatory)

A series of characters representing the composed image on the Main Display is transferred to the printer. The intent of this operation is to reproduce the contents of the screen as if a snapshot had been taken. Due to various terminal and printer limitations, some compromises may be necessary for complex displays. The Main Display may consist of information from more than one Session depending on the User Window configuration.

This function is not affected by Printer Extent Mode (DECPEX). Print Screen is invoked by the host through the Media Copy (MC) control function, or by the user pressing the SHIFT key and LOCAL PRINT key together.

Guidelines:

If the Sixel Graphics Extension is present and graphics printing is enabled, the contents of the screen (both text and graphics) may be transmitted as sixels.

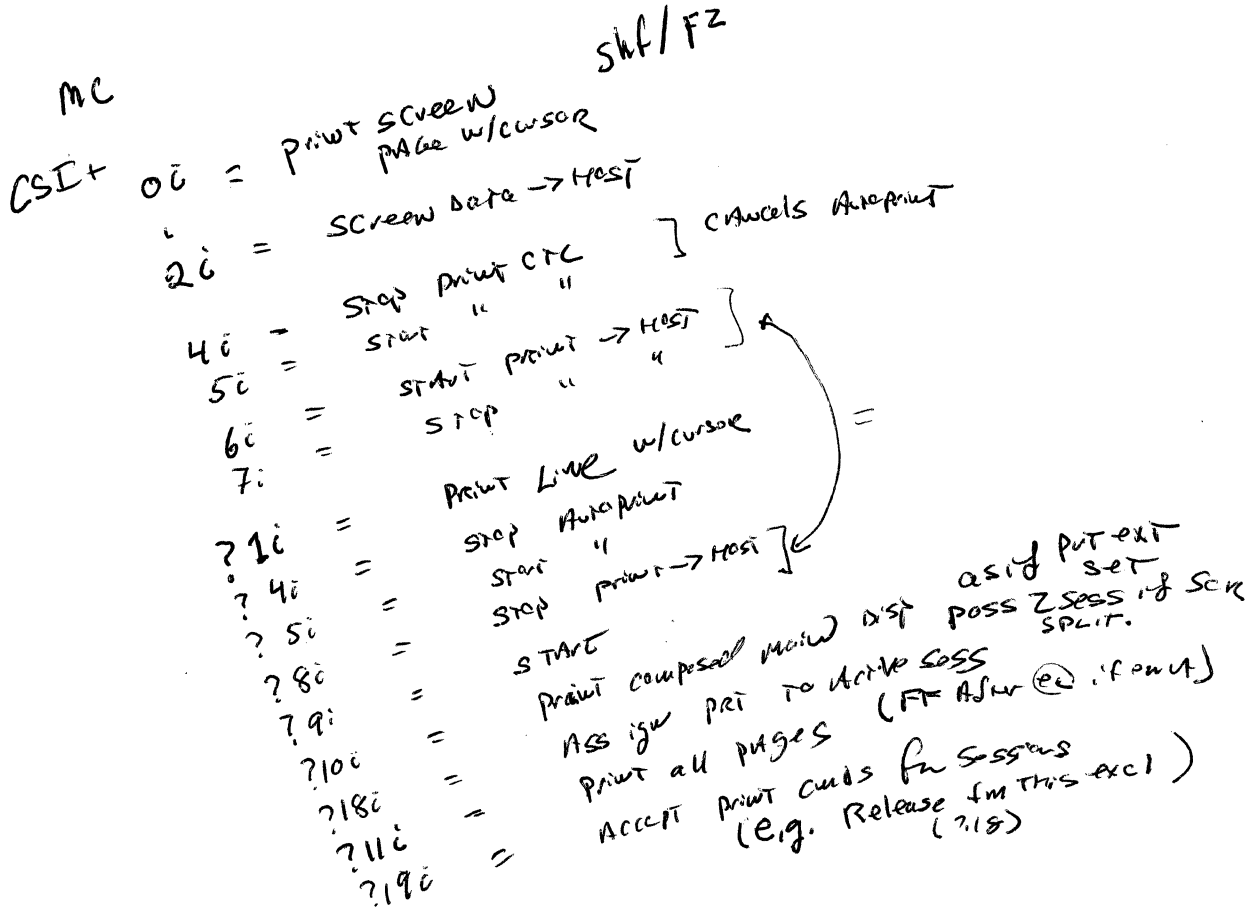
If graphics printing is not used, borders on the Main Display may be transmitted to the Printer Port as characters from the DEC Special Graphics character set.

7.6.3 Print All Pages (guideline)

The text from all pages in the active session is transferred to the printer. The setting of Printer Extent Mode (DECPEX) affects this function. If Print Form Feed (DECPFF) mode is set, a form feed (FF) character will be transmitted at each page boundary. The pages are transmitted in numerical order starting with page one. This function is invoked by the host through the Media Copy (MC) control function.

7.6.4 Print Line

The text of the line on which the output cursor resides is transferred to the printer. Whether the left and right scrolling regions (set through DECSLRM) affect which characters are sent to the Printer Port depends on the setting of Printer Extent Mode (DECPEX). This function is invoked by the host through the Media Copy (MC) control function.



CSI ?19 h Print Ext — Set (whole)
l CLR (Scroll)

?18 Print FF — Set FF
CLR NONE

Ps) p 0,1 National only; 2 + Line; 3 MULTINAT; 4 ALL

7.7 CONTROL FUNCTIONS

REPORT PRINTER STATUS **DSR**

Levels: 1X, 2X, 3X

Purpose: Report the status of the printer

Request Format:	CSI	?	15	n
	9/11	3/15	3/1 3/5	6/14
Report Format:	CSI	?	Ps	n
	9/11	3/15	Ps	6/14

Description: The terminal will respond to a DSR request by transmitting a Device Status Report control sequence containing one of the following selective parameters (preceded by the private parameter value ?, 3/15):

Selective Parameter	Meaning
10	The printer is ready to print. The Data Set Ready signal is asserted at the printer port.
11	The printer is not ready to print. The Data Set Ready signal at the printer port was previously asserted (since power on), but is not asserted now.
13	The printer is not connected to the terminal. The Data Set Ready signal at the printer port has not been on since the last power-up or reset (RIS).
Printer Status on Virtual Terminals (Session Management Extension Only)	
18	The printer is busy. The Data Set Ready signal at the printer port is asserted, but an alternate session is currently sending information to the printer, has the printer in auto-print mode, or has the printer in printer controller mode.
19	The printer is not assigned to the requesting session. The current session may not access the printer port (regardless of Data Set Ready) because the user has explicitly assigned it to another session.

Notes:

1. See the section State Descriptions - Printer Status for a complete description of how the status of the printer is maintained.

State Affected: None

Algorithm:

```
PROCEDURE REPORT_PRINTER_STATUS;
BEGIN
CASE PRINTER_STATUS OF
        PRINTER_READY     : WRITE (HOST_PORT, '<CSI>?10n');
        PRINTER_NOT_READY : WRITE (HOST_PORT, '<CSI>?11n');
        PRINTER_NO_PRESENT: WRITE (HOST_PORT, '<CSI>?13n');
        PRINTER_BUSY      : WRITE (HOST_PORT, '<CSI>?18n');
        PRINTER_NOT_ASSIGN: WRITE (HOST_PORT, '<CSI>?19n');
        END;
END;
```

Known Deviations: None

SET/RESET PRINTER CONTROLLER MODE MC

Levels: 1X, 2X, 3X

Purpose: To set or reset the state of Printer Controller Mode.

Set Format:	CSI	5	i
	9/11	3/5	6/9
Reset Format:	CSI	4	i
	9/11	3/4	6/9

Description: When Printer Controller Mode is set, all data received by the terminal is transmitted to the printer port without being displayed on the screen.

Notes:

1. When Printer Controller Mode is set, the state of Auto Print Mode becomes UNDEFINED. That is, when Printer Controller Mode is exited, Auto Print Mode may not be in the same state as it was when Printer Controller Mode was entered.

State Affected:

```
PRINTER_CONTROLLER_MODE:
  (PRINTER_CONTROLLER_OFF,PRINTER_CONTROLLER_ON);
```

Algorithm:

```
PROCEDURE SET_PRINTER_CONTROLLER_MODE;
BEGIN
PRINTER_CONTROLLER_MODE:= PRINTER_CONTROLLER_ON;
END;

PROCEDURE RESET_PRINTER_CONTROLLER_MODE;
BEGIN
PRINTER_CONTROLLER_MODE:= PRINTER_CONTROLLER_OFF;
AUTO_PRINT_MODE: = AUTO_PRINT_OFF;
END;
```

Known Deviations: None

SET/RESET AUTO PRINT MODE MC

Levels 1X, 2X, 3X

Purpose: To set or reset the state of Auto Print Mode.

Set Format:	CSI	?	5	i
	9/11	3/15	3/5	6/9
Reset Format:	CSI	?	4	i
	9/11	3/15	3/5	6/9

Description: When Auto Print Mode is set, the contents of the Active Line is transmitted to the printer port whenever the Active Position is advanced to the next line as a result of one of the following conditions:

1. Receipt of a Line Feed (LF), Vertical Tab (VT), or Form Feed (FF) control character.

2. When the Last Column Flag is set and a graphic character is received.

Notes:

1. When Printer Controller Mode is set, the state of Auto Print Mode becomes UNDEFINED. That is, when Printer Controller Mode is exited, Auto Print Mode may not be in the same state as it was when Printer Controller Mode was entered.

State Affected:

```
AUTO_PRINT_MODE:  (AUTO_PRINT_OFF,AUTO_PRINT_ON);
```

Algorithm:

```
PROCEDURE SET_AUTO_PRINT_MODE;
BEGIN
AUTO_PRINT_MODE:= AUTO_PRINT_ON;
END;

PROCEDURE RESET_AUTO_PRINT_MODE;
BEGIN
AUTO_PRINT_MODE:= AUTO_PRINT_OFF;
END;
```

Known Deviations: None

SET/RESET PRINTER TO HOST MODE MC

Levels 1X, 2X, 3X

Purpose: To set or reset the state of Printer To Host Mode.

Set Format:	CSI	?	9	i
	9/11	3/15	3/9	6/9
Reset Format:	CSI	?	8	i
	9/11	3/15	3/8	6/9

Description: When Printer To Host Mode is set, all input from the printer port (other than XON/XOFF flow control when enabled) is relayed to the host. Character input from the printer port is interleaved with input from the keyboard as it is received.

Notes:

1. Printer To Host Mode functions in Printer Controller Mode as well as other print modes.

2. When available, Printer To Host Mode works at all operating levels (Level 1, Level 2, and Level 3).

State Affected:

```
PRINTER_TO_HOST_MODE:  (PRINTER_TO_HOST, NO_PRINTER_TO_HOST);
```

Algorithm:

```
PROCEDURE SET_PRINTER_TO_HOST_MODE;
BEGIN
PRINTER_TO_HOST_MODE := PRINTER_TO_HOST;
END;

PROCEDURE RESET_PRINTER_TO_HOST_MODE;
BEGIN
PRINTER_TO_HOST_MODE := NO_PRINTER_TO_HOST;
END;
```

Known Deviations: None

SET/RESET PRINT FORM FEED MODE DECPFF

Levels: 1X, 2X, 3X

Purpose: To set or reset the state of Print Form Feed Mode.

Set Format:	CSI	?	18	h
	9/11	3/15	3/1 3/8	6/8
Reset Format:	CSI	?	18	l
	9/11	3/15	3/1 3/8	6/12

Description: When Print Form Feed Mode is set, a Form Feed (FF) control character is transmitted to the printer port at the conclusion of each Print Page or Print Screen operation.

State Affected:

```
PRINT_FORM_FEED_MODE: (PRINT_FF_OFF,PRINT_FF_ON);
```

Algorithm:

```
PROCEDURE SET_PRINT_FORM_FEED_MODE;
BEGIN
PRINT_FORM_FEED_MODE:= PRINT_FF_ON;
END;

PROCEDURE RESET_PRINT_FORM_FEED_MODE;
BEGIN
PRINT_FORM_FEED_MODE:= PRINT_FF_OFF;
END;
```

Known Deviations: None

SET/RESET PRINT EXTENT MODE DECPEX

Levels: 1X, 2X, 3X

Purpose: To set or reset the state of Print Extent Mode.

Set Format:	CSI	?	19		h
	9/11	3/15	3/1	3/9	6/8
Reset Format:	CSI	?	19		l
	9/11	3/15	3/1	3/9	6/12

Description: Print Extent Mode is used to select the area of the display to be transmitted to the printer port when a Print Page operation is performed. When this mode is in the state (Print Display) the entire display image is transmitted. When it is in the reset state (Print Scrolling Region), only the area of the display within the scrolling region (Top Margin to Bottom Margin inclusive) is transmitted.

State Affected:

```
PRINT_EXTENT_MODE:
  (PRINT_SCROLLING_REGION,PRINT_DISPLAY);
```

Algorithm:

```
PROCEDURE SET_PRINT_EXTENT_MODE;
BEGIN
PRINT_EXTENT_MODE:= PRINT_DISPLAY;
END;

PROCEDURE RESET_PRINT_EXTENT_MODE;
BEGIN
PRINT_EXTENT_MODE:= PRINT_SCROLLING_REGION;
END;
```

Known Deviations: None

PRINT PAGE MC

Levels 1X, 2X, 3X

Purpose: Transmit the current page to the printer port.

Format:

	CSI	0	i	default Ps: 0
	9/11	3/0	6/9	

Description: The Print Page function causes the text for the current page of the current session or just the lines in the current page which are in the active scrolling region to be transferred to the printer. Which occurs depends on the setting of Printer Extent Mode (DECPEX).

Notes:

1. Transmission of the text stream includes all necessary character set and graphic rendition controls as well as format effectors to properly represent the position of text on the page.

State Affected: None

Algorithm:

```
PROCEDURE PRINT_PAGE;
VAR     Y,TOP,BOTTOM: LINE_TYPE;
        X: COLUMN_TYPE;
        P_END_OF_LINE: INTEGER;
        P_LINE_RENDITION: LINE_RENDITION_TYPE;
        P_RENDITION: CHARACTER_RENDITION_TYPE;
        P_GL_CHARACTER_SET: GRAPHICS_CHARACTER_SET_TYPE;
        P_GR_CHARACTER_SET: GRAPHICS_CHARACTER_SET_TYPE;
        P_CURRENT_NRCS:  GRAPHICS_CHARACTER_SET_TYPE;
        P_G1_DESIGNATE:  GRAPHICS_CHARACTER_SET_TYPE;
BEGIN
(* determine print extent *)
IF PRINT_EXTENT_MODE = PRINT_DISPLAY THEN
  BEGIN
  TOP:= 1;
  BOTTOM:= MAX_NUM_LINES;
  END;
ELSE
  BEGIN
  TOP:= TOP_MARGIN;
  BOTTOM:= BOTTOM_MARGIN;
  END;

(* determine current National character set *)
P_CURRENT_NRCS:= ASCII_G;
IF (CHARACTER_SET_MODE = SEVEN_BITS) AND
```

```
    (KEYBOARD_USAGE_MODE = TYPEWRITER) THEN
  P_CURRENT_NRCS:= NRCS_TABLE(KEYBOARD_DIALECT);

(* handle each printer style separately *)
IF PRINTER_STYLE = NATIONAL_ONLY THEN
  BEGIN  (* Print National Only *)
  (* initialization *)
  P_GL_CHARACTER_SET:= P_CURRENT_NRCS;
  (* transmit data as characters from GL *)
  (* for each line *)
  FOR Y:= TOP TO BOTTOM DO
    BEGIN
    (* set end of line to skip trailing spaces *)
    P_END_OF_LINE = END_OF_LINE(Y);
    WHILE (DISPLAY[Y,P_END_OF_LINE].CODE = SPACE_CODE) AND
          (P_END_OF_LINE > 0)  DO
      P_END_OF_LINE:= P_END_OF_LINE - 1;
    (* for each character in the current line *)
    FOR X:= 1 TO P_END_OF_LINE DO
      BEGIN
      (* if not in current NRCS *)
      IF DISPLAY[Y,X].CHARACTER_SET <> P_GL_CHARACTER_SET THEN
        BEGIN
        (* do fallback - to be specified *)
        END
      ELSE
        (* otherwise send character code to printer *)
        WRITE (PRINTER_PORT, CHR(DISPLAY[Y,X].CODE));
      END; (* end for each character *)
    WRITE (PRINTER_PORT,'<CR><LF>');
    END;  (* end for each line *)
  IF PRINT_FORM_FEED_MODE = PRINT_FF_ON THEN
    WRITE (PRINTER_PORT,'<FF>');
  (* no termination sequence required *)
  END;  (* end National Only *)

ELSE IF PRINTER_STYLE = NATIONAL_PLUS_LINE_DRAWING THEN
  BEGIN
  (* Printer Style is National + Line Drawing *)
  (* initialization sequence *)
  P_RENDITION[BOLD]:= FALSE;
  P_RENDITION[UNDERSCORE]:= FALSE;
  P_RENDITION[BLINK]:= FALSE;
  P_RENDITION[REVERSE]:= FALSE;
  WRITE (PRINTER_PORT,'<CSI>0m');
  P_GL_CHARACTER_SET:= P_CURRENT_NRCS;
  (* designate current NRCS as G0
     ASCII is NRCS for North America *)
  WRITE (PRINTER_PORT, '<ESC>(');
  WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[P_GL_CHARACTER_SET]);
  (* designate line drawing as G1 *)
  WRITE (PRINTER_PORT, '<ESC>)0');
  (* invoke G0 to GL *)
```

```
    WRITE (PRINTER_PORT, '<SI>');

    (* data transmission *)
    (* for each line *)
    FOR Y:= TOP TO BOTTOM DO
      BEGIN
      (* set line rendition *)
      SEND_LINE_ATTRIBUTES(LINE_RENDITION[Y]);
      (* set end of line to skip trailing spaces
         with "normal" graphic rendition *)
      P_END_OF_LINE = END_OF_LINE(Y);
      WHILE (DISPLAY[Y,P_END_OF_LINE].CODE = SPACE_CODE) AND
            (P_END_OF_LINE > 0) AND
            (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[BOLD]) AND
            (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[UNDERSCORE]) AND
            (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[BLINK]) AND
            (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[REVERSE]) DO
        P_END_OF_LINE:= P_END_OF_LINE - 1;
      (* for each character in the current line *)
      FOR X:= 1 TO P_END_OF_LINE DO
        BEGIN
        (* set graphic rendition if it has changed *)
        SEND_GRAPHIC_RENDITION (P_RENDITION, DISPLAY[X,Y].RENDITION);
        P_RENDITION[BOLD]:= DISPLAY[Y,X].RENDITION[BOLD];
        P_RENDITION[UNDERSCORE]:=
          DISPLAY[Y,X].RENDITION[UNDERSCORE];
        P_RENDITION[BLINK]:= DISPLAY[Y,X].RENDITION[BLINK];
        P_RENDITION[REVERSE]:=
          DISPLAY[Y,X].RENDITION[REVERSE];

        (* new character set? *)
        IF DISPLAY[Y,X].CHARACTER_SET <> P_GL_CHARACTER_SET THEN
          BEGIN
          (* yes, send SCS sequence to printer *)
          CASE DISPLAY[Y,X].CHARACTER_SET OF

            P_CURRENT_NRCS:
              BEGIN
              (* character set is current NRCS *)
              (* invoke G0 to GL *)
              WRITE (PRINTER_PORT, '<SI>');
              P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
              END;

            LINE_DRAWING:
              BEGIN
              (* character set is Line Drawing *)
              (* is Line Drawing in G1? *)
              IF P_G1_DESIGNATE <> LINE_DRAWING THEN
                BEGIN
                (* no, re-designate it *)
                WRITE (PRINTER_PORT, '<ESC>)0');
                P_G1_DESIGNATE:= LINE_DRAWING;
```

```
            END;
          (* invoke G1 to GL *)
          WRITE (PRINTER_PORT,'<SO>');
          P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
          END;

        ASCII_G:
          (* character set is ASCII *)
          (* is ASCII current NRCS? *)
          IF P_CURRENT_NRCS <> ASCII_G THEN
            BEGIN
            (* no, if it's not already in G1, designate it *)
            IF P_G1_DESIGNATE <> ASCII_G THEN
              BEGIN
              WRITE (PRINTER_PORT, '<ESC>)B');
              P_G1_DESIGNATE:= ASCII_G;
              END;
            (* invoke G1 to GL *)
            WRITE (PRINTER_PORT, '<SO>');
            P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
            END;

          OTHERWISE: IGNORE;
          END;  (* end case of display character set *)
        END;  (* end new character set *)

      (* do we need to fallback? *)
      IF (DISPLAY[X,Y].CHARACTER_SET <> P_CURRENT_NRCS) AND
         (DISPLAY[X,Y].CHARACTER_SET <> LINE_DRAWING) AND
         (DISPLAY[X,Y].CHARACTER_SET <> ASCII_G) THEN
        BEGIN
        (* do fallback - to be specified *)
        END
      ELSE
        (* no, just send character code to printer *)
        WRITE (PRINTER_PORT, CHR(DISPLAY[Y,X].CODE));
      END;  (* end for each character in the current line *)
    WRITE (PRINTER_PORT,'<CR><LF>');
    END;  (* end for each line *)
  IF PRINT_FORM_FEED_MODE = PRINT_FF_ON THEN
    WRITE (PRINTER_PORT,'<FF>');

  (* termination sequence *)
  (* send LS0 if necessary to leave G0 invoked to GL *)
  IF P_GL_CHARACTER_SET <> P_CURRENT_NRCS THEN
    WRITE (PRINTER_PORT, '<SI>');
  (* if Line Drawing isn't in G1, designate it *)
  IF P_G1_DESIGNATE <> LINE_DRAWING THEN
    WRITE (PRINTER_PORT, '<ESC>)0');

  END;  (* end printer style is National + Line Drawing *)

ELSE IF PRINTER_STYLE = ALL_CHARACTERS THEN
```

```
    BEGIN
    (* printer style is All Characters *)
    (* initialization sequence *)
    P_RENDITION[BOLD]:= FALSE;
    P_RENDITION[UNDERSCORE]:= FALSE;
    P_RENDITION[BLINK]:= FALSE;
    P_RENDITION[REVERSE]:= FALSE;
    WRITE (PRINTER_PORT,'<CSI>0m');

    (* initialize printer G sets to correspond with terminal *)
    WRITE (PRINTER_PORT, '<ESC>(');
    WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G0]]);

    IF CHARACTER_SET_SIZE[DESIGNATED_GRAPHIC_SETS[G1]] = 94 THEN
      WRITE (PRINTER_PORT, '<ESC>)');
    ELSE
      WRITE (PRINTER_PORT, '<ESC>-');
    WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G1]]);

    IF CHARACTER_SET_SIZE[DESIGNATED_GRAPHIC_SETS[G2]] = 94 THEN
      WRITE (PRINTER_PORT, '<ESC>*');
    ELSE
      WRITE (PRINTER_PORT, '<ESC>.');
    WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G2]]);

    IF CHARACTER_SET_SIZE[DESIGNATED_GRAPHIC_SETS[G3]] = 94 THEN
      WRITE (PRINTER_PORT, '<ESC>+');
    ELSE
      WRITE (PRINTER_PORT, '<ESC>/');
    WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G3]]);

    (* send LS0 to invoke G0 to GL *)
    WRITE (PRINTER_PORT, '<SI>');
    P_GL_CHARACTER_SET:= DESIGNATED_GRAPHIC_SETS[G0];
    (* if 8-bit printer port environment *)
    IF PRINTER_PORT_ENVIRONMENT = EIGHT_BIT THEN
      BEGIN
      (* send LS2R to invoke G2 to GR *)
      WRITE (PRINTER_PORT, '<ESC>}');
      P_GR_CHARACTER_SET:= DESIGNATED_GRAPHIC_SETS[G2];
      END;


    (* data transmission *)
    FOR Y:= TOP TO BOTTOM DO
      BEGIN
      (* set line rendition *)
      SEND_LINE_ATTRIBUTES(LINE_RENDITION[Y]);
      (* set end of line to skip trailing spaces
         with "normal" graphic rendition *)
      P_END_OF_LINE = END_OF_LINE(Y);
      WHILE (DISPLAY[Y,P_END_OF_LINE].CODE = SPACE_CODE) AND
            (P_END_OF_LINE > 0) AND
```

```
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[BOLD]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[UNDERSCORE]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[BLINK]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[REVERSE]) DO
      P_END_OF_LINE:= P_END_OF_LINE - 1;
    (* for each character in the current line *)
    FOR X:= 1 TO P_END_OF_LINE DO
      BEGIN
      (* set graphic rendition *)
      SEND_GRAPHIC_RENDITION (P_RENDITION, DISPLAY[X,Y].RENDITION);
      P_RENDITION[BOLD]:= DISPLAY[Y,X].RENDITION[BOLD];
      P_RENDITION[UNDERSCORE]:=
        DISPLAY[Y,X].RENDITION[UNDERSCORE];
      P_RENDITION[BLINK]:= DISPLAY[Y,X].RENDITION[BLINK];
      P_RENDITION[REVERSE]:=
        DISPLAY[Y,X].RENDITION[REVERSE];

      (* handle 7-bit/8-bit printer port environment separately *)
      (* if 7-bit printer port environment *)
      IF PRINTER_PORT_ENVIRONMENT = SEVEN_BIT THEN
        BEGIN
        (* new character set? *)
        IF DISPLAY[Y,X].CHARACTER_SET <> P_GL_CHARACTER_SET THEN
          BEGIN
          (* yes, designate and invoke as needed *)
          CASE DISPLAY[Y,X].CHARACTER_SET OF

            DESIGNATED_GRAPHIC_SETS[G0]:
              (* character set is in G0 *)
              (* send LS0 to invoke G0 to GL *)
              WRITE (PRINTER_PORT, '<SI>');

            DESIGNATED_GRAPHIC_SETS[G1]:
              (* character set is in G1 *)
              (* send LS1 to invoke G1 to GL *)
              WRITE (PRINTER_PORT, '<SO>');

            DESIGNATED_GRAPHIC_SETS[G2]:
              (* character set is in G2 *)
              (* send SS2 to invoke G2 to GL *)
              WRITE (PRINTER_PORT, '<ESC>N');

            DESIGNATED_GRAPHIC_SETS[G3]:
              (* character set is in G3 *)
              (* send SS3 to invoke G3 to GL *)
              WRITE (PRINTER_PORT, '<ESC>O');

            OTHERWISE:
              BEGIN
              (* no fallback for print all characters *)
              (* designate the displayed set as G3 *)
              IF CHARACTER_SET_SIZE[DISPLAY[Y,X].CHARACTER_SET] = 94 THEN
                WRITE (PRINTER_PORT, '<ESC>+');
```

```
                ELSE
                  WRITE (PRINTER_PORT, '<ESC>/');
                WRITE (PRINTER_PORT, CHARACTER_SET_DSCS
                [DISPLAY[Y,X].CHARACTER_SET]);
                P_DESIGNATED_CHARACTER_SETS[G3]:=
                   DISPLAY[Y,X].CHARACTER_SET;
                (* send SS3 to invoke G3 to GL *)
                WRITE (PRINTER_PORT, '<ESC>O');
                END;
              END;  (* end case of display character set *)

            P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
            END;  (* end new character set *)
          END  (* end 7-bit printer port environment *)

        ELSE
          (* 8-bit printer port environment *)
          BEGIN
          (* new character set? *)
          IF (DISPLAY[Y,X].CHARACTER_SET <> P_GL_CHARACTER_SET) AND
             (DISPLAY[Y,X].CHARACTER_SET <> P_GR_CHARACTER_SET) THEN
            BEGIN
            (* yes, designate and invoke as needed *)
            CASE DISPLAY[Y,X].CHARACTER_SET OF

              DESIGNATED_GRAPHIC_SETS[G0]:
                BEGIN
                (* character set is in G0 *)
                (* send LS0 to invoke G0 to GL *)
                WRITE (PRINTER_PORT, '<SI>');
                P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
                END;

              DESIGNATED_GRAPHIC_SETS[G1]:
                BEGIN
                (* character set is in G1 *)
                (* send LS1 to invoke G1 to GL *)
                WRITE (PRINTER_PORT, '<SO>');
                P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
                END;

              DESIGNATED_GRAPHIC_SETS[G2]:
                BEGIN
                (* character set is in G2 *)
                (* send LS2R to invoke G2 to GR *)
                WRITE (PRINTER_PORT, '<ESC>}');
                P_GR_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
                END;

              DESIGNATED_GRAPHIC_SETS[G3]:
                BEGIN
                (* character set is in G3 *)
                (* send LS3R to invoke G3 to GR *)
```

```
                WRITE (PRINTER_PORT, '<ESC>|');
                P_GR_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
                END;

              OTHERWISE:
                BEGIN
                (* no fallback for print all characters *)
                (* designate the displayed set to G3 *)
                IF CHARACTER_SET_SIZE[DISPLAY[Y,X].CHARACTER_SET] = 94 THEN
                  WRITE (PRINTER_PORT, '<ESC>+');
                ELSE
                  WRITE (PRINTER_PORT, '<ESC>/');
                WRITE (PRINTER_PORT,
                       CHARACTER_SET_DSCS[DISPLAY[Y,X].CHARACTER_SET]);
                P_DESIGNATED_CHARACTER_SETS[G3]:=
                   DISPLAY[Y,X].CHARACTER_SET;
                (* send LS3R to invoke G3 to GR *)
                WRITE (PRINTER_PORT, '<ESC>|');
                P_GR_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
                END;
              END  (* end case of display character set *)

            END;  (* end new character set *)
          END;  (* end 8-bit printer port environment *)


        (* send character code to printer *)
        WRITE (PRINTER_PORT, CHR(DISPLAY[Y,X].CODE));
        END;  (* end for each character in the current line *)
      WRITE (PRINTER_PORT,'<CR><LF>');
      END;  (* end for each line *)
    IF PRINT_FORM_FEED_MODE = PRINT_FF_ON THEN
      WRITE (PRINTER_PORT,'<FF>');

    (* termination sequence *)
    (* send LS0 if necessary to leave G0 invoked to GL *)
    IF P_GL_CHARACTER_SET <> DESIGNATED_GRAPHIC_SETS[G0] THEN
      WRITE (PRINTER_PORT, '<SI>');
    (* if 8-bit printer port environment *)
    IF PRINTER_PORT_ENVIRONMENT = EIGHT_BIT THEN
      BEGIN
      (* send LS2R if necessary to leave G2 invoked to GR *)
      IF P_GR_CHARACTER_SET <> DESIGNATED_GRAPHIC_SETS[G2] THEN
      WRITE (PRINTER_PORT, '<ESC>}');
      END;
    (* if printer G3 isn't same as terminal's, re-designate it *)
    IF P_DESIGNATED_GRAPHIC_SETS[G3] <> DESIGNATED_GRAPHIC_SETS[G3] THEN
      BEGIN
      IF CHARACTER_SET_SIZE[DESIGNATED_GRAPHIC_SETS[G3]] = 94 THEN
        WRITE (PRINTER_PORT, '<ESC>+');
      ELSE
        WRITE (PRINTER_PORT, '<ESC>/');
      WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G3]]);
```

```
    END;

  END;  (* end printer style is All Characters *)

END;  (* end procedure PRINT_PAGE *)

(*
 * Send Line Attributes
 *)
PROCEDURE SEND_LINE_ATTRIBUTES (P_LINE_RENDITION: LINE_RENDITION_TYPE);

BEGIN
(* set line rendition *)
IF P_LINE_RENDITION = SINGLE_WIDTH THEN
  WRITE (PRINTER_PORT,'<ESC>#5');
IF P_LINE_RENDITION = DOUBLE_WIDTH THEN
  WRITE (PRINTER_PORT,'<ESC>#6');
IF P_LINE_RENDITION = DOUBLE_HEIGHT_TOP THEN
  WRITE (PRINTER_PORT,'<ESC>#3');
IF P_LINE_RENDITION = DOUBLE_HEIGHT_BOTTOM THEN
  WRITE (PRINTER_PORT,'<ESC>#4');
END;  (* end procedure SEND_LINE_ATTRIBUTES *)

(*
 * Send Graphic Rendition
 *   This procedure sends a new SGR to the printer port
 *   if the graphic rendition has changed.
 *)
PROCEDURE SEND_GRAPHIC_RENDITION
    (CURRENT_RENDITION: CHARACTER_RENDITION_TYPE;
     NEW_RENDITION:     CHARACTER_RENDITION_TYPE);
BEGIN
(* new graphic rendition? *)
IF (CURRENT_RENDITION[BOLD]       <> NEW_RENDITION[BOLD])       OR
   (CURRENT_RENDITION[UNDERSCORE] <> NEW_RENDITION[UNDERSCORE]) OR
   (CURRENT_RENDITION[BLINK]      <> NEW_RENDITION[BLINK])      OR
   (CURRENT_RENDITION[REVERSE]    <> NEW_RENDITION[REVERSE])    THEN
  BEGIN
  (* yes, send new SGR to printer *)
  WRITE (PRINTER_PORT,'<ESC>[0');
  IF NEW_RENDITION[BOLD] THEN
    WRITE (PRINTER_PORT,';1');
  IF NEW_RENDITION[UNDERSCORE] THEN
    WRITE (PRINTER_PORT,';4');
  IF NEW_RENDITION[BLINK] THEN
    WRITE (PRINTER_PORT,';5');
  IF NEW_RENDITION[REVERSE] THEN
    WRITE (PRINTER_PORT,';7');
  WRITE (PRINTER_PORT,'m');
  END;
END;  (* end procedure SEND_GRAPHIC_RENDITION *)
```

Known Deviations: None
(algorithm does not include code for handling fallback)

PRINT LINE MC

Levels 1X, 2X, 3X

Purpose: Transmit the line containing the Active Position to the printer port.

Format:	CSI	?	1	i
	9/11	3/15	3/1	6/9

Description: The MC control with private parameter (indicated by 3/15 "?") of 3/1 (one) causes a print function to be performed, in which the line containing the Active Position is transmitted to the printer port. b2 State Affected: None

Algorithm:

```
PROCEDURE PRINT_LINE;
VAR     Y: LINE_TYPE;
        X: COLUMN_TYPE;
        P_END_OF_LINE: INTEGER;
        P_LINE_RENDITION: LINE_RENDITION_TYPE;
        P_RENDITION: CHARACTER_RENDITION_TYPE;
        P_GL_CHARACTER_SET: GRAPHICS_CHARACTER_SET_TYPE;
        P_GR_CHARACTER_SET: GRAPHICS_CHARACTER_SET_TYPE;
        P_CURRENT_NRCS:  GRAPHICS_CHARACTER_SET_TYPE;
        P_G1_DESIGNATE:  GRAPHICS_CHARACTER_SET_TYPE;

(* determine current National character set *)
P_CURRENT_NRCS:= ASCII_G;
IF (CHARACTER_SET_MODE = SEVEN_BITS) AND
   (KEYBOARD_USAGE_MODE = TYPEWRITER) THEN
  P_CURRENT_NRCS:= NRCS_TABLE(KEYBOARD_DIALECT);

(* handle each printer style separately *)
IF PRINTER_STYLE = NATIONAL_ONLY THEN
  BEGIN  (* Print National Only *)
  (* initialization *)
  P_GL_CHARACTER_SET:= P_CURRENT_NRCS;
  (* transmit data as characters from GL *)

  (* transmit current line *)
  Y = ACTIVE_POSITION.LINE;
  (* set end of line to skip trailing spaces *)
  P_END_OF_LINE = END_OF_LINE(Y);
  WHILE (DISPLAY[Y,P_END_OF_LINE].CODE = SPACE_CODE) AND
        (P_END_OF_LINE > 0)  DO
    P_END_OF_LINE:= P_END_OF_LINE - 1;
  (* for each character in the current line *)
  FOR X:= 1 TO P_END_OF_LINE DO
    BEGIN
```

```
      (* if not in current NRCS *)
      IF DISPLAY[Y,X].CHARACTER_SET <> P_GL_CHARACTER_SET THEN
        BEGIN
        (* do fallback - to be specified *)
        END
      ELSE
        (* otherwise send character code to printer *)
        WRITE (PRINTER_PORT, CHR(DISPLAY[Y,X].CODE));
      END; (* end for each character in line *)
    WRITE (PRINTER_PORT,'<CR><LF>');
    (* no termination sequence required *)
    END;  (* end National Only *)

  ELSE IF PRINTER_STYLE = NATIONAL_PLUS_LINE_DRAWING THEN
    BEGIN
    (* Printer Style is National + Line Drawing *)
    (* initialization sequence *)
    P_RENDITION[BOLD]:= FALSE;
    P_RENDITION[UNDERSCORE]:= FALSE;
    P_RENDITION[BLINK]:= FALSE;
    P_RENDITION[REVERSE]:= FALSE;
    WRITE (PRINTER_PORT,'<CSI>0m');
    P_GL_CHARACTER_SET:= P_CURRENT_NRCS;
    (* designate current NRCS as G0
       ASCII is NRCS for North America *)
    WRITE (PRINTER_PORT, '<ESC>(');
    WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[P_GL_CHARACTER_SET]);
    (* designate line drawing as G1 *)
    WRITE (PRINTER_PORT, '<ESC>)0');
    (* invoke G0 to GL *)
    WRITE (PRINTER_PORT, '<SI>');

    (* data transmission *)
    Y = ACTIVE_POSITION.LINE;
    (* set line rendition *)
    SEND_LINE_ATTRIBUTES(LINE_RENDITION[Y]);
    (* set end of line to skip trailing spaces
       with "normal" graphic rendition *)
    P_END_OF_LINE = END_OF_LINE(Y);
    WHILE (DISPLAY[Y,P_END_OF_LINE].CODE = SPACE_CODE) AND
          (P_END_OF_LINE > 0) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[BOLD]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[UNDERSCORE]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[BLINK]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[REVERSE]) DO
      P_END_OF_LINE:= P_END_OF_LINE - 1;
    (* for each character in the current line *)
    FOR X:= 1 TO P_END_OF_LINE DO
      BEGIN
      (* set graphic rendition if it has changed *)
      SEND_GRAPHIC_RENDITION (P_RENDITION, DISPLAY[X,Y].RENDITION);
      P_RENDITION[BOLD]:= DISPLAY[Y,X].RENDITION[BOLD];
      P_RENDITION[UNDERSCORE]:=
```

```
    DISPLAY[Y,X].RENDITION[UNDERSCORE];
  P_RENDITION[BLINK]:= DISPLAY[Y,X].RENDITION[BLINK];
  P_RENDITION[REVERSE]:=
    DISPLAY[Y,X].RENDITION[REVERSE];

  (* new character set? *)
  IF DISPLAY[Y,X].CHARACTER_SET <> P_GL_CHARACTER_SET THEN
    BEGIN
    (* yes, send SCS sequence to printer *)
    CASE DISPLAY[Y,X].CHARACTER_SET OF

      P_CURRENT_NRCS:
        BEGIN
        (* character set is current NRCS *)
        (* invoke G0 to GL *)
        WRITE (PRINTER_PORT, '<SI>');
        P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
        END;

      LINE_DRAWING:
        BEGIN
        (* character set is Line Drawing *)
        (* is Line Drawing in G1? *)
        IF P_G1_DESIGNATE <> LINE_DRAWING THEN
          BEGIN
          (* no, re-designate it *)
          WRITE (PRINTER_PORT, '<ESC>)0');
          P_G1_DESIGNATE:= LINE_DRAWING;
          END;
        (* invoke G1 to GL *)
        WRITE (PRINTER_PORT,'<SO>');
        P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
        END;

      ASCII_G:
        (* character set is ASCII *)
        (* is ASCII current NRCS? *)
        IF P_CURRENT_NRCS <> ASCII_G THEN
          BEGIN
          (* no, if it's not already in G1, designate it *)
          IF P_G1_DESIGNATE <> ASCII_G THEN
            BEGIN
            WRITE (PRINTER_PORT, '<ESC>)B');
            P_G1_DESIGNATE:= ASCII_G;
            END;
          (* invoke G1 to GL *)
          WRITE (PRINTER_PORT, '<SO>');
          P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
          END;
      OTHERWISE: IGNORE;
      END;  (* end case of display character set *)
    END;  (* end new character set *)
  (* do we need to fallback? *)
```

```
    IF (DISPLAY[X,Y].CHARACTER_SET <> P_CURRENT_NRCS) AND
       (DISPLAY[X,Y].CHARACTER_SET <> LINE_DRAWING) AND
       (DISPLAY[X,Y].CHARACTER_SET <> ASCII_G) THEN
      BEGIN
      (* do fallback - to be specified *)
      END
    ELSE
      (* no, just send character code to printer *)
      WRITE (PRINTER_PORT, CHR(DISPLAY[Y,X].CODE));
    END;  (* end for each character in the current line *)
  WRITE (PRINTER_PORT,'<CR><LF>');

  (* termination sequence *)
  (* send LS0 if necessary to leave G0 invoked to GL *)
  IF P_GL_CHARACTER_SET <> P_CURRENT_NRCS THEN
    WRITE (PRINTER_PORT, '<SI>');
  (* if Line Drawing isn't in G1, designate it *)
  IF P_G1_DESIGNATE <> LINE_DRAWING THEN
    WRITE (PRINTER_PORT, '<ESC>)0');

  END;  (* end printer style is National + Line Drawing *)

ELSE IF PRINTER_STYLE = ALL_CHARACTERS THEN
  BEGIN
  (* printer style is All Characters *)
  (* initialization sequence *)
  P_RENDITION[BOLD]:= FALSE;
  P_RENDITION[UNDERSCORE]:= FALSE;
  P_RENDITION[BLINK]:= FALSE;
  P_RENDITION[REVERSE]:= FALSE;
  WRITE (PRINTER_PORT,'<CSI>0m');

  (* initialize printer G sets to correspond with terminal *)
  WRITE (PRINTER_PORT, '<ESC>(');
  WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G0]]);

  IF CHARACTER_SET_SIZE[DESIGNATED_GRAPHIC_SETS[G1]] = 94 THEN
    WRITE (PRINTER_PORT, '<ESC>)');
  ELSE
    WRITE (PRINTER_PORT, '<ESC>-');
  WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G1]]);

  IF CHARACTER_SET_SIZE[DESIGNATED_GRAPHIC_SETS[G2]] = 94 THEN
    WRITE (PRINTER_PORT, '<ESC>*');
  ELSE
    WRITE (PRINTER_PORT, '<ESC>.');
  WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G2]]);

  IF CHARACTER_SET_SIZE[DESIGNATED_GRAPHIC_SETS[G3]] = 94 THEN
    WRITE (PRINTER_PORT, '<ESC>+');
  ELSE
    WRITE (PRINTER_PORT, '<ESC>/');
  WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G3]]);
```

```
    (* send LS0 to invoke G0 to GL *)
    WRITE (PRINTER_PORT, '<SI>');
    P_GL_CHARACTER_SET:= DESIGNATED_GRAPHIC_SETS[G0];
    (* if 8-bit printer port environment *)
    IF PRINTER_PORT_ENVIRONMENT = EIGHT_BIT THEN
      BEGIN
      (* send LS2R to invoke G2 to GR *)
      WRITE (PRINTER_PORT, '<ESC>}');
      P_GR_CHARACTER_SET:= DESIGNATED_GRAPHIC_SETS[G2];
      END;

    (* data transmission *)
    Y = ACTIVE_POSITION.LINE;
    (* set line rendition *)
    SEND_LINE_ATTRIBUTES(LINE_RENDITION[Y]);
    (* set end of line to skip trailing spaces
       with "normal" graphic rendition *)
    P_END_OF_LINE = END_OF_LINE(Y);
    WHILE (DISPLAY[Y,P_END_OF_LINE].CODE = SPACE_CODE) AND
          (P_END_OF_LINE > 0) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[BOLD]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[UNDERSCORE]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[BLINK]) AND
          (NOT DISPLAY[Y,P_END_OF_LINE].RENDITION[REVERSE]) DO
      P_END_OF_LINE:= P_END_OF_LINE - 1;
    (* for each character in the current line *)
    FOR X:= 1 TO P_END_OF_LINE DO
      BEGIN
      (* set graphic rendition *)
      SEND_GRAPHIC_RENDITION (P_RENDITION, DISPLAY[X,Y].RENDITION);
      P_RENDITION[BOLD]:= DISPLAY[Y,X].RENDITION[BOLD];
      P_RENDITION[UNDERSCORE]:=
        DISPLAY[Y,X].RENDITION[UNDERSCORE];
      P_RENDITION[BLINK]:= DISPLAY[Y,X].RENDITION[BLINK];
      P_RENDITION[REVERSE]:=
        DISPLAY[Y,X].RENDITION[REVERSE];

      (* handle 7-bit/8-bit printer port environment separately *)
      (* if 7-bit printer port environment *)
      IF PRINTER_PORT_ENVIRONMENT = SEVEN_BIT THEN
        BEGIN
        (* new character set? *)
        IF DISPLAY[Y,X].CHARACTER_SET <> P_GL_CHARACTER_SET THEN
          BEGIN
          (* yes, designate and invoke as needed *)
          CASE DISPLAY[Y,X].CHARACTER_SET OF

            DESIGNATED_GRAPHIC_SETS[G0]:
              (* character set is in G0 *)
              (* send LS0 to invoke G0 to GL *)
              WRITE (PRINTER_PORT, '<SI>');
```

```
            DESIGNATED_GRAPHIC_SETS[G1]:
              (* character set is in G1'*)
              (* send LS1 to invoke G1 to GL *)
              WRITE (PRINTER_PORT, '<SO>');

            DESIGNATED_GRAPHIC_SETS[G2]:
              (* character set is in G2 *)
              (* send SS2 to invoke G2 to GL *)
              WRITE (PRINTER_PORT, '<ESC>N');

            DESIGNATED_GRAPHIC_SETS[G3]:
              (* character set is in G3 *)
              (* send SS3 to invoke G3 to GL *)
              WRITE (PRINTER_PORT, '<ESC>O');

            OTHERWISE:
              BEGIN
              (* no fallback for print all characters *)
              (* designate the displayed set as G3 *)
              IF CHARACTER_SET_SIZE[DISPLAY[Y,X].CHARACTER_SET] = 94 THEN
                WRITE (PRINTER_PORT, '<ESC>+');
              ELSE
                WRITE (PRINTER_PORT, '<ESC>/');
              WRITE (PRINTER_PORT, CHARACTER_SET_DSCS
              [DISPLAY[Y,X].CHARACTER_SET]);
              P_DESIGNATED_CHARACTER_SETS[G3]:=
                 DISPLAY[Y,X].CHARACTER_SET;
              (* send SS3 to invoke G3 to GL *)
              WRITE (PRINTER_PORT, '<ESC>O');
              END;
            END;  (* end case of... *)

          P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
          END;  (* end new character set *)
        END  (* end 7-bit printer port environment *)

      ELSE
        (* 8-bit printer port environment *)
        BEGIN
        (* new character set? *)
        IF (DISPLAY[Y,X].CHARACTER_SET <> P_GL_CHARACTER_SET) AND
           (DISPLAY[Y,X].CHARACTER_SET <> P_GR_CHARACTER_SET) THEN
          BEGIN
          (* yes, designate and invoke as needed *)
          CASE DISPLAY[Y,X].CHARACTER_SET OF

            DESIGNATED_GRAPHIC_SETS[G0]:
              BEGIN
              (* character set is in G0 *)
              (* send LS0 to invoke G0 to GL *)
              WRITE (PRINTER_PORT, '<SI>');
              P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
              END;
```

```
            DESIGNATED_GRAPHIC_SETS[G1]:
              BEGIN
              (* character set is in G1 *)
              (* send LS1 to invoke G1 to GL *)
              WRITE (PRINTER_PORT, '<SO>');
              P_GL_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
              END;

            DESIGNATED_GRAPHIC_SETS[G2]:
              BEGIN
              (* character set is in G2 *)
              (* send LS2R to invoke G2 to GR *)
              WRITE (PRINTER_PORT, '<ESC>}');
              P_GR_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
              END;

            DESIGNATED_GRAPHIC_SETS[G3]:
              BEGIN
              (* character set is in G3 *)
              (* send LS3R to invoke G3 to GR *)
              WRITE (PRINTER_PORT, '<ESC>|');
              P_GR_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
              END;

            OTHERWISE:
              BEGIN
              (* no fallback for print all characters *)
              (* designate the displayed set to G3 *)
              IF CHARACTER_SET_SIZE[DISPLAY[Y,X].CHARACTER_SET] = 94 THEN
                WRITE (PRINTER_PORT, '<ESC>+');
              ELSE
                WRITE (PRINTER_PORT, '<ESC>/');
              WRITE (PRINTER_PORT,
                     CHARACTER_SET_DSCS[DISPLAY[Y,X].CHARACTER_SET]);
              P_DESIGNATED_CHARACTER_SETS[G3]:=
                 DISPLAY[Y,X].CHARACTER_SET;
              (* send LS3R to invoke G3 to GR *)
              WRITE (PRINTER_PORT, '<ESC>|');
              P_GR_CHARACTER_SET:= DISPLAY[Y,X].CHARACTER_SET;
              END;
            END  (* end case of... *)

          END;  (* end new character set *)
        END;  (* end 8-bit printer port environment *)

      (* send character code to printer *)
      WRITE (PRINTER_PORT, CHR(DISPLAY[Y,X].CODE));
      END;  (* end for each character in the current line *)
    WRITE (PRINTER_PORT,'<CR><LF>');

    (* termination sequence *)
    (* send LS0 if necessary to leave G0 invoked to GL *)
```

```
    IF P_GL_CHARACTER_SET <> DESIGNATED_GRAPHIC_SETS[G0] THEN
      WRITE (PRINTER_PORT, '<SI>');
    (* if 8-bit printer port environment *)
    IF PRINTER_PORT_ENVIRONMENT = EIGHT_BIT THEN
      BEGIN
      (* send LS2R if necessary to leave G2 invoked to GR *)
      IF P_GR_CHARACTER_SET <> DESIGNATED_GRAPHIC_SETS[G2] THEN
      WRITE (PRINTER_PORT, '<ESC>}');
      END;
    (* if printer G3 isn't same as terminal's, re-designate it *)
    IF P_DESIGNATED_GRAPHIC_SETS[G3] <> DESIGNATED_GRAPHIC_SETS[G3] THEN
      BEGIN
      IF CHARACTER_SET_SIZE[DESIGNATED_GRAPHIC_SETS[G3]] = 94 THEN
        WRITE (PRINTER_PORT, '<ESC>+');
      ELSE
        WRITE (PRINTER_PORT, '<ESC>/');
      WRITE (PRINTER_PORT, CHARACTER_SET_DSCS[DESIGNATED_GRAPHIC_SETS[G3]]);
      END;

    END;  (* end printer style is All Characters *)

  END;  (* end procedure PRINT_LINE *)

  (*
   * Send Line Attributes
   *)
  PROCEDURE SEND_LINE_ATTRIBUTES (P_LINE_RENDITION: LINE_RENDITION_TYPE);

  BEGIN
  (* set line rendition *)
  IF P_LINE_RENDITION = SINGLE_WIDTH THEN
    WRITE (PRINTER_PORT,'<ESC>#5');
  IF P_LINE_RENDITION = DOUBLE_WIDTH THEN
    WRITE (PRINTER_PORT,'<ESC>#6');
  IF P_LINE_RENDITION = DOUBLE_HEIGHT_TOP THEN
    WRITE (PRINTER_PORT,'<ESC>#3');
  IF P_LINE_RENDITION = DOUBLE_HEIGHT_BOTTOM THEN
    WRITE (PRINTER_PORT,'<ESC>#4');
  END;  (* end procedure SEND_LINE_ATTRIBUTES *)

  (*
   * Send Graphic Rendition
   *   This procedure sends a new SGR to the printer port
   *   if the graphic rendition has changed.
   *)
  PROCEDURE SEND_GRAPHIC_RENDITION
      (CURRENT_RENDITION: CHARACTER_RENDITION_TYPE;
       NEW_RENDITION:     CHARACTER_RENDITION_TYPE);
  BEGIN
  (* new graphic rendition? *)
  IF (CURRENT_RENDITION[BOLD]       <> NEW_RENDTION[BOLD])       OR
     (CURRENT_RENDITION[UNDERSCORE] <> NEW_RENDITION[UNDERSCORE]) OR
     (CURRENT_RENDITION[BLINK]      <> NEW_RENDITION[BLINK])      OR
```

```
    (CURRENT_RENDITION[REVERSE]    <> NEW_RENDITION[REVERSE])    THEN
   BEGIN
   (* yes, send new SGR to printer *)
   WRITE (PRINTER_PORT,'<ESC>[0');
   IF NEW_RENDITION[BOLD] THEN
     WRITE (PRINTER_PORT,';1');
   IF NEW_RENDITION[UNDERSCORE] THEN
     WRITE (PRINTER_PORT,';4');
   IF NEW_RENDITION[BLINK] THEN
     WRITE (PRINTER_PORT,';5');
   IF NEW_RENDITION[REVERSE] THEN
     WRITE (PRINTER_PORT,';7');
   WRITE (PRINTER_PORT,'m');
   END;
END;  (* end procedure SEND_GRAPHIC_RENDITION *)
```

Known Deviations: None
(algorithm does not include code for handling fallback)

7.8 GRAPHICS PRINTING

This section describes graphics printing and is intended to apply to devices which support graphic displays and the Sixel Graphics Extension.

7.8.1 Graphics Print Operations

7.8.1.1 Graphics Print Screen

The complete graphics bit map is transferred to the printer. This can be invoked by the host through the ReGIS hardcopy command or by the user under local control.

7.8.1.2 Graphics Print Region

An image of a portion of the screen bit map is transferred to the printer. This can be invoked by the host through the ReGIS hardcopy command. Refer to the ReGIS chapter for determining how to define this region.

7.8.2 Sixel Printing

A Sixel is a group of 6 vertical pixels represented by 6 bits within a character code. A one value for a bit indicates that a pixel-spot will be placed at the corresponding grid position, a zero will cause the corresponding grid position to be unchanged (not written).

Sixel printing consists of setting context and attributes for the pixels and then printing each Sixel in left-to-right, top-to-bottom order. Wherever possible, run-length encoding is used to reduce the number of characters transmitted for a line of Sixels.

See the Sixel Graphics Extension chapter for further details on the Sixel format.

Guideline: When a Sixel dump is initiated by the user under local control (Print Key), the terminal first transmits a CR to reset the ANSI text position on the printer to the left margin. If the Sixel dump is initiated by the host, the CR is not sent so the host can initialize the starting sixel position.

Guideline: The VT240 sends the following characters out the printer port (or host port) when transmitting a sixel dump of the screen.

ESC	\	<CR>	ESC	P	1	q	<sixel-data>	ESC	\
1/11	5/12	0/13	1/11	5/0	3/1	7/1		1/11	5/12
	ST			DCS					ST

7.8.2.1 Color And Monochrome Sixels

On devices which support a color display, the user may select monochrome or color sixel printing. Monochrome Sixels are formed from the multiple plane image by a logical OR of all pixels which are selected for any color. If color printing is selected, The terminal will send out color specifiers at the beginning of each Sixel dump to reflect the contents of the terminal's color map.

If PRINT BACKGROUND is selected, all pixels with no bits set in any plane are printed in the color value selected for index 0.

7.8.2.2 Expanded Print

Graphics expanded print mode determines whether the terminal generates a small (compressed) or large (expanded) graphics image. The small image fits on 8-1/2 inch wide paper and expanded images on 13 inch wide paper in portrait mode. This mode is selectable by the DEC-private mode DECGEPM or by the user under local control.

7.8.2.3 Rotated Print

Graphics rotated print mode rotates the image by 90 degrees so that an expanded image can fit on a single 8-1/2 inch wide page (rotated images are always expanded). This mode is selectable by the DEC-private mode DECGRPM or by the user under local control.

Guideline: the VT240 rotates the image counter clockwise so that the left side of the paper (as it comes out of a typical dot matrix printer) corresponds to the top of the image on the terminal screen. This scanning order was chosen to allow punching holes for a looseleaf notebook on the left side of the page as it comes out of the printer.

7.8.2.4 Sixel Graphics Level (Guideline)

As the Sixel graphics protocol has become more general to accommodate a range of pixel aspect ratios and grid sizes, Terminals have begun supporting two levels of the Sixel graphics protocol to match the capabilities of the printer being used. Level 1 provides backward compatibility with older printers, while Level 2 allows the terminal to take full advantage newer printer capabilities to improve print quality.

The Sixel Graphics Level determines how the terminal matches the printer's Sixel Aspect Ratio, Horizontal Grid Size, Background Printing, and Color Printing capabilities.

The Sixel Graphics Level is selectable by the user under local control.

Level 1 Sixel Devices -

ASSUMPTIONS: Level 1 sixel devices do not support the Set Raster Attribute command, Background Select, Horizontal Grid Size, or Macro Parameter commands. The aspect ratio is fixed at 2:1, and the horizontal grid size is approximately 7.5 x .001 inches (800 pixels require 6 inches). Level 1 is the factory default.

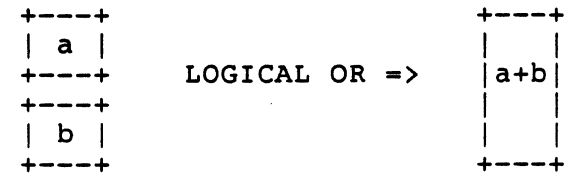
Sixel Control Strings will be sent to the printer as follows:

ESC P 1 q S...S ESC \ (always 7-bit controls)

COMPRESSED Sixel Print Option -

A terminal with a 1:1 pixel aspect ratio will combine each pair of horizontally adjacent pixels (from two successive scan lines) into a single pixel. The value of the combined pixel will be the LOGICAL OR of the values of the individual pixels.

1:1 Aspect Ratio Level 1 Printer

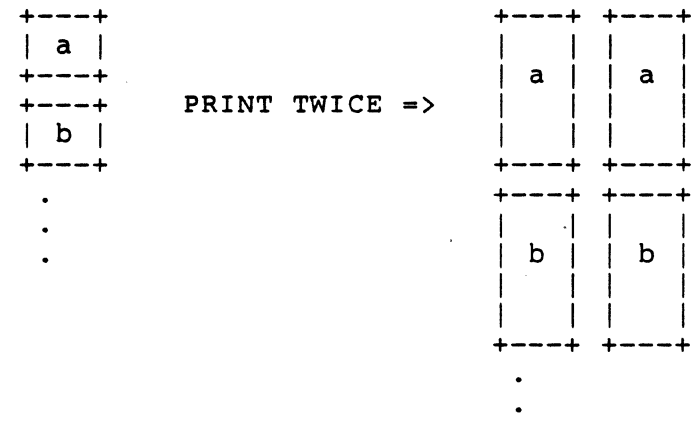


This will produce images of the same size and aspect ratio as the VT240 (2:1 Aspect Ratio).
ReGIS images will appear normal, but ANSI text may be distorted.

EXPANDED Sixel Print Option -

Terminals with pixel aspect ratios of 1:1 will transmit each Sixel twice (in immediate horizontal succession).

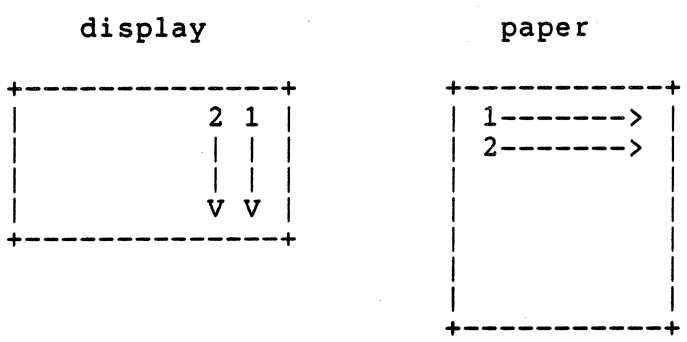
1:1 Aspect Ratio Level 1 Printer



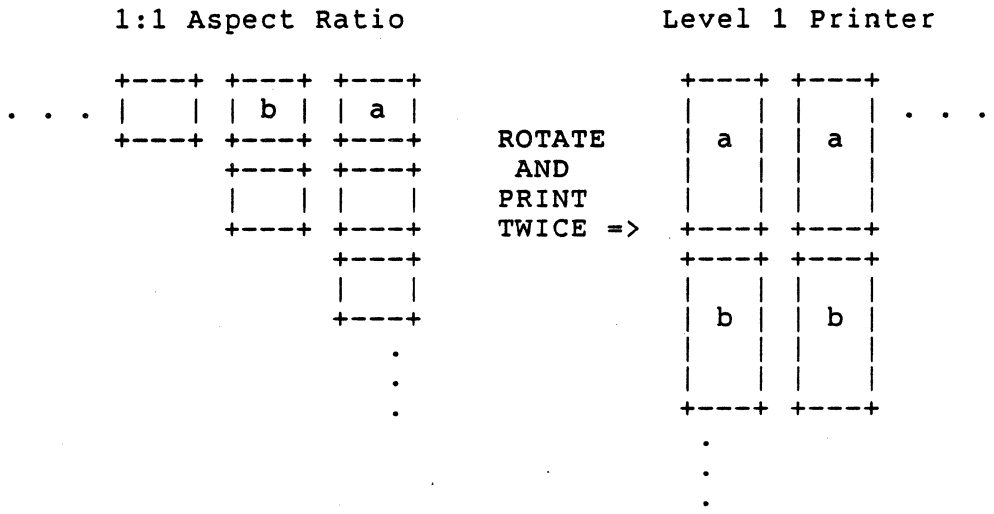
This will produce images of the same size and aspect ratio as the VT240 (2:1 Aspect Ratio), but with twice the vertical resolution.

ROTATED Sixel Print Option -

Terminals with 1:1 pixel aspect ratios will transmit the selected portion of the display bitmap in vertical strips six pixels wide from top to bottom, and right to left.



Each Sixel is sent twice in immediate horizontal succession (with respect to the printer).



This will produce images of the same size and aspect ratio as the VT240(2:1 Aspect Ratio), but with twice the vertical resolution (with respect to the image displayed on the screen).

Level 2 Sixel Devices -

ASSUMPTIONS: Level 2 sixel devices support the Set Raster Attribute command, Background Select, Horizontal Grid Size and Macro Parameter commands.

Sixel control strings are sent as follows:

```
  ESC P  Ps1 ; Ps2 ; Pn3 q " Pn4 ; Pn5 ; Pn6 ; Pn7 ******  ESC \
  \___/  \_________________/ \_____________________________/ \____/  \___/
   DCS   Protocol Selector     Raster Attributes            Picture  ST
                                                             data
  \_____________________________/ \_____________________________/
  DCS Introducer Sequence          sixel data
```

Where:

- Ps1 Is the Macro Parameter and is always ZERO.
- Ps2 Background Select
  1 if Background Printing is disabled in Set-Up
  2 if Background Printing is enabled in Set-Up
- Pn3 Horizontal Grid Size, given in units specified by ANSI SSU (default is decipoints, 1/720 inch). For default size units, the grid size should be 6 for COMPRESSED and 9 for EXPANDED or ROTATED print. Since the host can change the printer between accesses, SSU should be sent once before each sixel dump.

```
  ESC  [    2   SP  I     (Set Size Unit
  1/11 5/11 3/2 2/0 4/9    to Decipoints)
```

- Pn4 Pixel aspect ratio numerator, 1
- Pn5 Pixel aspect ratio denominator, 1
- Pn6 Horizontal extent
  (number of pixels in image horizontally)
- Pn7 Vertical extent
  (number of pixels in image vertically)

COMPRESSED Sixel Print Option -

When Level 2 sixel graphics is selected, the terminal will transmit the selected portion of the display bitmap as sixels from left to right, and top to bottom. No translation of screen pixels is required since the printer directly supports 1:1 sixels with the desired grid size. The horizontal grid size is set to 6 in the Set Raster Attributes command.

EXPANDED Sixel Print Option -

When Level 2 sixel graphics is selected, the terminal will transmit the selected portion of the display bitmap as sixels from left to right, and top to bottom. No translation of screen pixels is required since the printer directly supports 1:1 sixels with the desired grid size. The horizontal grid size is set to 9 in the Set Raster Attributes command.

ROTATED Sixel Print Option -

When Level 2 sixel graphics is selected, the terminal will transmit the selected portion of the display bitmap in vertical strips six pixels wide from top to bottom, and right to left. Each sixel is sent once using the same sixel initiator as for expanded print above (horizontal grid size set to 9).

7.9 CHANGE HISTORY

7.9.1 Revision 0.0 To AX10

1. Added Terminology section
2. Made extensive changes to the transmission characteristics descriptions.
3. Added notes that the state of Auto Print Mode is UNDEFINED across changes in Printer Controller Mode.
4. Completely rewrote the algorithms for Print Line and Print Screen.
5. Added note to the abstract indicating that this standard currently only applies to Level 2 conformance.
6. Added a note to Printer Controller Mode indicating that 7/15 and 15/15 are UNDEFINED when Printer Controller Mode is set.

7.9.2 Rev AX10 To AX11

1. Changed name of character rendition from REVERSE_VIDEO to REVERSE throughout code to be more consistent.

7.9.3 Rev AX11 To AX12

1. Removed all previous change bars. Added change bars to any change that could affect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.
2. Defined Level 3 Printer Port Extension to document how multiple sessions and off screen memory interact with the Printer Port. Added two new Device Status Reports (Printer busy, Printer not assigned). Added definitions for "Virtual Terminal" and "Session". Extended description of Printer Port status.

3. Renamed former "Print Screen" to "Print Page" (prints page from display memory including parts which may not be visible on the screen).

4. Defined "Print Composed Main Display" or new "Print Screen" which prints what is displayed on the screen (may include data from multiple sessions). New Print Screen is not affected by Printer Extent Mode.

5. Updated Print Form Feed Mode to affect both "Print Page" and "Print Screen".

6. Extended description of XON/XOFF flow control (more complete spec). Added section on hardware flow control (DSR/DTR).

7. In printer controller mode, 7-bit C1 controls are never translated to 8-bits.

8. Added description of "Local Controller Mode" to Print Modes (as in VT240 and VT300).

9. Updated Printer Style modes. Documented support for User Preference Supplemental Set (UPSS according to TIA DEC to ISO 8-bit transition plan). Changed "ASCII only" to "National Only". In Print National + Line Drawing, noted how ASCII can be designated to G1 for characters not in current NRC (as in VT220). Modified "Print All Characters" mode to print all characters efficiently without fallback.

10. Added table of NRC sets and designating sequences.

11. Information on transmitting display attributes grouped into one section to avoid duplication. Added material on invisible character attribute and trailing spaces.

12. All fallback tables grouped into one section eliminating duplication. Fallbacks organized by character set. Added fallbacks for DEC Technical, Control Representation Mode (CRM as in VT300), and NRCS characters not in DEC MCS or ISO Latin Alphabet Nr 1. Added sections clarifying DRCS and UPSS fallbacks.

13. Added section describing print operations separate from control functions.

14. Extended control function descriptions for Level 3. Updated algorithms to handle 96 character sets and new Print All Characters functionality.

15. Added entirely new Graphics Print section which describes Sixel printing on terminals.

Section Index

DEC STD 070-8 VIDEO & PRINTER STANDARDS REFERENCE MANUAL - ReGIS GRAPHICS EXTENSION

Document Identifier: A-DS-EL00070-08-0 Rev A, 31-Mar-1988

ABSTRACT: This standard defines the REmote Graphics Instruction Set (ReGIS). ReGIS is a set of graphics execution instructions designed primarily for graphics terminals operated over low bandwidth communications paths, such as serial lines.

APPLICABILITY: Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using or emulating terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria."

STATUS: APPROVED 31-Mar-1988; see EL-INDEX-00 for expiration date.

The material contained within this document is assumed to define mandatory standards unless it is clearly marked as (a) not mandatory or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques that are considered to be good practice, but should not be considered as requirements. The procedures for modifying or evolving this standard are contained within the contents of this document.

| TITLE: DEC STD 070-8 VIDEO & PRINTER STANDARDS REFERENCE MANUAL - ReGIS GRAPHICS EXTENSION

| DOCUMENT IDENTIFIER: A-DS-EL00070-08-0 Rev A, 31-Mar-1988

REVISION HISTORY:

Original Draft	15-Oct-1982
Revision 0.1	25-Dec-1982
Revision AX01	28-Feb-1983
Revision AX10	15-Apr-1983
Revision AX11	22-Aug-1984
Revision AX12	08-Dec-1987
Revision A	31-Mar-1988

Document Management Category:	Terminal Interface Architecture (STI)
Responsible Department:	DSG Terminals Architecture
Responsible Person:	Peter Sichel
Author:	Tom Powers
SMC Writer:	Patricia Winner

APPROVAL: This document, prepared by the Desktop Systems Group, has been reviewed and recommended for approval by the General Review Group for its category for use throughout Digital.

Peter Conklin, Technical Director,
Desktop Systems Group

Peter A. Sichel
Peter Sichel, Standard Owner,
Desktop Systems Group

Eric A Williams 7 July 88
Eric Williams, Standards Process Manager

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, VIDEO::TERMARCH

| Copies of this document can be ordered from Standards and Methods
| Control, CTS1-2/D4, DTN 287-3724, JOKUR::SMC

Please provide your name, badge number, cost center, mailstop, and ENET node when ordering.

CONTENTS

LIST OF FIGURES

8.1 INTRODUCTION

8.1.1 PURPOSE

This standard specifies a data representation for the device independent description of a certain class of imagery called synthetic graphics. "Synthetic graphics" is a term used to describe a mode of drawing in which images are defined in terms of component parts. These parts may be synthesized from geometric entities such as lines, circles, points, and uniform areas. Synthetic graphics differ from "natural imagery" in that "natural images" are usually derived photographically and stored in dot matrix or some other pixel oriented form, and contain only limited explicit information regarding their component content. The REmote Graphic Instruction Set (ReGIS), a generic graphic descriptor, is the system described for executing the graphics commands.

8.1.2 SCOPE

This standard achieves the above purpose by:

1. Postulating a system model based on current technology in which graphics can be supported;

2. Identifying commonly required access points into this model, and;

3. Specifying an execution model and instruction set (ReGIS) consistent with this model.

8.1.3 IMPLEMENTATION OF ReGIS

It is not the intent of this standard to specify what products, hardware or software, must implement ReGIS. The decision to support ReGIS or not will be made by the appropriate marketing and engineering organizations during the initial stages of the Phase Review Process for the product(s) in question. This standard defines the function and form which must be implemented in a product for it to be certified as conforming to ReGIS.

ReGIS can be implemented as either a hardware or a software product; any products which generate or interpret ReGIS are within the scope of this standard.

8.1.4 REFERENCE STANDARDS AND RELATED PUBLICATIONS

DIGITAL STANDARDS

EL-00138-00	DEC STD 138-0 Registry of Control Functions for Character-Imaging Devices
EL-00169-00	DEC STD 169-0 Digital Standard Coded Graphic Character Sets for Hardware and Software

Copies of Digital Standards Can be obtained from Standards and Methods Control, CTS1-2/D4, DTN 287-3724, JOKUR::SMC

Please provide your name, badge number, cost center, mailstop, and ENET node when ordering.

ANSI and ISO STANDARDS

ANSI X3.4 - 1986	American National Standard Code for Information Interchange (ASCII character set)
ANSI X3.41 - 1974	American National Standard Code Extension Techniques for use with the 7-Bit Coded Character Set of the American National Standard Code for Information Interchange
IS2022-1986	Information Processing - ISO 7-bit and 8-bit Coded Character Sets - Code Extension Techniques
(unnumbered)	ISO Register of Coded Character Sets for Use with Escape Sequences
ANSI X3.64 - 1979	Additional Controls for use with American National Standard Code for Information Interchange

ANSI X3.122-1986 (IS8632-1986) Computer Graphics Metafile

ANSI X3.124-1985 (IS7942-1985) Graphical Kernel System (GKS)

Copies of ANSI and ISO Standards and other related publications can be obtained from local Digital Libraries.

"Computer Graphics" vol 13, number 3 (August 1979) - GSPC CORE proposed graphics standard

8.1.5 TERMINOLOGY

Absolute Location - An unsigned coordinate pair that specifies a location based on the screen origin.

Active Position - The character position on the visual display at which the next graphic character would (will) be displayed.

Application Free Primitives - A ReGIS-implemented set of drawing primitives based upon common geometric rule and compass constructions. It contains basic, general constructions only, not specific capabilities for particular applications.

Base Logical Device - See Logical Graphics Device.

Bracketed Pair ([]) - A pair of quantitative values enclosed in brackets; used in ReGIS command strings to encode position values and other position or extent related information.

Closed Curve Sequence - A series of locations ReGIS uses to interpolate a curve whose end points meet. One example is a circle.

Command Keyletter - The ReGIS syntactic element which indicates that a command or command option is to be interpreted.

Complement Writing - Writing mode in which ReGIS complements the image existing on the display with new images being written to the screen.

Current Location - The location in the display space last moved or drawn to.

Display Surface - The active area of the screen; it may be the entire screen.

Echo - Retransmit encoded messages back to the sending device as they are sent.

Erase Writing - Writing operation which overwrites previously drawn objects from the display by using the background (secondary) writing value for both primary and secondary writing values.

Filling - Coloring in a bounded area as it is defined; contrast with "flooding" and "shading".

Flooding - Coloring in, with variable pattern or texture, a bounded area already defined in the display. Supported only by extension to ReGIS.

Graphic Character - A character, other than a control character, that has a visual, displayable representation.

Graphics Cursor - A visible indicator of the current drawing position.

Graphics Pipeline - Graphics environment models are often layered, with higher level layers requesting services of lower levels to effect actions. The route(s) through these layers is called the "graphics pipeline."

Graphic Text - Strings of characters drawn by executing graphics commands; these are distinguished from other types of text by the use of state stored by previous graphics commands.

Gray Scale - The range of varying levels of light displayable by black and white raster graphics devices. High resolution color raster devices can display the gray scale in addition to a full range of color hues.

HLS - Hue, Lightness, Saturation. A three-parameter system for describing a color based on human perceptual description of color.

Landscape Mode - The rectangular viewing area with the X (horizontal) axis as the longer one.

Logical Graphic Device - An abstract graphic device for which ReGIS defines an image. This abstract device is a composite of a wide range of physical devices, and is used as a hypothetical basis for the ReGIS device interface. Allowable exceptions to the Base Logical Graphic Device are provided for in the Extended Logical Graphic Device.

Line Pattern - Line styles or drawing patterns (solid, dashed, dotted, etc.) defined and supported by ReGIS.

Macrograph - A series of ReGIS commands or command elements that are stored as part of the ReGIS implementation state and invoked on request by a named string. Macrographs reduce communication line traffic between the terminal and computer in the case of frequently used commands. Concept derived from the assembly programming construct of macro instructions.

Offset - A distance from a given reference location.

Open Curve Sequence - A series of points interpolated by ReGIS to draw a curve whose end points do not meet.

Overlay Writing - Writing mode in which only the foreground (primary) writing value is executed into the display, and the background (secondary) writing value is not used. Contrast with "replace writing."

Pixel - Picture element; the smallest displayable unit on the screen.

Pixel Vector - A scalable unit vector whose direction is parallel to an axis or to a diagonal between the axes.

Portrait Mode - A rectangular viewing area with the Y (vertical) axis as the longer one.

Position Address - A specific point in the viewing area of a graphic device that is representable by an X-Y (horizontal-vertical) pair of numbers.

Protocol - The conventions or rules for the format and timing of messages sent and received. Devices must be using the same protocol to understand each other.

Raster Device - A display device that generates images using a raster scan.

Raster Scan - In computer graphics, a technique of generating or recording the elements of a display image by a line-by-line sweep across the entire display surface; for example, the generation of a picture on a television screen.

ReGIS - Mnemonic for REmote Graphics Instruction Set; a set of graphics object description commands.

Relative Location - A point on the screen measured from the current location rather than the screen origin.

Replace Writing - A ReGIS writing mode in which both foreground and background values are written to the display medium; contrast with "Overlay Writing."

Reset - Return to a known default condition.

Shading - An area fill operation that fills the area between a vector or curve and a specified horizontal or vertical line or a point. Shading differs from FLOOD and FILL in that it colors in a solid or textured area which is not explicitly bounded.

Sixel - A coined term derived from "six pixels." Sixels are an image transfer mechanism in which six pixels of information are encoded into a single character code. Sixels may be used as an image re-transmission scheme from ReGIS.

Transportability - The capability of carrying ReGIS-defined images across device technologies without loss of critical information.

Viewing Point - See Pixel.

Writing - The operation of executing graphics to the display medium.

8.1.6 MODELING THE GRAPHICS SYSTEM

The Remote Graphics Instruction Set (ReGIS) is intended to be used for device level portable graphic image definition. ReGIS is meant for the communication of graphic images between a host computer and a remote graphics device. It has applicability to tightly coupled graphics systems in any context in which a graphic image can be stored in geometric component form encoded as a sequence of characters. ReGIS is not intended for use in high performance graphics applications where picture complexity is great or where texture and graded shading are required.

8.1.6.1 The Graphics Pipeline

The state of the art in computer graphics layers the graphics environment in a manner very much like the way networks and communications models are layered. In layered models, higher level layers request services of lower levels to effect actions. Traversing these layers is known as proceeding through the graphics pipeline.

Existing proposals postulate levels for modeling of graphics information and performing viewing operations on the data represented in these models. The level of graphics execution is added to these levels. It is this level that most closely approaches the capabilities of existing graphics devices (as opposed to systems), and it is to this level that ReGIS is directed.

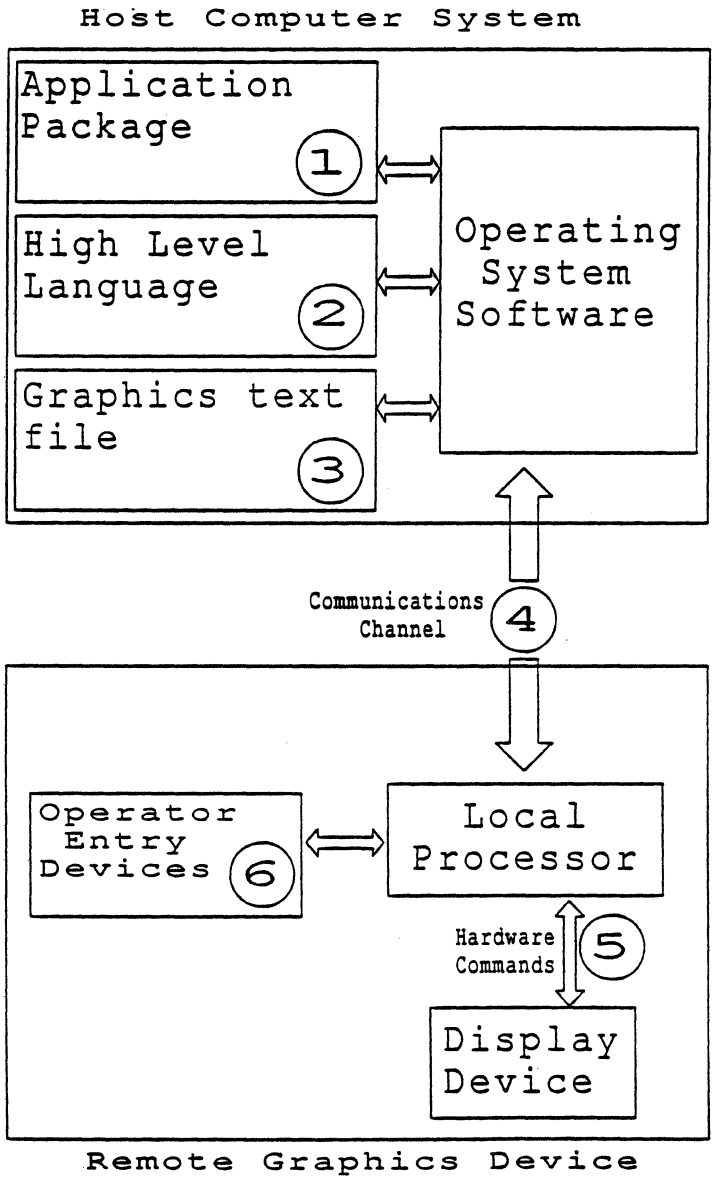


Figure 8-1: Graphics System Block Diagram

8.1.6.2 The System Model

Figure 8-1 illustrates the assumed general block diagram of the graphics system for which ReGIS was designed. As illustrated, the system consists of two major parts: (1) the host computer system, which from the user's point of view consists of various software modules and (2) the remote graphics device. The term "remote" here refers to the assumption that the graphics device is attached to the host computer by a possibly low-speed communications link.

The host computer system is assumed to consist of four major parts (not all of which may be present in any particular system):

1. Graphics application package - Such packages allow the user to access the graphics capabilities of the remote device in terms of syntax and semantics appropriate to the user's intended application. For example, a data plotting package may accept tables of data and allow the user to construct plots of varying types based on that data, using an interaction dialog appropriate to the application.

2. High level language - Users often need to generate graphic images of significant complexity requiring algorithmic definition, or images requiring dynamic changes. Such programs are generally written in higher level languages which may or may not have direct access to graphics functionality.

3. Text files - Sometimes called the graphics "metafiles"; static storage of the definition of a graphic image allows later use of that graphic image in the same manner as printing out a report file of standard text.

4. Operating system - The operating system may provide support to the graphics user in the general case by converting the graphic image definition provided by the other software to the required protocols necessary to drive one or more remote graphics devices, or may use graphics directly to implement a pictorial (iconic) dialog with the user.

8.1.6.3 Graphics Language Access Points

Based on the above assumed structure, the system has at least six language access points of interest to the users and developers of such a system:

1. Graphics application languages - such as plot commands in a data plotting package or the constructors of physical images as represented by the Graphical Kernel System (GKS) graphics standard.

2. High level languages - allow access to graphics semantics in three possible ways:

   a. By direct drive of character based communication protocols by using string constants output through "PRINT" statements

   b. Through graphics subroutines accessed by programmed calls

   c. Through embedded graphics statements accessed in the same manner as other language capabilities

3. Graphics text files - the syntax of how the graphics image is stored as text for later communication to the graphics device (metafiles).

4. Communications Interface - the syntax of the graphics commands sent sequentially to the graphics device.

5. Graphics hardware instructions - the (presumably digital) signals generated by the local device processor to control the hardware image generator.

6. User graphics entries - the specialized entry sequences for local control in addition to the syntax of the user commands required to use the application software programs.

8.1.6.4 ReGIS Scope of Application

Considering the breadth of graphics applications and the differences in "personality" between the different language points, it is not feasible to identify or specify one instruction set which satisfies the requirements of all access points in the system. On the other hand, the relatively large number of different graphics language syntaxes of which a user may have to be aware predicates a few standardized language approaches, with special languages applied when dictated by the application. These latter special applications must be based, where possible, upon

other existing standards. It is the purpose of ReGIS to standardize the syntax and semantics of the communications interface graphics commands. By extrapolation, it is also intended that ReGIS be used at other graphics language access points at which a character based descriptive representation of the graphic image is appropriate. In summary then, it is intended that ReGIS be used for the following graphic language access points for both remote and closely coupled graphics systems:

a. Graphics instruction string constants from high level languages, whether generated by print statements or the compilation of high level language programs having embedded graphics statements

b. The "semantic protocol" for graphics access across a serial communications line to a remote graphics device

c. The syntax of graphics commands when stored in a character oriented file

8.1.7 RANGE OF INTENDED DEVICES

ReGIS attempts to cover a broad range of both soft-copy (temporary record) and hard-copy (permanent record) remote graphics devices. ReGIS executes well-defined, two dimensional, descriptively encoded graphics images. The technology of a device is generally not a factor in the determination of whether that device would be suitable for an implementation of ReGIS.

The principal requirement for transporting ReGIS-defined images is that the essential information content of an image MAY be encoded in the basic set of ReGIS instructions so that little, if any, essential information content is lost. This must hold true if attributes of one device are transformed, or even ignored, by another device which does not have that particular attribute capability.

8.1.8 ReGIS OVERVIEW

Good design practice specifies the function set of a proposed interface before selecting an encoding to implement it. However, the primary operational environments of ReGIS are required to be:

a. Serial communication channels operating under existing text-oriented protocols, and
b. Text-oriented data storage for deferred execution of pictures.

These design constraints make it feasible to define the function set of ReGIS and its encoding concurrently. As a consequence, the function set of ReGIS is defined in this standard in terms of its encoding, both syntax and semantics.

ReGIS instructions are sequences of characters based upon the alphabet common to most computing systems. This alphabet consists of letter, digit, and punctuation characters. A graphic image is drawn by sending the graphics device a sequence of instructions encoded into this alphabet. The general format of each instruction is a keyletter denoting the type of operation to be performed, plus instruction arguments which have a prescribed meaning when associated with a keyletter. The nine basic (keyletter) instructions in ReGIS are briefly defined as follows:

1. Screen Instruction - keyletter "S" - Screen operations affect the entire visible viewing area. Actions include setup of the display space, clearing the display in preparation for a new image, and controls which may affect the re-direction or retransmission of an image.

2. Writing Attributes Instruction - keyletter "W" - Write Attribute instructions are modifiers which affect the actions of drawing instructions following them. Attributes include line style, color and intensity, patterns and textures, and other image modifiers.

3. Position Instruction - keyletter "P" - The Position instruction causes the drawing process to commence or resume from a specific position in the drawing area.

4. Vector Instruction - keyletter "V" - The Vector instruction invokes the line primitive causing straight lines to be drawn between positions specified by the instruction arguments.

5. Curve Instruction - keyletter "C" - The Curve instruction causes circles, arcs, or general curve images to be drawn based upon the arguments of the instruction.

6. Text Instruction - keyletter "T" - The Text instruction causes a sequence of text characters to be drawn in a manner based upon the arguments of the instruction.

7. Report Instruction - keyletter "R" - The Report instruction allows state information to be returned to the host from the ReGIS implementation.

8. Fill Instruction - keyletter "F" - The Fill instruction allows linear elements (lines and curves) to be used as boundaries for simple or complex closed areas, which can then be "colored in."

9. Load character cell Instruction - keyletter "L" - The Load character cell instruction allows ReGIS programs to define character patterns for use in the Text instruction and general character patterns that are used in area operations.

There are four general types of arguments used in ReGIS:

1. Position arguments are strings of characters written within the bracket characters "[" and "]". Bracketed position specifiers are used as effective addresses or address increments.

2. Text strings are character sequences contained within paired quote characters, either ' or ", and are used when a character is itself an argument of an instruction, as in the case of the Text instruction.

3. Digits are interpreted as numeric values, decimally encoded binary values (as logic masks, and so on), or addresses of adjacent neighbors of a given point in a display.

4. Option arguments are strings of characters written within the parenthesis characters "(" and ")" which in some manner modify the way in which the instruction carries out its intended purpose. The syntax of the option strings is the same as for instructions and for this reason, option arguments are also called sub-instructions.

These basic instructions and their arguments are sufficient to define most of the information of most graphic images required by users of graphics systems.

8.2 ReGIS PHILOSOPHY

This section describes and discusses the rationale of ReGIS in terms of:

1. The approach of allowing an image originally defined for one graphics device to be drawn on a dissimilar graphic device, called the instruction set "transportability";

2. Concepts of the instruction syntax and the orientation to the ASCII character code conventions; and

3. Concepts of handling dissimilar device attributes and capabilities, referred to as the "semantics" of ReGIS.

8.2.1 TRANSPORTABILITY CONCEPTS

The goal of image definition transportability is to enable a given sequence of graphic instructions to draw similar looking images on dissimilar graphic devices. There are three major aspects of the design of ReGIS which enhance transportability. These are:

1. Application free primitives

2. A mandatory base implementation

3. Imposition of critical device parameters

Note that complete transportability depends on the user selecting features of a device that are possible to transport across technologies without loss of critical information. For instance, ReGIS cannot preserve the information content encoded as blinking when the commands are transported to a hard-copy device.

8.2.1.1 Application Free Primitives

ReGIS implements a set of drawing primitives based upon common geometric rule and compass constructions, and does not contain any specific capabilities which can be interpreted in terms of the semantics of the application. For example, ReGIS does not include viewing transformation, hidden line removal, and other capabilities generally useful in presenting images of real physical things. Neither does it have built-in axis and bar graph primitives which would be used to support only data plotting. Instead, this model assumes that these capabilities are adequately supported at higher levels in the graphics pipeline, and that these functions are encoded into a sequence of ReGIS instructions. In this way, ReGIS does not burden one application with features required specifically for another.

8.2.1.2 Mandatory Base Implementation

ReGIS instructions are based on a general instruction syntax which all devices are required to parse.

Every device using the ReGIS instruction set must completely interpret all possible character sequences defined by this general instruction grammar, whether or not those character sequences have semantic meaning in the device in question. The syntax of ReGIS is complete and well formed. Only the semantics can vary among devices of differing capabilities, and, at that, only among extensions to the most fundamental required instruction set.

Semantically, ReGIS instructions and parameters are separated into a Base ReGIS category, a Raster Extensions category, and Open Extensions. Other extensions may be defined to apply to other technologies or categorizations. It is assumed that most of the essential information content of any desired graphic image may be defined by the Base ReGIS instructions and parameters. All ReGIS based graphic devices must accurately interpret these Base capabilities.

In addition, users expecting to output the defined images to a variety of devices should use only the Base ReGIS capabilities to ensure maximum transportability. For example, color is not a common attribute of every graphic device, and for this reason the ReGIS Base is "color-blind". Users desiring maximum transportability should avoid using color to relate essential information where that same information is not also implied by monochrome boundaries or patterns.

There are several possible categories of Standard Extensions to ReGIS. These extensions are based on differences in technology, and are used to standardize device features across the class of devices represented by that technology. The most important extension class is for raster scan display devices, including color CRT displays. Standard extensions for a given class of device allow all devices within that class to be controlled in a class dependent, yet device-within-class independent manner. Hence, all raster devices must implement the full range of the standard Raster Extensions.

This concept works, and even allows commands from an extension set to work on simpler devices, because all devices must fully parse the general instruction syntax. Therefore, a capability in one of the extension sets is simply ignored or transformed to an alternate attribute in a device for which that capability does not exist.

This approach is feasible because of the small number of syntactic elements in ReGIS, and their ease of identification.

8.2.1.3 Critical Device Parameters

The third major concept in allowing maximum image transportability is the use of critical setup parameters defined once for each image and placed at the beginning of the instruction sequence for that image definition. Thus, in the worst case, the user might have to adjust these parameters to achieve maximum fidelity in the reproduction of a graphic image. As an example of such a critical device parameter, ReGIS allows the user to set the range of (X,Y) coordinates to be used for an image definition. These should be set to the actual physical range values of the device which the user most often accesses. These range parameters could be adjusted to maximally utilize a dissimilar low resolution device but would not have to be adjusted for displaying the same image on a high resolution device.

Imposing expectations on the device departs from what has been common practice in graphics and other areas of computer peripheral access. Commonly, devices are interrogated for their function sets and features, and the applications take on the burden of manipulating data to fit the device. By placing the burden of accommodation on the device, the application or picture file becomes more portable and less likely to require modification as new devices or classes of devices come into use.

8.2.1.4 Fidelity of Information Presentation

Devices must be allowed varying levels of fidelity in the presentation of information implied by ReGIS instructions. This is a reflection of differences in technology and engineering constraints in product implementations. Fidelity may be measured in a qualitative and quantitative sense. Qualitatively speaking, a line is a line because it appears to connect two points in the display in a continuous fashion. A circle is a circle because it is "round." These aspects of fidelity are, in part, subjective. Pixel oriented devices actually draw lines as a sequence of intensified dots, and the ragged nature of some lines is entirely unpleasant in some instances. However, even these less than perfect lines are, qualitatively speaking, lines.

When fidelity is measured quantitatively, some features of some systems cannot be satisfactorily transported. Display resolution varies from device to device, and the capacity of a display in terms of discernible line pairs or character cells will vary, as will capacities of character cell stores or attribute memories. Where such limits can be quantitatively described, they will be specified with the appropriate instructions as guides to transportability.

Lower resolution and/or monochrome devices by their nature have limited information presentation capacity. Drawing a very detailed image whose information content exceeds the information capacity of the target device may result in essential information

being lost. These devices may conform to the ReGIS standard if the qualitative judgement is that they will serve to display images which preserve "enough" information content for normal use. While this definition may appear quite open and perhaps circular, it is a pragmatic and workable one when the decision as to whether ReGIS is the appropriate graphics support mechanism is well-founded. Higher resolution devices may be expected to implement the highest degree of fidelity of information presentation.

Transportability is achievable if images are prepared on lower resolution devices for later output on higher resolution devices, or that a high information content image is prepared directly for the high resolution device. It is not expected that an image prepared on a high resolution device will be drawn on a substantially lower resolution device, except perhaps for "quick checking." Should this situation exist, however, the user must be willing to accept the possible misinterpretation of information as a result of the lower fidelity of reproduction.

8.2.1.4.1 Guidelines for Image Creators

Given the above, the user must bear some of the burden for balancing the devices available and the applications in mind. If transportability is a goal, then appropriate consideration must be given to the range of devices that may be encountered; the complexity of the presentation should be tailored to that environmental range. For this reason, device or technology specific features should be used only where they are absolutely necessary, or when transportability is not a goal. Even though they are supported through ReGIS, special features cannot be guaranteed across all devices.

There are certain specific methodologies image creators can adopt to enhance the transportability of images. This list is not exhaustive, but should serve to lead the reader's thoughts in the direction that will allow for accommodation of device differences.

a. Do not use expected device resolution to encode maximum information content into an image. While the most aesthetically pleasing results can often be obtained by planning patterning, character cell placement, and the like based on the pixel resolution of known devices, such attention to detail builds into an image a dependency on resolution that precludes image transportability.

b. Specify text character sizing in terms of the imposed address space using the expanded Text commands for Unit {T(U[])} and Display {T(S[])} cell sizes rather than as multiples of standard size {T(S<n>), T(H<n>)}. The definition of standard sizes has been a concession to device dependency, and the inability of devices to smoothly scale or otherwise provide for variable text sizing. While such hardware considerations are not entirely moot, it can be expected that as device resolutions increase and windowed displays come into wider use, the ability of devices to better couple graphics drawing elements and descriptive text will be enhanced.

c. Select colors descriptively using RGB or HLS parameterization rather than by index value. Index values wrap and collide in a manner dependent upon the number of index values supported, and probably without any attention to the perceptual flavor of what is being specified. Collisions between descriptive colors are defined in this standard to merge similar colors (red and orange or red and pink, for example) to some common color ("closest match"). Dissimilar colors (blue and yellow or orange and green, for example) are less likely to collide in a single common color, thereby maintaining perceptual separation.

d. Perhaps most importantly, use the imposition of coordinate address space to define the expected device. The use of device inquiry and accommodation of device differences in software is not the best way to ensure transportability. Such an approach cannot easily be applied to stored files, as it places a burden of re-interpretation of the file on the image transfer system.

   In windowed systems, the actual physical granularity of a display may not be known to driving software, or it may change dynamically in a way that cannot be communicated to driving software. Also, accommodation by inquiry requires that software try to guess what future implementations will do to enhance or subset present devices.

   Device developers have the advantage of hindsight with regard to what features of devices software has used, and in what combinations,and can thus make future devices compatible with existing software, rather than having software try to anticipate, perhaps inconsistently, the direction future hardware will take.

8.2.1.4.2 Guidelines for Image Interpreters

Device developers can minimize the problems that software will encounter in pursuing the goal of transportability.

1. First and foremost, device developers should be guided by the past. Various ReGIS implementations must be syntactically and semantically compatible and minimize non-functional changes from one product to the next. "Non-functional" changes mean capricious changes in device parameters that this standard allows developers the freedom to define, but which have minimal impact in the details of device design. For example, one might argue that all devices with less than 1000 addressable points in the horizontal dimension should have exactly 800. While the "Guidelines for Image Creators" above recommend that software not depend on pixel level registration for images, small changes in resolution, as from 800 to 768 pixels across, can result in significantly different screen presentations.

2. Hardware address spaces should be isotropic, meaning that equal increments of physical addressing quanta horizontally and vertically should translate to the same measured distance. ReGIS requires that the imposed coordinate space be treated as isotropic, and the device must mask any variation from the 1-to-1 pixel aspect ratio. It may not be possible to hide the aesthetic

effects of such mismatches, however. Consider that 45 degree rotation of text is generally deemed an "easy" case. Deviations from 1-to-1 (or at least integrally related) pixel aspect ratios make this case either difficult to execute or unaesthetic.

3. The most visible departure from fidelity in image transportability does revolve around character handling. Variables relating to character storage size, standard sizes, and scalability should be scrutinized carefully for effects on software and existing imaging practice before they are changed. As with other aspects of "non-functional" change, any resulting variations should be based on significant improvement in capability, not just for incremental improvement.

8.2.2 SYNTAX CONSIDERATIONS

8.2.2.1 Character Codes

All ReGIS commands may be formed from characters in columns 2 through 5 of the ASCII code chart. It may be considered, in light of strict interpretation of ANSI character set and coding rules, that ReGIS is a violation of the use of printing character codes for commands. This compromise achieves two goals:

1. To allow an encoding more terse than that required by existing control code standards (ANSI X3.41 and ANSI X3.64)

2. To develop an instruction set which could be easily generated, stored, and interpreted

As a matter of convention, ReGIS is defined in terms of upper-case letters with the understanding that lower-case letters are equivalent to upper-case for command interpretation. Lower-case letters remain lower case within quoted string arguments, however.

8.2.2.2 7-bit Versus 8-bit Encoding

The ReGIS command syntax and semantics are specified for use in a 7-bit coding environment, though provision has been made for the support of 8-bit encoding of character arguments. See subheadings 8.4.3 Argument Types, 8.4.3.2 Quoted Strings. Support of ReGIS command codes in an eight bit environment is an environmental issue, and is not addressed in this standard.

8.2.2.3 ASCII Control Codes

ReGIS does not specify the use of any ASCII control codes, except for certain occurrences within quoted text strings. Any controls or control or escape sequences passed to a ReGIS interpreter are not required to be recognized or parsed by the interpreter[*]. Such controls may be included in a ReGIS data stream for purposes of control of the system or device in which the interpreter is running (such as XON/XOFF for communications synchronization or escape or control sequences to activate or deactivate the ReGIS interpreter) but the recognition and interpretation of such sequences is not required by this standard for conformance.

---

[*]Some devices find it appropriate to scan for and parse escape and control sequences while interpreting ReGIS. Some of these devices execute escape sequences as if they (the devices) were not in ReGIS mode, and return to ReGIS mode after such execution. Other devices use the occurrence of any escape code (code position 1/11 or 8/* or 9/*) as reason to exit the ReGIS mode. Such use may be necessary to support non-conforming software, but such use is deprecated by this standard.

8.2.2.4 Linguistic Relationship to Higher Level Languages

The ReGIS general syntax is based on a simple, easily parsed grammar. This grammar can be easily generated by common high level languages and user application packages. Also, the ReGIS instructions encompass most, if not all, of the graphics primitives required to support the execution of graphics constructs required by higher level graphics applications packages.

The encoding of ReGIS instructions and arguments is such that the use of ReGIS directly in the PRINT or WRITE commands of such languages as BASIC, FORTRAN, and Pascal is practical and straightforward. The representation allows reasonably clear, although admittedly terse, coding. In particular, numeric arguments are encoded as the usual decimal integer, fixed point, floating point, and scientific notation strings generated by these languages.

8.2.2.5 Algorithmic Relation to Higher Level Languages

ReGIS is not meant to be a language in its own right. ReGIS has no algorithmic structures, such as conditional or loop constructs, no general arithmetic capability, and no named variable storage. All these features are presumably present in the higher level languages from which ReGIS may be used for the encoding of pictorial information.

8.2.2.6 Communications Line Efficiency

The level of ReGIS instructions is set high enough so that reasonable performance can be obtained even at low transmission rates. A relatively few command characters can invoke fairly complex graphics activity. The "macrograph" construct is included in the ReGIS Base to allow string sequences to be transmitted once, stored locally, and then referred to by name for subsequent, repeatable invocation.

ReGIS is generally free format and requires no instruction separators, thereby allowing multiple instructions per "line." In fact, ReGIS has no awareness of line boundaries, since ASCII control characters are ignored (except for a few instances, and only within quoted string arguments).

8.2.2.7 High-level Language "Personality"

Although it is not possible to define a completely compatible personality among all higher level languages, ReGIS has attempted to capture the essence of the personality of many higher level languages to allow the most rapid learning by users with minimal corruption of well-known syntactic constructs. Common numeric and string constant constructs, the ignoring of blanks and auto-conversion of lower case to upper case are examples of the implementation of this concept.

8.2.3 SEMANTIC CONSIDERATIONS

8.2.3.1 Range of Devices

The ReGIS Base defines a logical graphics device modeled after a composite of existing and envisioned graphics execution devices. The common capabilities of flat bed and rotary plotters, storage tube devices, raster video and hard-copy devices, and other devices were evaluated in the definition of the command complement of ReGIS. The ReGIS Base defines a common denominator of these devices, and the Extensions categories are defined to access the unique capabilities of these and other graphics devices.

8.2.3.2 Position Addressing

A specific point in the viewing area of a graphic device is assumed to be representable by an (X,Y) (horizontal-vertical) pair of numbers. It is also assumed that the viewing area is rectangular in shape for the purpose of code transportability. The size of an address increment is defined by the coordinate space as imposed by the device setup, or by device default if this setup is not used.

8.2.3.3 Semantic Defaults

In the ReGIS Base, all parameter ranges required by a device are derivable from user supplied screen coordinates. These are transformed to physical coordinates by the device. An exception to this is the size of "standard characters" in the device. These are typically constrained by the physical design of the device. To accommodate such variances, ReGIS allows the user to "adjust" some of these parameters so that the actual visual result is as close as possible to what is desired. Such adaptations may require user intervention (that is, trial and error) to reproduce picture content accurately.

Devices that implement ReGIS extensions are required to implement default values for all such semantic features that affect the visible display. These defaults must approximate, to the degree possible, the visual impression of the image on a device without these visual enhancements. These defaults must be assumed each time the screen addressing operation, S(A[][]), is performed.

8.3 BASE LOGICAL GRAPHIC DEVICE

ReGIS defines an image for an abstract graphic device called the ReGIS Logical Graphic Device. In principle, this abstract device is a composite of a wide range of physical devices. Features unique to specific types of physical devices are part of the extended ReGIS logical device and are discussed in later sections. An image defined for one physical device is transportable to a different type of physical device. However, feature extensions may be ignored or approximated on some devices. Therefore, exact duplication of an image may not always be expected when image definitions are so transported. It is not expected that any one specific device have the parameters of the logical device, but rather each ReGIS based physical device maps (transforms) the parameters given in terms of the logical device to its specific physical parameters.

The ReGIS logical device is defined in terms of:

a. Parameters of the viewing area.

b. Attributes of viewing points in the viewing area.

c. The general process of defining an image in the viewing area by modifying the attributes of the viewing points; interpretation of ReGIS is a serial process, and the visible portions of the image are drawn in the order in which the instructions are received.

d. The allowed range of parameters and attributes.

The phrase "implementation dependent" refers to a parameter or attribute value that has a semantic meaning which may vary with the type of physical device or because of other implementation constraints. The allowed range of variance for an implementation dependent feature is included with the discussion of that feature in this standard.

The physical embodiment of the Logical Device is not presumed by this standard. The Logical Device may be represented by a single physical device, or it could, in fact, be a subset of a single physical device comprising a number of independent Logical Devices. In the latter case, the viewing area (and other physical device resources) available to a particular Logical Device are but a fraction of the whole of the physical device. In either case, the view of the Logical Device as seen through ReGIS is that of a single, independent presentation entity having no effect on and not being affected by any other device.

8.3.1 VIEWING AREA DEFINITION

The ReGIS Logical Device viewing area consists of a rectangular grid of a finite, possibly large, number of viewing points. Each viewing point is of finite area and is located at the intersection of the invisible lines of the two dimensional Cartesian coordinate grid. Usually, a viewing point is the smallest physical area with homogeneous structure that can be seen by the viewer. This viewing point corresponds to the smallest area which can be modified by the physical device.

This area has historically been referred to as the "screen." Because of the connotation of "video" in the use of the term "screen," modern practice uses the term "display surface." Both terms are used in this document. Mathematically, the display surface may be thought of as a set of viewing points such that the union of all viewing points completely covers the viewing area. A viewing point may also be referred to as a picture element or "pixel."

When the screen is rotated so that the image has its "normal" orientation, then the four edges of the rectangular screen are called the left, right, top and bottom edges. The terms horizontal and vertical are used to refer to pixel positions relative to these four sides. The position of each pixel on the screen is uniquely identified by the combination of two numbers which refer to the horizontal (or X) position and the vertical (or Y) position of the pixel. Using this, the pixel VP(X,Y) refers to the Viewing Point at the horizontal value denoted by the number X and at vertical value denoted by the number Y.

This model of addressing the view surface centers pixels on their enumerated coordinates. A different device model places pixels between coordinate values. Device implementors are allowed to exercise certain freedoms in mapping the coordinate enumeration scheme used by their devices to that of the ReGIS Logical Device. There are two important reasons for this:

1. The ReGIS Logical Device Addressing scheme will not always map one-for-one to the physical device space when device transporting S(A) transformations are invoked.

2. The ReGIS Logical Device addressing scheme does not map to all device technologies.

   For example, stroke oriented devices have positioning increments, but no real pixels. Also, the underlying hardware of some devices may support the pixel-interval model as opposed to the pixel-centered model, and do so in a way that precludes the pixel-centered model from being accurately supported.

Conforming software must not depend on pixel level access to a ReGIS device.

Using ReGIS instructions, the user defines the possible range of values of X and Y by identifying the numeric values which are taken on by X and Y at the four edges. Without loss of generality, these parameters may be assumed to be non-negative and directly define the following screen area parameters:

SYT - Value of the Y-position at the top of the screen area
SYB - Value of the Y-position at the bottom of the screen area
SXL - Value of X at the left edge
SXR - Value of X at the right edge

If they are all of integer value, these parameters define the total number of usable horizontal and vertical positions. These totals are represented as follows:

SXN = abs(SXR-SXL)+1
= the number of horizontal screen positions
SYN = abs(SYT-SYB)+1
= the number of vertical screen positions
STN = SXN * SYN
= the total number of unique screen positions (number of pixels)

The ReGIS Base Logical Device is ignorant of the real physical size of a pixel. This accommodates most low resolution graphic devices which have no controllable pixel size, as in the case of raster CRT, stroke CRT, and storage CRT devices. Other devices must define default values of physical pixel sizes from the given data while determining the resolution of the original device. This may be determined by the parameters described above. The concept of physical measure is covered in Extensions to Base ReGIS under subheadings 8.6 The Extended Logical Graphics Device, 8.7 Required Extensions for Raster Devices, and 8.8 Open Extensions to ReGIS.

The parameter SXR is not necessarily greater than SXL and similarly, SYT is not necessarily larger than SYB. To define the ReGIS instructions in terms of the logical device, is necessary to be able to identify the pixels which are above, below, to the left and to the right of a given pixel, in terms of the user supplied screen coordinate definition. This is accomplished by the X,Y screen increments SXINC and SYINC which are defined as follows:

SXINC = SGN(SXR,SXL)
SYINC = SGN(SYB,SYT)

where SGN( ) is the function returning the sign of the difference of the two arguments. By this convention:

VP(X+SXINC,Y) is to the right of VP(X,Y),
VP(X-SXINC,Y) is to the left of VP(X,Y),
VP(X,Y+SYINC) is below VP(X,Y), and
VP(X,Y-SYINC) is above VP(X,Y).

8.3.2 VIEWING POINT ATTRIBUTES

Each viewing point generally has background intensity and foreground intensity attributes as well as a foreground/background selector attribute. They are defined in the following paragraphs. Each pixel has a background "intensity" value called the BI(X,Y). The semantic meaning of the word intensity is implementation dependent in the sense of psychophysical and photometric measurement units. For purposes of the Base Logical Device, intensity is defined simply as some visually discernible property of a viewing point; pixels with different intensity values are, by definition, distinguishable from each other. The Base ReGIS Logical Device is both color-blind and binary in terms of intensity attributes, so the possible values BI(X,Y) are either "off" or "on".

The foreground intensity attribute associated with the pixel at position X,Y is denoted by FI(X,Y) and in the Base Logical Device also has values of either "off" or "on". Because of the binary nature of the base device, the FI and BI quantities are redundant, but are included here for completeness since they will be needed in the definition of the Extended Logical Device.

This model does not elegantly accomodate devices with other than binary display states. In devices that do not have immediately differentiable foregrounds and backgrounds, such as multiplane raster bit map displays, an arbitrary distinction between foreground and background may be required. The accommodation chosen for ReGIS is to consider the background specifier as a secondary foreground. These issues are discussed further in the Raster Extensions.

Each viewing point has a foreground-background selector function denoted by FBS(X,Y,t), which has a binary value that may be a function of time. The FBS function for each VP(X,Y) selects whether the currently defined foreground or the currently defined background attribute is to be seen. Without loss of generality, the values of FBS are "F" (for foreground selection, also denoted by the binary value "1") and "B" (for background selection, also denoted by the binary value "0"). In the base logical device, the value of BI(X,Y) is "off" for all (X,Y) and the value of FI(X,Y) is "on" for all (X,Y). Then the set of all VP(X,Y) for which FBS(X,Y) has value "F" ("1") is visually discernible since it is imposed on a background of different, and uniform, intensity.

8.3.3 GENERAL DRAWING PROCESS

Generating an image on the logical device (as distinguished from defining the image) involves executing a sequence of instructions to modify the FBS values for a selected subset of (X,Y) values. The sequence of instruction execution is important in ReGIS. The general procedure for generating a ReGIS logical image is as follows:

1. Set up the viewing area parameters using the Screen control command, S(A[][]).

2. Erase the viewing area using the S(E) command.

3. Identify a sequence of (X,Y) positions and change the FBS for each of these pixels using the Position and Write control commands and the action commands for Vector, Curve, and Text.

An image may be modified by repeating step 3. A new image may be drawn by repeating from step 2. A new drawing session may be repeated from step 1. Note that step 1 should result in a screen erasure, since step 2 follows it. This serves to preclude pictures with conflicting definition attributes, such as expected display size or orientation, from being shown together. Also, it is consistent with the description of the screen setup command, which cautions that the state of an existing image is not guaranteed across a screen setup operation.

8.4 ReGIS GENERAL SYNTAX

This section discusses the general syntax and some aspects of the semantics of ReGIS. This discussion applies not only to the Base ReGIS instructions discussed under heading 8.5, but also provides the general framework for extending ReGIS in a backward and forward compatible way.

Note

> The syntax of ReGIS is complete as described in this section. No extension to ReGIS will require that new instruction set data types be added which will cause existing implementations to fail. All implementations of ReGIS must recognize and parse all syntactic elements presented here, regardless of the semantic meaning of the syntactic representation. In this way, all programs that generate ReGIS and all those devices and programs that interpret it are compatible.

8.4.1 ALPHABET

ReGIS is based on the ASCII character set (and includes assignment of some national use positions from that code table). Columns 2 through 5 of the ASCII Code Table contain all the characters needed to express all the constructs of ReGIS.

The description of ReGIS is in terms of characters, not their encoding. It is presumed that all Digital systems support ASCII as their standard character encoding. However, there is nothing which precludes ReGIS from being transported in foreign codes, such as EBCDIC, as long as all the desired descriptive characters can be expressed in the chosen code. The user alphabet may differ from the required ReGIS alphabet since any characters are allowable within quoted strings. Transporting such use is in the hands of the users.

The ReGIS command alphabet consists of the following characters:

- Letters: A,B,...,Z
- Digits: 0,1,...,9
- Punctuation: [ ] , ; . ( ) ' " : @ + - =

Any and all extensions to ReGIS must conform to these alphabet selection criteria. In addition, the following alphabet conventions apply.

a. Lower case letters a, b, ... z may be used but are treated as upper case letters when used outside of quoted strings.

b. The parenthesis characters "(" and ")" are reserved for option sequences (outside of quoted strings).

c. The bracket characters "[" and "]" are reserved for position and extent arguments (outside of quoted strings).

d. Single and double quote characters ("'" and '"') are reserved for delimiting literal string arguments.

e. The semicolon character ";" is reserved for use as an instruction separator. The occurrence of the ";" character always terminates an instruction, even if the instruction arguments are incomplete or nesting levels remain pending. The only exception is that the ";" character does not terminate a quoted string construction, and may be included within the quoted string as a literal character. Synchronization should be considered a "recovery" mechanism which, while returning the parser to a known state, does not guarantee how much of the command currently being parsed has been or will be executed. Refer to subheading 8.4.2 General Grammar for elaboration of this point.

   A ";" does not terminate macrograph storage operations, because such storage definitions are not "instructions," but are extra-syntactic operations.

f. The "at" character "@" is reserved for use by macrograph strings and is not used in any other context, outside of quoted strings. Within quoted strings, the "@" character is treated as a literal character. There is no ability to expand macrographs within quoted strings.

g. The comma character "," serves as an argument separator. The comma always terminates the parsing of a multicharacter token, except within a quoted string, where it is treated as a literal character. The comma is needed to separate the X and Y parts of a position argument, consecutive numeric arguments to the alphabet character Load command, and the option letter "E" from any preceding numeric reference in which exponential notation might be used. The comma is legal between any other paired tokens without affecting the syntactic state, unless otherwise noted in this standard.

h. The blank (space) character " " is ignored outside of quoted strings and does NOT serve as an argument separator. Spaces are treated as control characters and may be included anywhere within the data stream without effect, except within quoted strings.

i. ASCII control characters and similar communications line control and device control characters are ignored in ReGIS and are allowable anywhere without affecting the interpretation process, except within quoted strings, where a specified subset of the control characters apply.

Note

Certain controls described above are stated as being ignored or insignificant in ReGIS under certain conditions. There is one exception to this statement: The macrograph definition terminator "@;" may not be divided by any character sequence which may be delivered to the ReGIS parser, including a blank space or other controls. There is no instruction parsing during a macrograph storage, therefore the indivisibility of the terminator construct is a necessity.

8.4.2 GENERAL GRAMMAR

A ReGIS image definition consists of an arbitrary length sequence of instructions that may or may not be separated by commas or semicolons. Any character other than a defined command keyletter is ignored. By this rule, an unattached ReGIS argument (one not logically coupled to a command key letter) is ignored without effect. Therefore, unattached quoted strings can be used to "comment" ReGIS instruction streams or files. The proper way to "unattach" an argument is to precede it with a semicolon. The semicolon terminates any outstanding keyletter, and the following argument is ignored as having no context.

A ReGIS instruction consists of a keyletter followed by an argument list. ReGIS can encode up to 26 different instructions. Only eight keyletters are used in the ReGIS Base allowing considerable room for extension. See subhead 8.1.8 ReGIS Overview.

An instruction argument list consists of an arbitrary number of four argument types arranged in an arbitrary syntactic order. Semantically, all instructions and instruction arguments are assumed to be executed in the "left-to-right" order they are received. ReGIS syntax does not require that instruction arguments be received in any particular order by type. It is the semantics of a given command which will determine the sense of intermixed argument types.

Actions take effect when enough characters have been received to sufficiently specify an action. A drawing action, for example, will usually commence, or continue, upon receipt of a closing bracket (]) of a position argument, indicating that the given position specification is complete. Unless there is a specific need for further information (as would be the case in a curve interpolation sequence), drawing can occur regardless of whether more point specifiers are being sent as arguments to the current command. Similarly, characters received as part of a quoted string argument to a Text command can usually be displayed as received, without waiting for the string to be terminated.

Actions involving potentially interacting argument types, may have to be buffered before "enough" information can be identified. This will often be the case where arguments to option commands are involved. Most option commands set state, as opposed to generating visible output. Where an action (such as output) does occur, it is often modified by state elements that are other arguments to the same option. For example, the Raster ReGIS command to dump the screen bitmap to a printer is, most simply, S(H). The option specifier (H) may, however, include suboption specifiers to indicate the offset of the image when printed, such as

```
S(H(P[10,50]))
```

or arguments to indicate the exact portion of the display to be dumped, such as

```
S(H[100,250][600,400]).
```

The complete command might be

```
S(H(P[10,50])[100,250][600,440])
```

or the equally valid combination

```
S(H[100,250][600,440](P[10,50])).
```

No action should be taken on the occurrence of an option keyletter alone until it is seen to have no subarguments that would modify such action.

In the case of most command options, the entire option must be parsed to identify the complete intent of the command. Syntactically, arguments of the four types could follow the H option indefinitely. The terminating condition must therefore be some indication that no further option arguments can apply to the option letter in question. This indication may be any one of the following:

a. The matching right parenthesis that closes the left parenthesis that initiated the current option level is encountered.

   This is the means by which the example above was closed. Subsequent arguments apply to the S command, and a new keyletter is interpreted at command level.

b. A new option keyletter is seen at the same level at which the current option keyletter is in force.

   Thus the X in

   S(H[100,250][600,440](P[10,50])X)

   indicates that the (H) option is complete, and any buffered state may be applied to its execution. The new option indicated by X need not be semantically significant in the context in question; it is its syntactic presence which serves to delimit.

   The ReGIS implementation resumes parsing at that character, so the character is doubly parsed: once as a syntactic delimiter, secondly for its semantic meaning. Parsing resumes at the option level to the S command.

c. The synchronizing semicolon is encountered.

   In the command stream

   S(H[100,250][600,440](P[10,50];

   the semicolon indicates that the entire parser nesting is to be popped. As shown, this represents popping two levels of options, since the suboption P was not closed. There is no command keyletter in force after this action, and any following arguments will be parsed as unattached.

   The visible actions which derive from this synchronizing action are not always clear, however. In general, any state-saving actions are performed as necessary, but the actions conditioned by this state cannot be guaranteed to occur. In the example above, the offset definition implied by the suboption P[10,50] would likely be saved since it is a state definition action, and does not

initiate a drawing or communication action. The synchronizing action would prevent the actual image dump from occurring. Also, the image extents would not be saved, since they are specified to be volatile settings, and the next S(H...) operation would cause them to be reset to defaults.

Use of the semicolon should be viewed as a syntactic recovery operation, not a semantic terminating operation, except as may be noted in specific exceptions in this standard. Refer to the terminating condition to the Load alphabet command.

It is NOT necessary or expected that exiting the ReGIS environment and reentering it via the resynchronizing entry will have the same effect as as encountering a resynchronizing semicolon. The semicolon and environmental resynchronizations are two separate schemes; they are not meant to be interchangeable.

Environmental resynchronization will force resynchronization even from within a quoted string argument or a macrograph definition. Semicolon resynchronization can do neither of these.

The actions described above show an entity is not complete until it is delimited. Such delimiting may be explicit, as with the closing quote of a quoted extent, or it may be implicit, being terminated by a character of a type not valid for the token in question. This is illustrated by the parsing numeric of values. In the command

T(S1D90A2)

the parsing of the values 1, 90, and 2 is terminated upon the characters "D", "A", and ")", respectively.

In some cases, command parsing may have to be "batched" for good visual effect. Commands which change an output map, for example, may be deferred until all such updates are received, or their option string is terminated. This may be required to prevent undesirable flashing of the image were updates to be received and made piecemeal. Some batched parsings are defined in this standard. Implementations that extend such batchings must clearly state this in their product documentation.

Any decision to implement batching of output may not restrict function otherwise defined in this standard. For example, curve interpolation in ReGIS places no bounds on the number of intermediate points which may be specified as part of the curve definition. An implementation which batches the interpretation of such curve rendering must still support an unbounded number of points.

8.4.3 ARGUMENT TYPES

The following describes the syntax for each of the argument types.

8.4.3.1 Bracketed Coordinate Specifier

A sequence of characters enclosed in the bracket characters ([]) refers to an (X,Y) position argument or increment. The contents of a bracketed extent are always numeric and there is no nesting of brackets. Any non-numeric character other than a comma encountered within a bracketed position argument causes the part (x value or y value) being decoded to be terminated and interpreted as not present. If more than two parts are encountered within one pair of brackets (that is, more than one comma is present), only the first two parts are accepted; the third and any subsequent parts are ignored. See subheading 8.4.3.3 Numerics.

8.4.3.2 Quoted Strings

A sequence of characters beginning with the single quote character (') or the double quote character (") and ending with a quote character of the same type is a text string and treated as a single argument. The contents of a quoted string are entirely arbitrary. Once the opening delimiter is seen, all following characters are interpreted literally until the string is terminated by the matching quote character.

ReGIS allows 8-bit encoding of character arguments within quoted strings. Legal character codes are 2/0 through 7/15, and 10/0 through 15/15 (two 96 character sets, 192 characters total). The accessibility of codes 7/15 and 15/15 is not guaranteed because of their interpretation as control characters in some environments. Provision for them must be made in alphabet storage, however.

The quote characters themselves are not part of the text string; a string which is introduced by a quote of one type is not terminated by a quote of the opposite type. Therefore, a quote of one type can be included in a string by using the quote character of the other type as the string delimiter.

Should it be required to include a quote character of a given type within a string delimited by quotes of the same type, the target quote character must be doubled, that is, repeated in place without any intervening characters. For example,

```
"A"      encodes the string  A
'"'      encodes the string  "
"'"      encodes the string  '
'a''C'   encodes the string  a'C
''''     encodes the string  '
"A'""B"  encodes the string  A'"B
""       encodes the empty string
```

ReGIS instructions may have multiple string arguments. Consider, however, that if 'ABC' and 'DEF' are two consecutive string arguments, then the resulting structure is 'ABC''DEF', which will be treated by ReGIS as the single string ABC'DEF. This undesired result is avoided by separating possible consecutive string arguments with the comma character.

A potential ambiguity arises in the case of doubled quotes separated by control characters. Outside of quoted strings, all control characters and SPACE are treated as "no character" by ReGIS. Within quoted strings used as arguments to the Text command, the controls BACKSPACE, CARRIAGE RETURN, LINE FEED, HORIZONTAL TAB, and SPACE have meaning.

Should paired quotes within a set of paired quotes be separated by BS, CR, LF, HT, or SP, then they are NOT considered doubled, and two quoted arguments are recognized as if they had been separated by a comma. When separated by other controls, these controls are interpreted as "no character" and the quotes are considered doubled, forcing the inclusion of one of the quotes in the string argument. Hence, if in the command stream

T"text"<>"moretext"

the token <> were replaced by one of SP, BS, HT, LF, or CR, then an equivalent command stream would be

T"textmoretext"

because of the resulting doubled quote. Were <> to be replaced by any other control, an equivalent command stream would be

T"text""moretext"

resulting in the effective text argument

text"moretext.

Were <> to be replaced by a macrograph definition (which is specified not to disturb the state of the parser, but which is not allowed within quoted strings), the quotes are NOT doubled. The result is, again, as if the two quoted extents were separated by a comma. The macrograph definition would succeed, but would have the side effect of acting like a supported control character.

8.4.3.3 Numerics

Digit characters represent either numbers or sequences of arguments where each argument is a single digit character. The distinction between such uses is semantic, not syntactic and is distinguished by context. The ReGIS grammar allows numeric information to be skipped when the context is not known to an implementation. Digit strings may have semantic meaning as numbers, and integer, fixed point, floating point, and scientific notation formats are accepted.

The allowable alphabet for numeric representation is composed as follows:

- Digits zero through nine
- Plus and minus signs
- Decimal point
- The letter "E" or "e", for scientific notation

The composition of these elements into numbers is by common rules. A simple BNF-like expression of the allowed format for these numbers is

```
<number>    ::=  [+|-] <unsgnreal> [<exponent>]
<unsgnreal> ::=  <digits>* [.] <digits>+
                   | <digits>+ [.] <digits>*
<exponent>  ::=  E[+|-]<digits>+
```

In this expression format, "|" means OR, ":::=" means production, "<token>*" means zero or more instances of <token>, "<token>+" means one or more instances of <token>, and [] enclose optional constructs. In words, this means that a number consists of:

A string of one or more digits, with an optional decimal point placed either before, after, or within the string, this string being

a. optionally preceded by a single sign character, AND

b. optionally followed by an exponent comprising

   1. the letter "e" (upper or lower case),

   2. followed by a string of one or more digits, optionally signed.

Any character other than those noted as legal numerics terminates the scan, and the accumulated numeric extent is taken as complete.

The character causing the termination is rescanned as part of the next token following the number. Consecutive numeric arguments must be separated by commas to ensure proper decoding.

The results of encountering an apparent numeric which violates the above grammar (such as having more than one sign character, more than one decimal point, a sign character or decimal point with no digits, or more than one exponentiation operator (E) within a single numeric token) are not specified in this standard, and such occurrences must be considered an error with device-dependent recovery.

The inclusion of the letter "E" in the numeric alphabet may create ambiguity in certain instances of consecutive arguments. The construct X(Y1E2) could be a legal specification of the (hypothetical) X command with quantified options Y and E, having values 1 and 2, respectively. Because of the use of the letter "E", the above construct could also be interpreted as the X command with a quantified option of Y with a value of 1e2, or 100. A ReGIS interpreter must always interpret the letter "E" numerically where such use is conceivable in context.

Hence, as shown, the interpretation of (Y1E2) would be as (Y100). Note that (Y1E2E3) would have to be interpreted as (Y100,E3), since the numeric grammar precludes the interpretation of a second exponentiation operator in a single numeric.

ReGIS generators must always precede the "E" character with the option-separating comma when there is any chance of a semantic error.

Where digit strings are used as pixel vectors described under subheading 8.4.5.3 Pixel Vectors, the sign characters and the decimal point are ignored without effect, and the interpretation of the digits takes place normally. Any other character terminates the digit string scan, and the scan state changes to that indicated by the non-digit character. This is also true of the letter "E", which, although normally valid in numerics as an exponentiation operator, must be interpreted as a keyletter at the current option level.

8.4.3.3.1 Dynamic Range and Precision

Implementations are not required to USE any fractional part of a completely decoded numeric. However, all implementations must RECOGNIZE when fractions occur, and not truncate to integers until the number is COMPLETELY parsed and decoded. In particular, exponential format numbers may have a fractional part and yet resolve to an integer. The format 1.02e2 must resolve to 102, not 100.

The need for proper decoding of values expressed using negative exponents is less clear. For example, the form 100000e-3 resolves to 100, but the 100,000 part would overflow the integer precision of many implementations. Thus, the goal of interpreting exponential formats is targeted toward the use of exponentials that resolve into the supported integer range of most devices, and the use of large negative exponents in such forms is deprecated.

All implementations must support signed integer arithmetic in a range no less than -32767 to +32767, that is, sixteen bits of 1's or 2's complement expression.

8.4.3.3.2 Numeric Format Violation

The numeric arguments for the Load character cell command (not a part of Base ReGIS) operate somewhat in violation of the above rules. The Load command uses a potentially variable radix for specifying character masks, and typically uses hexadecimal. In this case, the characters a through f (and A through F) are not keyletters, but numerics. Hence, implementations should recognize the L command as a special case, and, if it is not supported, enter a special skip state to wait for the next synchronizing semicolon, which is required to terminate the Load command.

8.4.3.4 Options

Sequences of characters enclosed within the parenthesis characters "(" and ")" represent instruction options. The syntax of each option is the same as the syntax of a ReGIS instruction, including the four argument types. The option syntax is thus fully recursive, allowing theoretically infinite extensibility of ReGIS semantics. The components of all the arguments types are the same within options as they are at the command level.

For readability, options, also called sub-instructions, may be separated by comma characters but not by the semicolon character. The semicolon character terminates the instruction to the highest level.

ReGIS interpreters must be prepared to accept recursion of options to at least six levels. A conforming ReGIS generator must not exceed this level of recursion. Recovery from an attempt to exceed allowable recursion depth is to the highest command level, just as if a semicolon character had been encountered.

The hypothetical command

```
X(A(B(C(D(E(F))))))
```

represents the deepest legal nesting of option levels. Any argument type applied to suboption letter "F" other than another option level would be properly interpreted. Suboption "E" could legally accept other suboptions, as well.

Unless otherwise specified in the body of the option description, option settings are modal, that is, they represent a change of state which is maintained across commands until explicitly changed by resetting the option or by the side effect of another action. Coloquially, such options are referred to as "sticky".

Some options, particularly in the Curve instruction and all of the temporary Write options, are "volatile." They remain in effect only for the duration of the command in process. These options revert to their default state (or to their previous modal state, if applicable) when a new command keyletter or a synchronization operation is processed.

All commands whose actions reference state set by the Write options command, W, may include W as an option, and may set any W-settable state for the duration of that command. The W-state previously in force (set by a W at the top command level) is saved when the option W is encountered, and is restored when the current command is terminated by synchronization or the occurrence of a command keyletter. Such option level state is always volatile.

8.4.4 MACROGRAPH STRINGS

The concept of the macrograph is derived from the assembly language programming construct of macro instructions (macros). The macrograph facility is "extra-syntactic."

Macrograph strings provide the ability for commonly used character sequences to be stored in the graphics device and then referred to, repeatedly if necessary, in the sequence of instruction execution. This facility reduces the number of characters that must be transmitted over the communications interface. For example, if the image being drawn is a sheet of music, each of the different types of notes could be defined in a macrograph; they could then be referred to by name rather than by content. The result can be a significant reduction of communications overhead without substantially reducing image definition readability. In some cases, readability may even be improved by such compaction.

Macrograph strings are "extra-syntactic" in that they conform to a grammar built on top of the ReGIS general grammar. Therefore, macrograph definitions may take place anywhere in a ReGIS command stream without affecting the state of the ReGIS parser. A macrograph definition may even occur within the extent of a numeric argument without adversely affecting the interpretation of that numeric value.

Because of the co-precedence of quoted string extents and macrograph transactions, however, neither macrograph definition nor expansion may take place within a quoted string. Similarly, a quoted string within a macrograph definition is not treated as a quoted string until the macrograph is invoked.

A macrograph invocation affects the state of the parser only insofar as it provides a different character input path. In the process of parsing ReGIS instructions, the detection of a macrograph string reference causes the characters previously defined for that macrograph to be substituted for the string reference characters. In the general case, a macrograph string may be just an argument of an instruction (such as a position argument) or may even be a piece of an argument (such as an argument to a sub-instruction or one element or part of an element in a coordinate pair).

All operations relating to macrographs are initiated by the "at" character "@". Altogether 26 macrograph strings can be defined, each identified by a single letter of the alphabet. Lower-case letters are converted to upper-case for the purpose of macrograph string identification.

The entire set of macrograph strings is cleared by the character sequence "@." ("at" followed by period). The operation of clearing a macrograph causes the string to consist of no characters (the empty string). The default content of all macrographs is the empty string.

An attempt to execute the global clear (@.) from within a macrograph is an error. Determination of and recovery from this error is not specified in this standard.

A macrograph string is defined by the sequence

@:<letter><string>@;

where <letter> refers to any alphabetic character, and <string> is any sequence of characters not containing "@;" or "@:". The sequence "@:" (colon) is called the macrograph definition initiator. The sequence "@;" (semicolon) is called the macrograph definition terminator.

Neither a macrograph clear nor a macrograph definition affects the state of the ReGIS interpreter. A macrograph definition may occur at any point in the ReGIS instruction stream, between keyletters and arguments, between arguments, or within an argument, without affecting the interpretation of of the ReGIS stream. Macrograph invocation, on the other hand, causes previously stored pieces of the ReGIS stream to be inserted into the interpreted stream, in a manner indistinguishable from non-macrograph ReGIS.

A macrograph string is invoked by the sequence

@<letter>

where <letter> is one of the names given a previously defined macrograph. Such a sequence may appear anywhere in a sequence of ReGIS instructions, except within a quoted string. Macrographs cannot be expanded from within quoted strings.

A macrograph string cannot contain the definition of another macrograph string. Determination of and recovery from this error is not specified in this standard.

A macrograph string may include invocations of other macrograph strings, but a macrograph string must not invoke itself either directly or indirectly. There are no conditional execution controls in ReGIS, therefore there is no termination mechanism for recursive invocation. Determination of and recovery from this error is not specified in this standard.

Once defined, a macrograph may not be extended. The first action taken upon the occurrence of a macrograph definition initiator must be to clear the contents of the specified macrograph.

A single macrograph string may be of any length from no characters up to the defined storage limit of the ReGIS device, thereby excluding all other macrographs from use. Conforming implementations must provide for no less than 4000 characters of macrograph storage. User attempts to exceed the macrograph storage capacity will "fail soft" with the following results:

a. No other macrographs already stored will be affected.

b. Any part of the macrograph string being stored which is already stored will remain stored.

c. The remainder of the attempted definition will be discarded, but the definition will terminate "normally" upon the next occurrence of the macrograph definition terminator, and there will be no effect on the command state of the ReGIS interpreter.

8.4.5 POSITION ARGUMENTS

ReGIS maintains a current drawing position. This means that each graphics primitive starts at the "position" left by the previous operation. Vector commands, for example, have an implicit start point, and only a single argument (the other end point) is required to specify a line. The effect of any command upon the current position is described with the definition of that command.

ReGIS implementations may, at their option, provide a visible cursor which tracks the current drawing position. This standard neither specifies nor precludes such a feature.

ReGIS provides two syntactic structures for the definition of drawing positions: (X,Y) coordinates and pixel vectors. The (X,Y) coordinate structure is the traditional system of defining positions. It specifies two numbers separated by a comma, with a semantic extension which allows absolute and relative coordinates to be distinguishable in context, thus eliminating the need to have separate relative and absolute instructions or options. Pixel vectors correspond to the "chain encoding" technique used to drive many incremental plotting devices.

8.4.5.1 Position Argument

A position argument is delimited by a pair of left/right square brackets, both of which must be present. A position argument consists of an x-position part and a y-position part, either or both of which may be missing. The x part precedes the y part. Each component is made up of numbers, following the rules for numbers as standalone arguments. The y part, if present, is separated from the x part by a comma. See subheading 8.4.3.4 Numerics.

Each coordinate part consists of a number possibly preceded by a plus or minus sign ("+" or "-"). If a coordinate part is not preceded by a sign character, that argument represents an absolute coordinate; the value of the coordinate part is the new number, irrespective of the value of the current drawing position. If a coordinate part is preceded by a sign character, then that part is evaluated relative to the current drawing position value. A position argument may have one coordinate part as a relative specifier and the other coordinate part as an absolute specifier.

If a coordinate part is missing, that part is interpreted to mean +0, which is a relative specifier indicating a displacement of 0. Note that +0 is a different specifier from 0, which is an absolute specifier.

For each example below, assume the initial values of X and Y are 100 and 50 respectively.

Position Spec	Effective Addr		
[0,0]	0	0	
[30,23]	30	23	
[200]	200	50	(only X changed)
[,42]	100	42	(only Y changed)
[+10,-25]	110	25	(relative change)
[0,+10]	0	60	(combination relative and absolute change)
[15,20]	15	20	
[10,20][+5]	15	20	(multiple specifiers have cumulative action)

8.4.5.2 Use of Bracketed Extents for Non-Positional Information

Some ReGIS commands are defined to use the square brackets to enclose non-positional information, as for specification of positional information not coupled to the current drawing position or to the imposed coordinate space in effect. Some uses are for expression of rectangular extent, others are for situations where "position-like" information is required.

The following uses of bracketed extent are non-positional.

In Base ReGIS:

1. S(A[][]) - The contents of the bracketed subarguments define the imposed address space. They have no dependence on current position, and signed values are interpreted as absolutes. Negative values for imposed coordinate space are not allowed. Missing values (fewer than two bracketed extents or unspecified values within the brackets) result in illegal specification.

2. T[] - Character escapement for the text command is measured in the imposed coordinate space, but these values are always interpreted as relative to the current position, even if the provided values are unsigned. Omitted values are interpreted as spacing of +0.

In Raster extentions to ReGIS:

3. T(M[]) - The storage cell multiplier is not positional, but the two elements are an ordered pair of multipliers.

4. T(U[]) and T(S[]) - The Unit and Display sizes of character cells are measured in the imposed coordinate space, but do not relate to current position. Missing values default to values associated with standard size characters.

5. L[] - The storage size of a character is measured in technology-dependent units, usually bits of storage. There is no association with current position; missing values are interpreted as 0.

In Open extentions to ReGIS:

6. S(H(P[])) - The hard-copy offset is not coupled to current position. Its relationship to the imposed coordinate space is implementation- and retransmission protocol- dependent. Missing values are interpreted as 0. The bracketed arguments to the H option itself ARE positional, do relate to the imposed coordinate space, and may be expressed in terms of the current position.

8.4.5.3 Pixel Vectors

A pixel vector is represented by a single digit character in the range of 0 to 7. Each of these numbers refers to one of the eight pixel neighbors relative to the current position. As illustrated in Figure 8-2, pixel vector number 0 refers to the right, pixel vector number 1 refers to the pixel up and to the right, and so forth around a circle in a counter-clockwise direction. These directions are constant, and are not affected by the display surface orientation set by the Screen address setup command S(A[][]). That is, pixel vector 0 is always to the right, even if the +X direction has been defined to be to the left.

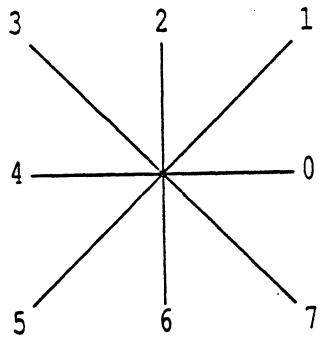


Figure 8-2: Pixel Vector Directions

The character sequence 2057 refers to a sequence of four pixel vectors and not to the number two thousand fifty seven.

Pixel vectors allow only relative positioning and not absolute positioning.

The default size of a pixel vector is the unit integer size defined by the screen coordinate setup operation, S(A[][]). This length is modifiable by a multiplicative factor specified in the writing attributes instruction. Scaling of horizontal and vertical components of the pixel vector is isotropic, that is, an increment of distance in X is taken to be the same distance as that increment in Y.

Depending on the granularity of the unit pixel size defined by the screen coordinate setup operation, the exact position after executing a sequence of pixel vectors is subject to roundoff error and thus pixel vector positioning sequences should be broken occasionally by absolute positioning, or by position reports.

For the examples which follow, assume that the ReGIS coordinate space has been set with an upper left origin, and that initial values of X and Y are 100 and 50, respectively.

Multiplier	Pixel string	New (X,Y) Value	
1	0	101	50
1	1	101	49
1	0002	103	49
(any)	01234567	100	50
5	667	105	65
23	5	77	73
1	0000000000	110	50

8.4.5.4 Position Stack

ReGIS provides for the current drawing position to be saved for use by commands which may follow in the command stream. The mechanism for such saving is a stack. A current drawing position can be saved by a "begin block" operation and later, after possibly several intervening position changes, the position can be restored to its old position by an "end block" operation.

The begin and end points are identified by option characters "B" and "E" in the option arguments of the drawing and positioning instructions which use the block structure.

There is only one stack on which positions are saved, and only the Vector and Position instructions have access to this stack. The Curve instruction does NOT use this stack, since the state required for curve maintenance is more complex than that of vectors and positions. Thus, an "end" reference in a vector instruction can cause a position saved by a "begin" operation in a position instruction to be returned.

ReGIS devices must provide a stack at least eight positions deep. While more depth may be provided by devices, conforming software should never assume more than eight entries.

Should stack capacity be exceeded during a save, the current element (the value for which the save failed) is not saved, but discarded.

Stack underflow will cause NO ELEMENT to be returned. NO ELEMENT is distinguished from an empty bracketed pair ([]) in that the former implies no argument present (as would be fetched from an empty macrograph), and the latter implies the use of current position for a position argument. In the case of a null argument to a Vector command, no drawing is done; in the case of an empty argument ([]), a single dot is drawn to indicate a line drawn from the current position to the current position.

8.4.6 EXTENSIBILITY REQUIREMENTS

The following notes give general requirements for extending ReGIS to take advantage of device specific characteristics:

a. The ReGIS syntax of keyletter and four defined argument types is fixed and will not be extended. New features must be added to ReGIS by assigning meaning to previously unassigned keyletters, or by assigning semantic meaning to previously unspecified keyletter/argument combinations, such as by adding options to existing instructions.

b. A new instruction type shall be added to ReGIS only if:

   1. The new operation cannot be conveniently or efficiently performed by a sequence of existing ReGIS instructions, and

   2. The designer can ensure a minimal loss of information content when the new instruction is ignored by a ReGIS device which does not implement the instruction.

   By this reasoning, new instruction types must minimize side effects of state to the greatest degree possible. For example, no command not already defined shall affect the drawing position, as this element of state is critical to the basic drawing process, and commands unknown to older implementations could not possibly be accommodated if such changes were to occur.

c. Extensions must be unique. The same command semantics must not be used to specify different effects on different types of devices, except where the differing effects are closely related and controlled by a mode setting. (Refer to the two modes of on-screen data movement and the S(D) switch as an example.)

d. ReGIS shall not be extended beyond its intended range of applicability as outlined under heading 8.2 ReGIS Philosophy.

e. Extensions will be registered via Engineering Change Orders (ECOs) to this document, either as additions to existing chapters, new chapters, or in appendices, as appropriate to the meaning and classification of the extension.

Warning to Device Implementors

Product dependent extensions to ReGIS not registered in this standard may jeopardize the certification of the product as ReGIS-conforming. Such extensions include assignment of unused keyletters or arguments, or interpretation of assigned elements in a non-standard way.

8.5 BASE ReGIS INSTRUCTIONS

This section details the semantics of the eight Base ReGIS instructions.

8.5.1 SCREEN INSTRUCTION

The screen instruction, keyletter "S", controls screen coordinate parameters and attributes which affect the entire viewing area. In the ReGIS Base, this instruction includes clearing the screen area, setting the coordinate system to be used, and making adjustments to device specific parameters.

8.5.1.1 Position Arguments

There is no significance to bracketed position arguments to the S command in Base ReGIS.

8.5.1.2 Numeric Arguments

There is no significance to numeric arguments to the S command in Base ReGIS.

8.5.1.3 Quoted String Arguments

There is no significance to quoted string arguments to the S command in Base ReGIS.

8.5.1.4 Options

8.5.1.4.1 S(A[][]) - Screen Address definition

The screen coordinate definition option allows the user to define the coordinate range to be used in position arguments. In the absence of this command, the default device coordinate range will be in effect. The implementation default is not standardized, and may or may not, at the implementor's option, reflect the actual physical resolution of the device. Throughout this document, the coordinate space defined by the last effective S(A[][]) command is referred to as the "imposed" coordinate space.

Warning to Applications

> Applications failing to use the screen address command for device setup run a great risk of sacrificing portability. There can be no guarantee that future devices will maintain the same address range as existing devices; applications that do not include provision for imposing a coordinate space may prove very limited in terms of inter-device portability.

The screen definition option has two position arguments. These specify the user-defined coordinates of the upper left and the lower right corner of the viewing area rectangle, respectively.

If fewer than two address specifiers are given in the S(A[][]) command, or if the specified address space would require absolute negative coordinate values to be properly referenced, the command is ignored and the address space previously in effect shall be maintained.

If more than two address specifiers are given in the S(A[][]) command, just the first two are used, the remainder are ignored. Missing elements inside the brackets cause the command to be ignored. As described under heading 8.3 Base Logical Graphic Device, this will be sufficient information for the device to set up appropriate scaling functions for most other address-related device features. In particular, the default length of the pixel vector is taken to be equal to 1.0 unit as defined by the values of the addresses of the corners specified in the S(A[][]). Other side effects of this operation are listed at the end of this discussion.

The screen coordinate option is not meant to be used as a viewing or image transformation. That is, it shall not be used to define a coordinate system which "makes sense" for the user's specific problem. Such transformations are more appropriately accomplished with higher level software. The option is to provide a mechanism for transporting image definitions from one device to another, while ensuring that the maximum capability of a device is usable in the simplest cases.

The state of any image existing in the display need not be guaranteed when an S(A[][]) is executed. In particular, devices may perform a new frame or screen clear action if that is required to support a new imposed coordinate range. Applications should still send the S(E) command before beginning to draw. Not doing so could produce undesirable side effects.

8.5.1.4.1.1 Mandatory Arithmetic Range

The screen address setup applies to the orientation and magnitude of the Cartesian (bracketed) coordinate specifications of addressing the display surface. Left and right, up and down, clockwise and counterclockwise describe the user's view of the display in its normal orientation. As described earlier, pixel vector orientation is always 0 to the right, 2 up, 4 left, and 6 down, with the odd numbers diagonally in between. The default direction for text is left to right along the baseline; down is the direction below the baseline. Thus, an attempt to reflect an image in its totality merely by inverting the X address space (as to implement rear projection video, for example) will likely fail.

Though the screen addressing limits specified in the S(A[][]) command will normally cover the desired range of device addresses, use of values outside this range may be desirable. ReGIS devices must properly interpret addresses beyond the "edges" so defined to at least twice the defined extent of the display. Such legal use is positioning the center of an arc outside the display, even though the entire arc drawn is within the viewing area.

Since ReGIS implementations may provide for as little as sixteen bits of integer arithmetic precision, the effect of the two bit "guard band" on off-screen addressing is that the largest guaranteed value for on-screen addressing is 16383, or (2**14)-1.

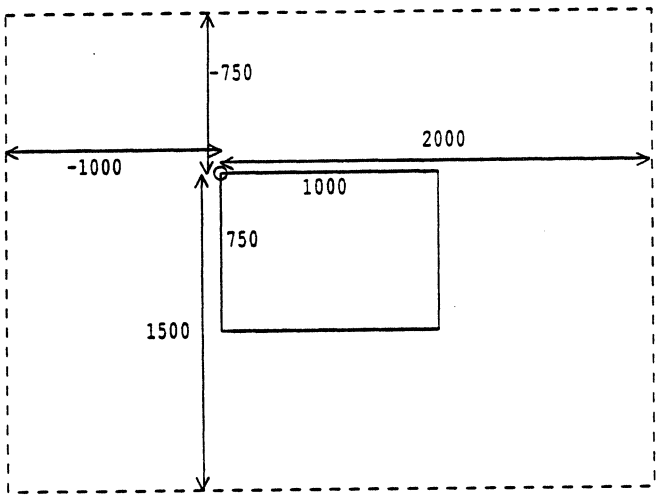


Figure 8-3: Off-display Screen Addressing Example

The command

S(A[0,0][1000,750])

defines a display extent 1000 by 750 units, with an upper-left corner origin. The legal addressing range must then be at least -1000 to +2000 in X, and -750 to +1500 in Y. This allows relative specifiers to access negative addresses, and off screen accesses to be recovered without special precautions. Figure 8-3 shows this positioning of address space.

Left and right margins shall not "wrap" immediately to one another, nor shall top and bottom. ReGIS generators should never expect to take advantage of such abnormal effects for defined purposes.

8.5.1.4.1.2 Accommodation of Shape

The region defined by the S(A[][]) is the "region of interest" of graphics output. It is device dependent as to whether all graphics output is limited to this area. For example, the area specified by S(A[][]) might not use the entire visible extent of the device, either because the device cannot adequately scale the range specified to the entire width and height of the display, or because the shape of the area specified does not exactly match the shape of the display. In such cases, it is the device's option as to whether graphics actions taken outside the range specified by S(A) are visible in this unmapped area.

Furthermore, it is left to the device to position the area represented by the S(A[][]) as it can within the visible display, with three constraints:

1. The shape (aspect ratio) implied by S(A[][]) must be maintained.

   The command S(A[0,0][1000,750]) implies an area one and a third times as wide as it is tall. The area on the display to which the source address space is mapped must maintain this shape; if the target width is to be 768 units wide, for example, then the height must be 576 units tall.

2. All the area implied by the S(A[][]) must be visible in the target mapping.

   In the case immediately above, if the available height of the display were only 480 units, so that the required 576 units would not fit, then the width used could not exceed 640 units, maintaining both shape and visible extent.

3. An implementation may elect to use an integral scaling to fit the requested coordinate space into the device space, but any such mapping must use at least half the visible extent of at least one of the horizontal and vertical extents of the available hardware.

   Consider the cases as above, in which the requested space is 1000 by 750 units. If the target device has native addressing of 768 by 480, then incoming address values can be divided by two, and only 500 by 375 device addressing will be used. Note that if the requested address space were 1000 by 300, then dividing by two would use only 500 by 150 device units. While 150 is less than 1/3 the available vertical range, more than half of the horizontal range is used, meeting this requirement while maintaining the requested aspect ratio.

The placement of any subset range within the device space is left to the device implementor. Conforming software shall not depend upon a given alignment of requested space within device space, nor shall it depend upon the exact scale factor used to map the requested space to the device space. Figure 8-4 shows how a 1000 by 750 unit space could map into video- and page printer-like display spaces.

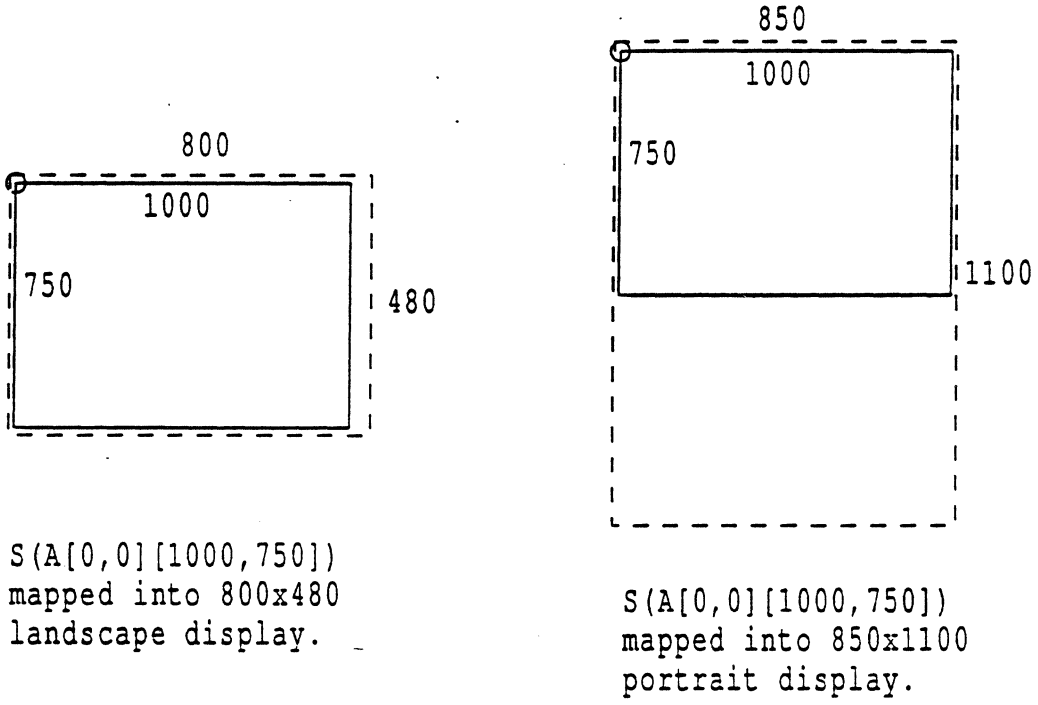


Figure 8-4: Accommodation of Aspect Ratios

Within the constraints described above, a device is allowed to re-orient its display surface with respect to horizontal and vertical to better match an imposed address space, as long as all reorientation takes place as if the device were exactly a device of the assumed aspect ratio. In other words, "turning" the display surface horizontally or vertically to fit a particular graphic may be desirable.

The above action is not likely in an interactive video device, but may be useful in a hard-copy device, where the output is viewed out of the context of its generating hardware. For example, an 8 1/2 by 11 inch plotter best mimics the 4 to 3 aspect ratio of video when the long axis is the horizontal (landscape mode). However, if used to execute graphics for publication, the vertical may be the more natural long axis (portrait mode). The magnitude of the horizontal to vertical aspect ratio implied by the S(A[][]) relative to unity may be used to infer the utility of reorientation.

Each physical device has its own pixel and picture aspect ratios. The pixel aspect ratio is the ratio of the width to the height of the physical picture elements of the display. ReGIS expects the display medium to be isotropic (equal measure in the X and Y dimensions, that is, a perceived pixel aspect ratio of unity). The implementation must shield variations in pixel aspect ratio from the application; ReGIS provides no assistance.

8.5.1.4.1.3 Numeric Accuracy

Note that ReGIS allows numeric parameters to be given in integer, floating point, and scientific notation. The following two rules are guidelines to be used in determining the degree of fidelity necessary for different devices in implementing the numeric formats:

1. Low resolution devices are assumed to use integer numeric quantities only. However, low resolution devices must extract the integer portions of floating point and scientific notation number constants. Refer to subheading 8.4.3 Argument Types and subheading 8.4.3.3 Numerics for rules on numeric interpretation.

   Allowing the truncation of numbers with fractional parts to the next lower integer numbers on low resolution devices precludes the use of the unit screen concept of fractional addressing.

2. High resolution devices are expected to use the entire resolution provided by floating point numbers. The fractional portion of numbers should not be ignored. These devices are expected to use a nearly exact coordinate transformation algorithm which loses little, if any, of the usable viewing area, subject to mismatches of imposed and available aspect ratios.

8.5.1.4.1.4 States Set to Default as a Side Effect of S(A[][])

The following states are set to the following default values whenever a valid S(A) is executed: A "valid" S(A) is one with two extent arguments, both of which are expressed in terms of non-negative imposed position values. The equivalent ReGIS command which would institute this setting is shown after the defined action.

a. Current position is set to [0,0] in the imposed coordinate space.

   Equivalent explicit command: P[0,0]

b. The pixel vector multiplier is set to 1.

   Equivalent explicit command: W(M1)

c. The line pattern is set to solid, and the line pattern element multiplier is set to 2.

   Equivalent explicit command: W(P1), W(P(M2))

d. Text character escapement is set to rightward movement the width of one standard size 1 character cell, with device-optional extra intercharacter spacing, and no vertical movement. Vertical movement executed upon Line Feed is one standard size 1 character height downward, with device-optional extra downward interline spacing. Pixel vector character motion is always one-half a character extent.

   Equivalent explicit command: T[<+w>,0] assuming X increases rightward (else use <-w>); w is not less than the width of a standard size 1 character

   Equivalent explicit command: T[0,<+h>] assuming Y increases downward (else use <-h>); h is not less than the height of a standard size 1 character

e. Character set selection is returned to the default state for the device. This is normally ASCII in GL and the appropriate supplemental set in GR.

   Equivalent explicit command: T(A(L"0B",R"1A"))

The macrograph store is not affected by executing an S(A).

8.5.1.4.2 S(E) - Screen Erase

The screen erase option initializes the display surface in preparation for a new image. The precise meaning of "erase" is device dependent, but generally has the connotation of creating a homogeneous visual image. In some systems, this operation is called "new frame."

S(E) clears the position stack (loaded by P(B/S) or V(B/S)). Shading state, a ReGIS extension set by W(S), is disabled on S(E).

8.5.1.4.3 S(F<integer>) - Feed medium

In hard-copy devices with movable media, this command causes a "sheet feed" or equivalent action to occur. The numeric argument to the F option indicates how many frames of the medium are to be advanced. The default value, 0, indicates that the medium is to be moved one frame only if some drawing action has occurred since the last S(F) or S(E) command. Devices which can recover media already advanced are allowed to implement negative values for this option. Negative values are interpreted as 0 in devices which cannot support negative values.

S(F) can be used as an "image complete" indicator for ReGIS interpreters, including both hard-copy devices and image conversion utilities. Systems which do not run interactively often need such an indicator to identify the end of the ReGIS stream, so that they may take further image processing action. S(F) does NOT imply image retransmission, as does S(H), and should be used at the end of any stored ReGIS file to indicate "image complete," even on files sent to interactive devices. ReGIS-based images intended for file storage should be terminated with S(F) to facilitate their handling in such environments.

8.5.2 POSITION INSTRUCTION

ReGIS assumes and uses the "current position" or "current drawing position" convention for determining where a drawing instruction starts its operation in the viewing area. A ReGIS device is required to maintain the value of an X and Y set of numbers, and these saved values are assumed always to be the starting point for a drawing operation. These values are updated to new values after each drawing operation. An example of the current value concept is the position of a drawing pen on a flat-bed plotter device.

The Position instruction, keyletter "P", changes the value of the current drawing position without drawing any visible image. One physical analog is the movement of the plotter pen with the pen up. ReGIS allows the drawing position to be moved on both absolute and relative bases.

8.5.2.1 Position Arguments

Position arguments to the position command cause the current drawing position to take on the value specified by the argument.

Multiple position arguments may be associated with a single Position command, or pixel vectors may be intermixed with position arguments. Each argument is executed in turn, providing cumulative results.

8.5.2.2 Numeric Arguments

Numeric arguments to the Position command are interpreted as single digit pixel vector specifiers. This supports programs which use "chain encoding" techniques and allows small relative changes in the drawing position. A sequence of pixel vector digits may be used, with accumulative results. The number of unit pixels actually moved for each digit is determined by the granularity (the unit distance) of the imposed coordinate space and the current pixel vector multiplier value as set by the writing options instruction.

8.5.2.3 Quoted String Arguments

There is no defined meaning for quoted text as an argument to the Position command in Base ReGIS.

8.5.2.4 Options

8.5.2.4.1 P(B), P(E), and P(S) - Position stack operations

The position begin and end options allow a simple means for recording a current position value and then recovering that value at some later point in the instruction stream. The P(B) option saves the current value of X and Y on the position stack. Should subsequent drawing or positioning instructions change the drawing position, execution of the P(E) option causes the drawing position to be restored to the saved value. This helps define images which are to have maximum transportability, since it allows accumulative roundoff errors to be occasionally eliminated, particularly in the case of text drawing which may be very roughly approximated on low resolution devices.

The construct P(S) is provided for symmetry with the V(S) and C(S) position constructs. P(S) saves a dummy entry on the position stack. This entry, when recovered by a P(E), performs no action beyond incrementing the stack pointer to pop the dummy entry. In this way, the same drawing positions will be visited by Position, Vector, and open Curve instructions executing the same parameter sequence.

The position stack is cleared to no entries on the execution of S(E) or when a valid S(A) is executed.

8.5.2.4.2 P(W) - temporary Write options

The W option to any of the drawing action commands temporarily changes the write attributes to the specified values for the extent of the instruction in which it occurs. Temporary write options revert to their previous state upon the occurrence of a new command keyletter (not necessarily different from the one in effect) or when the interpreter is brought to command level by synchronization, such as by encountering a semicolon.

Suboptions to the P(W()) form are the same as for the W() command itself. The only meaningful suboption in the context of the Position command is the pixel vector multiplier, that is, P(W(M<m>)).

8.5.3 WRITING ATTRIBUTES INSTRUCTION

The writing attributes instruction, keyletter "W", provides controls under which the image is to be drawn. In the ReGIS Base this includes the selection of line drawing patterns and the adjustment of pixel vector sizes. Extensions to ReGIS add controls for color, intensity, and drawing modes, among other features.

8.5.3.1 Position Arguments

There is no assigned meaning for bracketed position arguments to the Write attributes command in Base ReGIS.

8.5.3.2 Numeric Arguments

There is no assigned meaning for numeric arguments to the Write attributes command in Base ReGIS.

8.5.3.3 Quoted Text Arguments

There is no assigned meaning for quoted text arguments to the Write attributes command in Base ReGIS.

8.5.3.4 Options

8.5.3.4.1 W(M) - Pixel Vector Multiplier

The M option to the attributes command defines the pixel vector multiplier that apply to pixel vector arguments. These pixel vector arguments apply to the commands which interpret numeric arguments as addressing constructs. A numeric argument is a multiplicative scale factor to be applied to the default pixel size set by the S(A) command. It may be a floating point value in systems which support floating point arithmetic, but may be truncated to an integer in systems which do not. A value of 0 is taken to be 1. There is no upper bound on legal values for the numeric argument other than that imposed by the base arithmetic limits of the ReGIS interpreter (16 bit integer arithmetic).

The pixel vector multiplier is set to 1 when a valid S(A) is executed.

8.5.3.4.2 W(P<n>(<suboption>)) - line Pattern definition

The ReGIS Base does not require that devices draw in color or gray scale, as these attributes do not transport easily across devices. The Base ReGIS definition assumes that lines can be presented which differ by line style, that is, dashed, dotted, solid,and so on. There are ten predefined line drawing patterns, and ReGIS supports bit by bit definition of custom drawing patterns. A single digit argument to the P option selects a line style from the following list.

P0 - no visible foreground image specified
P1 - Solid line
P2 - Dash pattern
P3 - Dash dot pattern
P4 - Dot pattern
P5 - Dash dot dot pattern
P6 - Sparse dot pattern
P7 - Asymmetrical sparse dot pattern
P8 - Sparse dash dot pattern
P9 - Sparse dash dot pattern

P1 is the default line drawing style.

When the numeric argument is a string of binary digits (decimal ones and zeros), the argument represents a bit by bit definition of a custom pattern. Base ReGIS interprets the user defined pattern as a sequence of line segments based upon an alternating pattern of P0 and P1 line types. For example, the sequence

P111010

constructs a Dash Dot pattern similar to the P3 line type.

Digit arguments to the (P) option are treated individually. If a digit string consists entirely of ones and zeros, then it is interpreted as a binary pattern specification. Any other single digit specifies one of the line patterns 2 through 9 noted above. The last such digit in a sequence of digits selects the appropriate pattern.

These standard patterns 2 through 9 are represented in an eight bit pattern length as:

2 - 11110000 - Dash pattern
3 - 11100100 - Dash dot pattern
4 - 10101010 - Dot pattern
5 - 11101010 - Dash dot dot pattern
6 - 10001000 - Sparse dot pattern
7 - 10000100 - Asymmetric sparse dot pattern
8 - 11001000 - Sparse dash dot pattern
9 - 10000110 - Sparse dash dot pattern

The length of the units of the pattern elements is not specified, nor is the relationship of the lengths of the "dashes" and "dots." In raster devices, the unit dot may typically be the size of the physical pixel, but there is no requirement that this length be coupled to any absolutes of screen addressing. Therefore, only the "duty cycle" of the patterns, not their component lengths, is guaranteed by this standard.

Each device must have a certain repetition length for the line drawing patterns. This length is not standardized since a basic dot pattern will carry essentially the same information content independent of this repetition period. If the number of binary digits specified exceeds the implemented width <k>, then the implementor may choose to keep either the first <k> bits specified or the last <k> bits specified, at his option.

Conforming software should use only the standardized patterns (with multipliers to expand the distinguishable range of patterns) unless the details of the pattern register in the device being driven are known.

Patterns specified should repeat within the repetition length of the device. In this way predefined pattern P1 is identical to the custom pattern P1. The action taken by the device in repeating patterns not a submultiple of its internal pattern length is not specified.

The only defined suboption to the W(P()) command is M<m>, where the value <m> represents a multiplier to be applied to each element of the requested pattern, either predefined or custom. As noted above, the unit size of the pattern elements is not specified, but should be large enough to be easily distinguished, yet small enough to multiply well for scaled patterns. The multiplier value may be used to expand the pattern spacing, and thus achieve additional discernible line pattern types. Implementations must provide for multipliers as large as 16.

There is to be no interaction between the pixel vector multiplier specified by W(M<n>) and the pattern element multiplier W(P(M<m>)).

The line pattern is reset to solid and the pattern element multiplier is set to 2 when a valid S(A) is executed.

8.5.3.5 Writing Modes

As was noted at the start of this section, the only Writing Attributes required in Base ReGIS are those for the control of line pattern and pixel vector length. Extensions to ReGIS provide for other controls, among them the writing mode, that is, how new drawing actions affect the execution medium. Base ReGIS does not specify what writing mode must be present when these controls are not or cannot be implemented. The two most natural choices are either:

1. Overlay mode, the model that most closely resembles writing with pen on paper, or

2. Replace mode, a model which may be convenient for certain bitmap devices.

Refer to subheadings 8.7.2 Raster Extension and 8.7.2.4.1 Writing Attributes Extensions - Writing Mode Options. for a complete definition of writing modes. The guiding criterion to the device implementor is that visible output is generated.

8.5.4 VECTOR INSTRUCTION

The Vector instruction, keyletter "V", draws straight line segments connecting the current drawing position and one or more specified positions. It uses bracketed position arguments or pixel vectors.

8.5.4.1 Arguments

Semantically, the vector instruction is exactly the same as the Position instruction, with the exception that the V instruction can cause a visible image to be generated. The lines drawn by this instruction are subject to the current line drawing pattern selected by the W instruction and the pixel vectors drawn are subject to the current pixel vector multipliers selected by the W instruction.

The B and E options allow the user to define a closed polygon, though this is not the mechanism used in ReGIS to specify a filled polygon. Refer to the section on Position instruction options, and the section on the position stack, for detailed descriptions of the use of the B and E options.

The S option to the Vector instruction is included to provide symmetry with the open Curve instruction. An S option to a vector or position instruction saves a dummy entry on the position stack, consuming one entry of stack space. The matching E option to this S pops the dummy element from the stack without other effect. This dummy element will cause the same action as would be achieved by trying to pop from the empty stack, except that any data below the dummy entry is still valid and available. The dummy entry implies NO ELEMENT, as contrasted with the empty bracket ([]), which implies action at the current position. This allows vector and open curve sequences to use exactly the same argument list to achieve parallel results. Note that the position stack available to the Vector instruction is not available to the Curve instruction. These parallels are parallels of semantics, not of shared or shareable state. Refer to the C(B), C(E) and C(S) sections later in this standard for descriptions of closed and open curve actions.

The W option allows the writing attributes to be temporarily set. After completion of the Vector instruction, the value of the writing attributes returns to those they had before the execution of the V instruction.

8.5.5 CURVE INSTRUCTION

The Curve instruction, keyletter "C," draws circles, arcs of circles, and interpolated curve sequences. "Interpolated Curve Sequence" means a curved line image of varying radius of curvature such as a draftsman using French curves or similar aids would draw.

A Curve construct in Base ReGIS has been included for the following reasons:

a. Each graphic device knows its abilities better than the software driving the device, and thus is better able to select an optimal quality versus performance tradeoff. This avoids the problem which arises when software approximates a circle by specifying an 18 sided polygon for use on a low resolution device, and produces an 18 sided polygon precisely when the application is run on a high resolution device.

b. With the assumption that all devices will have local processors, there is no longer any significant complexity to having a local curve capability.

c. The availability of a local curve capability can substantially reduce the number of characters transmitted to the device.

d. The inclusion of a curve capability gives ReGIS a degree of completeness in terms of incorporating the "rule and compass" drawing primitives.

Circles vs. Curves

While both circles and general curve constructs are defined as parts of the same ReGIS commands, differences are great enough in goals and actions of the commands that they should be considered as separate commands for purposes of explanation. Hence, the description of the actions of the argument types below is divided into a circle set, a curve set, and a common set. The results of mixing circle options (C and A) with curve options (B, S, and E) in the same command are not specified, and should be considered an error.

8.5.5.1 Position Arguments

The meaning of a position argument depends upon whether or not a curve position block has been selected.

8.5.5.1.1 Circles

Outside of a position block, position arguments mean "draw a circle" or, if the arc-angle option has been selected, "draw an arc of a circle." The instruction form C[<position>] draws a circle with the current drawing position as the center and [<position>] a point on the circumference of that circle. The drawing position is left at the center after the circle is drawn; that is, the drawing position is unchanged.

Thus, consecutive [<position>] arguments to a C command causes concentric circles to be drawn. The (C) option inverts the relationship of the initial drawing position and the position arguments.

If the arc-angle option (A<ang>) is used, an arc of a circle is drawn starting at the point on the circumference and ending <ang> degrees from that point.

8.5.5.1.2 Curves

Within a position block, position arguments refer to points on a curve through which an interpolated curve is to be drawn. S and B represent the two types of curve interpolation sequences. The S option specifies an open end point curve. The B option specifies a bounded curve, in which the end point of the interpolated curve is drawn back to the beginning point to form a smooth closed curve.

8.5.5.2 Numeric Arguments

Numeric arguments to both forms of the Curve command are interpreted as pixel vectors. The pixel vector specifier shall be interpreted as if it were a relative bracketed extent, and executed in the circle or curve context, as appropriate. For instance, C0 = C[+m] where m is the current pixel vector multiplier. If m = 10, then C0 draws the circle just as if it were C[+10]. The pixel vector multiplier should be set to a reasonably large value for these vectors to have worthwhile effect.

8.5.5.3 Quoted String Arguments

There is no meaning defined for any quoted string arguments to the Circle or Curve command in Base ReGIS.

8.5.5.4 Options

8.5.5.4.1 Circles

8.5.5.4.1.1 C(A<ang>) - Circular Arc

The circular arc option causes only a portion of the specified circle to be drawn. The current drawing position specifies the center of the arc, and a position argument or pixel vector argument specifies the start point of the arc. The arc is drawn <ang> degrees, counterclockwise for positive <ang>, clockwise for a negative <ang>. Arc directions, clockwise and counterclockwise, are referenced against the viewing reference, that is, as seen by the viewer. Inverting the address space by changing the sense defined in S(A) does not change angle orientation.

The accuracy to which <ang> may be specified is device dependent, It is based on the methods and computational resources available in the device. ReGIS devices are allowed to round angular specifiers to the nearest 10 degree increment, rounding up to the next 10 degree increment at the five degree point. Where consecutive arcs are a requirement, as for drawing pie charts and the like, careful applications should specify arcs rounded down to the next lower increment of 10 degrees, and complete the arc by one or more vector segments to more closely reach the idealized end point.

Should the absolute value of the specified angle exceed 360 degrees, the value executed will be plus or minus 360 degrees, as appropriate.

The drawing position is left at the end point of the arc if the (C) option was used. Otherwise, the drawing position is left at the center of the arc.

8.5.5.4.1.2 C(C) - Circumferential Circle

The (C) option to the circle command causes the relationship between the current drawing position and specified position arguments to be inverted. When the (C) option is in effect, the current position defines a point on the circumference, and a position argument or pixel vector specifier indicates the center of the circle or arc.

A circle command with the (C) option draws the circle with the current drawing position as a point on the circumference and [<position>] as the center. In this case, the drawing position is left on the circumference of the circle at the the end point of the circle drawing.

The (C) and (A) options to the Circle command may be used together, as illustrated by Figure 8-5. In this case, the drawing position defines the start point of the arc, and is left at the end of the arc when it is completed.

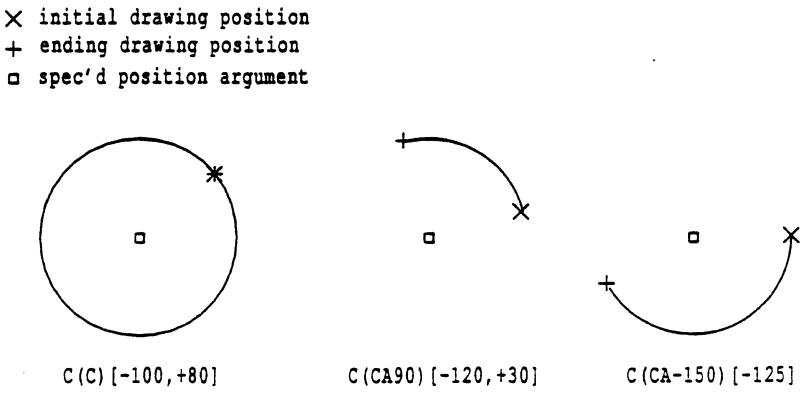


Figure 8-5: Circumferential Circle and Arcs

8.5.5.4.2 Curves

8.5.5.4.2.1 C(B), C(E), and C(S) - Curve interpolation

Either the option "S" (for an open curve sequence) or "B" (for a bounded curve) starts a general curve interpolation sequence. The sequence ends with the "E" (end) option. There must be no intervening circle or block structured P or V instructions within the range of a curve begin-end sequence, nor shall C(S)...(E) or C(B)...(E) sequences be nested. (Curve and circle operations may be present within a vector or position (B)(E) block, however.) Within these option specifiers, the position arguments (starting with the current drawing position) represent points through which a smooth curve image is drawn.

An open or closed curve interpolation must be completed within one invocation of the C command, or the state of the interpolation may be lost. Note that in the command stream

C(B)[]C[]C[]C[]...

the intent seems to be that a closed curve sequence be invoked. The occurrence of the second C in this stream returns the ReGIS parser to command level, however, voiding the actions of the (B) option and resulting in some number of concentric circles.

The only three elements allowed within a curve interpolation sequence, either open or closed, are:

1. Position elements (bracketed or pixel vector)

2. Options (some of which may have no meaning within a curve construct)

3. Text arguments (which cause a skip state and are ignored)

Any new command keyletter, even a reiteration of the C command keyletter, results in the abnormal termination of the curve context, and the curve state is lost. Termination of curve context by a synchronizing semicolon has the same effect. In either case the number of curve elements drawn is device dependent. Thus, conforming software should always properly terminate a curve construct to ensure consistent visible results.

The use of (W()) suboptions in curve interpolation commands is similarly device dependent. Given the allowed variability of curve interpolation algorithms, prediction of what writing mode or rendition will be coupled with what curve segment is beyond the scope of this standard.

The algorithms used and the drawing methods employed to execute open and closed curves are not prescribed by this standard. The intent of this standard is not to prescribe a class of curves by definition (as, say, "cubic spline" or "conic approximation"). The following constraints do apply, however, and may limit the practical choice of algorithms:

a. The curve must pass through every point specified in the argument list with the same accuracy that a vector would pass through them (zero-order continuity).

b. The slope of the curve must be continuous across every specified point (first order continuity). Furthermore, the slope of the curve at any given point in the argument list must be parallel to the line connecting the previous and subsequent adjacent points in the argument list.

c. Four or more collinear points must result in a straight line from the second to the next to last of these four points. This is a necessary result of constraints 1 and 2.

d. Intermediate computed points found for the curve must be the same regardless of the direction in which the points are specified, that is, whether the points defining it are given from 1..n or from n..1. This does not mean that the visible results must coincide, since this constraint does not impose a method of rasterization on the line segments which connect these intermediate points.

The slope constraint above leads to an indeterminacy of slope at the first and last points on an open curve. For a closed curve (option B), the point information at the beginning of the sequence must be retained by the ReGIS interpreter so that the closed curve will have a continuous first derivative at the end point. Determining the slope of open curves at the end points is handled in the following manner:

- The curve segments from the start to the second point and from the penultimate to the last point are not drawn. These point pairs (first and third, and second next to last and last) are used to define the slope of the curve at the second and penultimate points.

- The begin and end points may be visually included in the curve by using the "null" point syntax [] to indicate that the point is to be repeated in the sequence. In this case, the visual appearance of the curve at the end points may not be realistic, since the effects of doubling the end points bypasses the natural constraint of first order continuity and the intended shape of the curve.

As a result of these constraints and conditions, at least three position arguments in addition to the current cursor position must be given to result in a visible curve segment between the second and third positions (the first position being the initial drawing position). One additional curve segment is then drawn for each additional position argument.

For open curves, the current cursor position is always maintained at the last position argument given and thus leads the drawing process by one curve segment. For closed curves, the drawing position at the end of a closed curve command is not changed as a result of curve drawing. The value of the drawing position resulting from abnormal termination of a curve interpolation sequence, as by sending a parser synchronizing sequence, is not standardized and is thus device dependent.

Despite the definition of curve constraints outlined earlier in this section, ReGIS does not specify the exact algorithm by which devices will implement curve interpolation. The exact curve sequence drawn varies depending upon the algorithm used, but this is not considered a problem, since the essential information content is shown in spite of minor variations in the curve path. Devices may implement more than one curve interpolation algorithm; these algorithms may be user selectable by option extensions.

8.5.5.4.3 Common

8.5.5.4.3.1 C(W) - temporary Write options

The drawing style of the lines which make up the curve or circle are controlled by the settings of the Write attribute controls. These settings may be temporarily modified by the use of the W suboption, just as in the Vector instruction.

8.5.5.4.4 Volatility of C Instruction Options

All options to the Curve and Circle commands are temporary, and apply only within the scope of the instance of the C command keyletter in which they appear. Specifically, arc and circumferential circle specifiers apply to all position or pixel specifiers that follow it until:

a. These options are respecified.

b. A new keyletter is encountered.

c. The interpreter is synchronized to command level.

8.5.6 TEXT INSTRUCTION

The graphic text instruction, keyletter "T", draws strings of characters starting at the current drawing position.

Each ReGIS device must have a built in character definition and writing capability. The default alphabet for writing is a full 8-bit encoded printing set, typically ASCII or some other base set in GL, and some supplemental set in GR. The recommended default and initial contents of the in-use table is ASCII in GL and ISO Latin-1 Supplemental in GR. For products targeted for use in areas where these sets are not reasonable defaults, product-dependent initial and default values are allowed.

Each character of the text string is written by placing the visual image of the referenced character at the current drawing position such that the upper left corner of the character cell coincides with the current drawing position. After writing the character, the drawing position changes to a point which would be the logical position for writing another character, normally horizontally a distance a little more than the visual width of a character. In an extended system, the color or intensity of text is determined by the setting of the Write attributes, W(I), just as are lines and curves.

ReGIS does not specify the size of the text, its spacing, or the number of rows or columns of text which must be supported on a given display surface. ReGIS text is meant to be used annotatively, that is, to supplement the drawing process, not as a high density verbal display medium.

Only the capabilities expected of low resolution devices are specified in Base ReGIS. Variable character size, orientation, aspect ratio, and other factors are standardized in ReGIS extensions. Standard (default) character sizes are not defined by the ReGIS standard. Thus the drawing position resulting from the execution of a text string cannot be determined without knowledge of the target device or the detailed specification of text size and escapement. The quality of character representation at this level is also assumed to be poor and typically drawn as a sequence of short vectors or as a pixel pattern.

ReGIS does not have the facility to choose among alternate typestyles (fonts) that may be stored in a device. ReGIS extensions do support the idea of down-line loadable character sets, which may be used to implement such a capability.

8.5.6.1 Position Arguments

A position argument to the Text command specifies character escapement, that is, how much the drawing position moves after a character has been drawn. This measure is expressed in units of the imposed coordinate space. It defaults to rightward movement equal to the current width of the character display cell, and no vertical movement. This character to character movement defines what is known as "character path," and describes the orientation of the entire character string. It should not be confused with, nor does it affect, the orientation of the character cell baseline, which is not controllable in Base ReGIS, but is covered in ReGIS extentions. The presence of a bracketed argument to the T command is non-positional. Refer to the subheading 8.4.5.2 Use of Bracketed Extents for Non-Positional Information. This value is always considered relative, regardless of the presence or absence of sign characters on the X and Y parts. Any missing elements are interpreted as +0, that is, no escapement in that dimension. If character size options are supported, they also set character escapement as a side effect.

Character escapement is returned to its default value when a valid S(A) operation is executed.

8.5.6.2 Numeric Arguments

Numeric arguments to the Text command are interpreted as a form of pixel vector, and cause special motion of the drawing point. This motion is equal to half the current character size. When character height and width are not equal, this movement will take on the proportions of the character cell.

Movement is executed only when the pixel vector digit is encountered, and does not affect, nor is it affected by, the character escapement value. The direction of the 0 pixel vector is parallel to, and in the direction of, the character baseline. (This reference is significant in ReGIS extensions, where the character baseline is not constrained to be parallel to the horizontal axis.) Numbers 1 through 7 step in equal increments, counterclockwise, as for normal pixel vectors. This argument can be used for superscripting and subscripting in text strings.

8.5.6.3 Quoted String Arguments

Quoted string arguments to the Text command provide the character data to be displayed. Character patterns are retrieved from the stored alphabet (the last alphabet selected by T(A) if multiple alphabets are supported). A0 is the default alphabet, and is the only alphabet required in Base ReGIS. The contents of the A0 may be affected by state set external to ReGIS.

Only within quoted strings used as arguments to the Text command are any control characters interpreted. Only four codes are used: carriage return, line feed, backspace, and horizontal tab:

1. The carriage return code resets the drawing position, both horizontally and vertically, to the value it had at the start of the current quoted text extent.

2. The line feed code moves the drawing position down one character height unit vertically, and saves this new position as that which will be restored should a carriage return code be subsequently encountered in the current quoted extent.

3. The backspace code moves the drawing position back one character escapement step, both horizontally and vertically.

4. The horizontal tab code moves the drawing position forward one character position, just as a space character would, but without drawing anything. The distinction between drawing a space and just moving the drawing position is significant when different drawing modes are considered. A replace mode space executes the background color into the drawing medium, while an overlay mode space or a horizontal tab in any mode do not.

8.5.6.4 Options

8.5.6.4.1 T(A) - Alphabet selection

Arguments to the Text instruction select among possible alphabets of characters. Extensions to ReGIS specify loadable alphabets. The only selection mechanisms in the ReGIS Base are among pre-loaded alternate character sets.

The T(A) command has suboptions R and L, each of which takes a quoted string argument, the tail of the reformatted ANSI designating escape sequence. The resulting format is

T(A<n>(L"<designating tail>"))

to select a character set into the left half, and

T(A<n>(R"<designating tail>"))

to select a character set into the right half.

Under the formats established by ISO 2022, a sequence of the form

ESC I ... I F

is a character set designating sequence if the first intermediate (I) is from certain code positions in column 2 of the code table. This first intermediate identifies the type of character set and its G-set (G0, G1, G2, or G3) assignment. By replacing this particular intermediate with a character from column 3 of the table, the G-set assignment is ignored but identification of the character set type is retained. This technique is used in the DEC Common Font File Format (CFFF) for identifying character sets where G-set assignment is not a factor. The "tail" of the resulting sequence is everything except the introducing ESCAPE code. Without the ESCAPE code, the tail is clear text, and can be used as a string-type argument.

The designating tail in the quoted argument to L and R must start with a legal first intermediate as noted in either of the two tables below. An incomplete or illegal tail will be ignored. There may be multiple tails in a single quoted argument, as multiple tails may be required to couple a character set selection with a version number tail, or to list alternative designations in a single command.

The following table lists the legal first intermediates (the "column two" intermediates) used in the normal character set selection scheme.

Code	Graphic	Use
2/4	$	multi-byte set
2/6	&	version number
2/8	(	94 character set, G0
2/9	)	94 character set, G1
2/10	*	94 character set, G2
2/11	+	94 character set, G3
2/13	-	96 character set, G1
2/14	.	96 character set, G2
2/15	/	96 character set, G3

This table lists the replacement intermediates (the "column three" intermediates) and their resulting character set assignments.

Code	Graphic	Use
3/0	0	94 character set
3/1	1	96 character set
3/2	2	multi-byte 94 character set
3/3	3	multi-byte 96 character set
3/4	4	complete code character set
3/5	5	character set revision

ReGIS has a single 8-bit in-use table with left and right halves. Use of these "new" intermediates from column 3 allows the categorization implied by G0-G3 to be ignored. Thus, the character set shifting controls, such as Shift In (SI), Shift Out (SO), Single Shifts (SSn), and Locking Shifts (LSn, LSnL and LSnR) are not supported. This is true regardless of which tail format is chosen.

The character sets listed in the following table are required to be supported in all ReGIS implementations. The indicated designating tails are the column three tails, and are the preferred tails for use by new ReGIS creators. The older column two tails are allowed, but their use should be phased out.

Tail	Name of set/Source standard
00	DEC Special Graphics
04	DEC Dutch NRCS (optional)
05	DEC Finnish NRCS
07	DEC Swedish NRCS
09	DEC French-Canadian NRCS
0=	DEC Swiss NRCS
0>	DEC Technical
0A	British NRCS/BS 4730
0B	ASCII/ANSI X3.4-1986
0I	JIS Katakana/JIS X 0201
0J	JIS Roman/JIS X 0201
0K	German NRCS/DIN 66 003
0R	French NRCS/AFNOR NF Z 62-010 [1973]
0Y	ISO Italian NRCS
0Z	ISO Spanish NRCS
0'	Norwegian/Danish NRCS/NS 4551-1, DS 2089
0%5	DEC Supplemental/DEC STD 169
0%6	DEC Portuguese NRCS
1A	ISO Latin Alphabet Nr 1 Supplemental

"Latin-1" is an allowed synonym for "Latin Alphabet Nr 1." DEC Dutch is not required to be supported in ReGIS, but '04' is the approved tail if it is supported.

ReGIS devices supporting the Kanji Display Extension must also support the following character sets and designators.

Tail	Name of set/Source standard
20	DEC Kanji 1978
22	DEC Kanji 1983
2@	JIS Kanji 1978/JIS X0208-1978
2@	JIS Kanji 1983/JIS X0208-1983

The default state of the ReGIS character set table shall be that specified by the command

T(A(L"0B",R"1A")).

This is equivalent to the 8-bit set Latin-1.

The command

T(A(L"0J",R"0I"))

puts JIS (Japanese) Roman in the left half and JIS Katakana in the right half of the ReGIS in-use table.

This selection methodology extends to multi-byte support.

The command

T(A(L"(J",R"$+@"))

uses the column two tails and selects JIS Roman into the left half and JIS X0208 (Kanji) into the right.

The command

T(A(R"5A1A"))

uses a multiple tail to modify the selection of a multinational character set into the right half by preceding the Latin-1 Supplemental tail with a version identifying tail. The above sequence would specify the second revision of the ISO Latin-1 set. [This is just an example - as far as is known, the first revision hasn't even been considered yet.]

8.5.7 REPORT INSTRUCTION

The Report function is included to allow the user software to interrogate the ReGIS device for state elements of interest.

8.5.7.1 Position Arguments

There is no meaning assigned to bracketed position arguments to the Report command in Base ReGIS.

8.5.7.2 Numeric Arguments

There is no meaning assigned to numeric arguments to the Report command in Base ReGIS.

8.5.7.3 Quoted String Arguments

There is no meaning assigned to quoted string arguments to the Report command in Base ReGIS.

8.5.7.4 Options

8.5.7.4.1 R(M(x)) - Report Macrograph state

The contents of an individual macrograph is reported back to the controlling program by specifying the M option to the R command with a suboption keyletter which is the same as the name of the desired macrograph.

The format of the report is

@=<name><string>@;

which parallels the format of the macrograph definition, with the substitution of an equal sign (=) for the colon (:).

In some systems, unrestricted allowance for reporting of macrograph contents will sometimes result in system security problems. Devices are therefore allowed the freedom to restrict macrograph reports. The preferred fallback is the empty report, that is, just the terminating CR character.

An alternative is to report the macrograph as specified above, replacing each character in the macrograph with SPACE or some other printing character. This allows the user to verify the length of the stored macrograph, and might indicate the proper storage of the macrograph in question.

The macrograph storage allocation may be returned by specifying the command

R(M(=))

which returns a quoted string of the form

"<free>,<total>"

where <free> is the minimum guaranteed amount of space, in characters, still available for macrograph storage, and <total> is the amount of space, in characters, allocated by the device for macrograph storage. It is not required that "<free>" be accurate to the character count level, as long as its value represents a lower bound on the measure of available space.

8.5.7.4.2 R(P) - Report current Position

The returned report is a bracketed position specifier of the form

[x,y]

representing the current drawing position in the imposed coordinate system. The parts, x and y, return in ReGIS-conforming numeric representations.

If the drawing position has been moved into absolute negative coordinate space, the format of the report includes a signed number indicating this. Hence, in position reports signed numbers represent absolute, not relative, locations.

8.5.7.5 Further Notes

All reports are terminated by a single carriage return character. Requests for unimplemented reports are acknowledged by the device's returning just a carriage return; this prevents device deadlocks by ensuring that an application will receive some response, even if the requested report cannot be generated.

Multiple options or suboptions to a single R command are interpreted as requests for multiple reports.

All devices must return a report for every option keyletter in the argument stream. Some forms of the report command define a report for every suboption letter as well, as is the case with macrograph reports.

In the case of unimplemented reports, the device may not be able to discern how many reports are due, especially if suboptions are involved (as in the case of the macrograph report). Hence, software should request reports singly and in turn, so that only one report response is outstanding at any time. This will minimize the chance of deadlock or phase error due to unanswered reports.

There are some classes of devices for which reports are not necessary or easily supported. Software to convert ReGIS files to some other graphic form, for example, might not have an identifiable return path for reports. Some devices are meant to be run spooled, and will not have the ability to provide reports in a timely manner, if at all.

Though the ReGIS Report command is part of the ReGIS Base, it clearly cannot be supported in all environments. Lack of support for Reports in such environments does not constitute a failure to conform to the ReGIS standard.

8.5.8 FILL INSTRUCTION

The Fill instruction, keyletter "F", defines bounded areas by their edges. The command colors in only and entirely, subject to area texture or pattern in effect, the area (or areas) bounded by the edges presented.

8.5.8.1 Position Arguments

There is no meaning assigned for bracketed position arguments to the Fill instruction in Base ReGIS.

8.5.8.2 Numeric Arguments

There is no meaning assigned for numeric arguments to the Fill instruction in Base ReGIS.

8.5.8.3 Quoted Text Arguments

There is no meaning assigned for quoted text arguments to the Fill instruction in Base ReGIS.

8.5.8.4 Options

8.5.8.4.1 F(C, P, V) - define filled area

The actions of the P, C, and V commands inside the option extent of a Fill command mimic the actions of the Position, Curve/Circle, and Vector commands at command level. Instead of executing positions and drawing lines to outline an area or shape, these options define the extent of whatever figure would be drawn at command level, and cause that area to be executed as a filled area.

Each P, C, or V option to the F command takes whatever subarguments that would apply at the command level to define a shape and use that same argument in the context of executing the figure as filled, subject to some of the following constraints.

a. Allowable suboptions V and P are stack controls V(B,E,S) and P(B,E,S); for C, closed and open curves, C(B,S,E), and arcs and circumferential arcs and circles, C(A,C).

b. Write suboptions are allowable for all of V, P, and C in contexts where they make sense, as described in later sections.

c. Allowable arguments for V, P, and C are bracketed position specifiers and pixel vectors.

The definition of a polygon boundary is determined by the edges described by the drawing options V, P, and C. These edges are determined by what would be drawn by the V and C options were they executed at command level. The P option modifies how position arguments to V and C may be interpreted, but no position specifier on a P option will be entered into the list of vertex list which describes the border. This implies that the current position in effect at the start of a polygon description is NOT automatically the first vertex of the defined polygon. If the current drawing position is meant to be that vertex, then an option specifier such as V[] (the empty brackets implying current position) must be the first executed position. The termination (closure) of the Fill command will close the polygon to this point. More explicit perhaps, is the form V(B) (with a matching V(E) at the termination) to apply the current position to start the polygon.

During polygon outline definition the current position is updated as if the V, P, and C options to the F command were being executed outside of the F context.

For Fill, no perimeter is drawn as a result of the boundary definition process or of the execution of the area defined. A perimeter, if desired, must be specified separately as a set of ReGIS commands (probably the same set, as macrographs are useful here).

Each device has some implementation limit on the number of vertices which can be stored before the filled figure can be executed. A conforming ReGIS device must provide for a vertex list with a capacity of less than 256 entries, but note that certain other implementation dependencies with regard to curve interpolation (noted below) could result in many fewer than 256 vertices available to software. (On the other hand, a device may save space in the vertex list by allowing consecutive vertices which map to the same physical pixel to be counted as a single vertex.)

If the number of vertices generated exceeds the implementation limit, any additional vertices are ignored and the figure is closed and filled as if it had been terminated at the implementation limit.

The remaining elements in the fill specification will be ignored. No further drawing will be attempted until terminating right parenthesis of the Fill option list is encountered or a synchronizing action is executed.

Polygon fill must accommodate non-convex and self-intersecting polygons. The interior of a polygon, that is, that area filled as a result of the polygon definition, is defined by the "even-odd rule," as follows. From any given point not on an edge of the defined polygon, consider a ray beginning at that point and extending in any direction to infinity. If that constructed ray

crosses an odd number of edges, then the specified point is inside the polygon, and it should take on the fill attributes of the polygon. If the ray intersects an even number of edges, the point is outside the defined polygon. Intersection of the ray with a vertex point or complete edge of the polygon provides information useful to resolve the even-odd rule only if the positions of vertexes adjacent to those intersected can be recognized as lying on the same side of the test ray or on opposites sides of it. If a ray intersects one or more vertex points, a new ray may have to be chosen. These rules suffice to distinguish the interior and the exterior of simple polygons, and also nested polygons, as may be described by the use of F(F), (a ReGIS Open extension for filling complex polygons).

The current drawing position is saved at the beginning of an F command, and restored at the end of an F command, whether any drawing takes place or not. This behavior allows some degree of compatibility with older devices that do not implement the F command, and therefore simply ignore it.

The state of the ReGIS position stack is not preserved by F commands. An implication of this activity is that position arguments may be passed into the polygon fill process by pushing them on the stack and popping them inside the fill process, and that position arguments may be passed out of the polygon fill process by pushing them inside and popping them after the Fill is complete.

Any ReGIS command string that changes the state of the position stack within an F command will not be compatible with devices that do not implement the F command. A device that does not implement the F command will not recognize the V, P, and C options to the F command in their proper context, and hence cannot recognize stack operations in the option list either.

The vertex list resulting from the interpretation of positional subarguments always represents a closed figure whether the drawing commands used to specify that vertex list represent a closed figure or not. If the commands used to specify a polygon do not represent a closed figure, for example, because of disconnected start condition on an open curve interpolation, the filling proceeds as if consecutive vertices had been connected by straight lines, possibly causing unexpected results.

Curve and arc interpolations may, depending upon the curve and circle algorithms chosen in a device, generate multiple vertices per specified point. Such intermediate points will have to be counted against the implementation limit of the vertex list of the polygon filling process if the device must save them to compute the filled area boundary. The implication of this is that devices may need an device-dependent number of entries in the vertex list to accommodate a given ReGIS area definition. Devices should provide a greater number of entries in the vertex list to mitigate the adverse effects of such curve algorithm dependencies.

Open curves used to define polygon boundaries must abide by the constraints for such curves defined in the section on the Curve command. For such curves, the current position at the beginning of open curve interpolation does not generate any vertices, but helps determine the slope of the curve at the next specified point. The first specified point does not generate any vertices, since it is the beginning point of the curve. Subsequent specified points then generate some number of intermediate vertices each, as in normal (unfilled) curve interpolation. The last point specified does not generate any vertices; it determines the slope of the curve at the next-to-last point specified. Thus the total number of vertices generated when n points are specified for open curve interpolation is (n-2)*m, where m is the number of intermediate points per specified curve segment.

In order to connect different types of open figures, such as V and C(A) sequences, it is sometimes necessary to use the P option to reset the current drawing position. Repositions via P may also be needed when it is desired to draw both a polygon and its outline using the same command string.

Temporary write options may be specified as options of the of the F command itself, F(W...), or as temporary write suboptions to the C option, F(C(W...)), or V option, F(V(W...)). These forms are all equivalent and all supersede each other. Temporary write options are canceled by a new command keyletter.

A right parenthesis at the option level terminates an option list and causes a filled polygon to be drawn. The filled polygon is drawn using the write options (including temporary write options) in effect at the time the right parenthesis is encountered.

The only element of the write options which takes effect at the time it is encountered, rather than at polygon execution time, is the pixel vector multiplier, which conditions any pixel vectors used in defining the polygon boundary.

The area texture used to fill the polygon is variable by product classification. ReGIS devices that support the Raster Extensions use the interior style set by the area texture argument to the W(S) option, the same option used to enable shading. The "snapshot" of attributes established by enabling shading applies to polygon fill. An area texture is established by invoking shading and disabling it, as by the command fragments W(S1,S0) or W(S"x",S0). See subheading 8.7.2.4.2.3, W(S) - Shading for a description of the shading snapshot.

Devices that do not support the Raster Extensions may elect to fill with a solid color or intensity, or they may define a set of hatch patterns if solid fill is not appropriate.

If a synchronizing semicolon is encountered (or external resynchronization occurs) before the terminating right parenthesis, the F command is aborted and no drawing takes place.

8.6 THE EXTENDED LOGICAL GRAPHICS DEVICE

This section describes extensions to the Logical ReGIS Graphic Device required to accommodate specific graphical features not common to all devices. This section is an illustrative presentation to follow for extending ReGIS to device capabilities not described here. The case of raster CRT devices is covered in some detail to fully illustrate the process of extending ReGIS.

8.6.1 DIMENSIONAL DISPLAYS

"Dimensional displays" refers to the class of graphics devices in which the physical size of pixels has a repeatable meaning and allows the user a variety of pixel sizes. Examples of such devices include both paper and photographic plotting devices. ReGIS could be extended to accommodate these devices by allowing definition of the size of pixels in physical dimensions.

By knowing the number of pixels selected by the screen coordinate definition operation and their size, the device is then able to determine the size of paper needed to draw the image. The units of the physical measure should be optionally selectable by the user, with the metric system being a likely default.

These devices also will generally allow (and need) the width of line segments to be user controllable. This feature has been included in Open Extension to ReGIS as an option to the writing attributes instruction.

8.6.2 GRAY SCALE AND COLOR

The simplest model of color capability is that of a plotter device using different colored pens to draw on different colors of paper. In this case, the paper represents the background color and the pens represent the foreground colors. Monochrome devices often have the ability to show varying intensity levels of light, called gray scale. More complex devices, such as high resolution color raster devices, have the ability to present a full range of color hues in addition to the gray scale (intensity) capability.

8.6.2.1 Selection Scheme

To accommodate this broad range of capabilities, the extended ReGIS logical device adopts a scheme to allow the selection of color and gray scale in a simple manner for simple devices, but still has full access to the broad spectrum of attributes of the most capable of the devices.

a. Each pixel has associated with it a foreground and a background "intensity" attribute which are user controllable. The differentiation between foreground and background is most clear in the case of the plotter, and may have little meaning in a multiplane raster color device.

b. For simple color devices, the scale of eight intensity values is interpreted as the three primary colors, the three complementary colors and the black and white intensity values.

   Mappings between gray scale and color should follow the conventions which have been established for photographic and broadcast television work. These are summarized as:

Gray Scale	Color Value
0	Black (dark)
1	Blue
2	Red
3	Magenta
4	Green
5	Cyan
6	Yellow
7	White

   This mapping is based on the standard RGB to luminance level transformation used in the North American television industry (NTSC color standard). The NTSC RGB to luminance equation is

$$Y = 0.59*G + 0.30*R + 0.11*B$$

   where Y is the luminance value, and G, R, and B are the green, red, and blue components of the color drive, respectively. When normalized to the range 0 to 7, the equation becomes

$$I = 4.1*G + 2.1*R + 0.8*B$$

   where I is the resulting gray scale intensity for that range. The coefficients (rounded to 4, 2, and 1, respectively) lead to the encoding of the table above.

c. For devices having a full range of color capability, the Hue-Lightness-Saturation system of color definition is adopted. In summary, this system requires three possibly floating point numbers to be used in defining a specific intensity parameter value. These are:

- Hue (H) - the hue of the color expressed as an angle on the color wheel,
- Lightness (L) - the relative brightness of the color expressed as a percentage of full brightness, and
- Saturation (S) - expressed as a percentage of the fully saturated hue.

The intensity parameter may be applied independently to the foreground and to the background of the graphic image. Devices have extreme ranges of implementation of the intensity attributes. ReGIS does not define how many colors or intensity levels must be provided by an implementation which offers color or gray scale support.

8.6.2.2 Color Specification

There are two methods of specifying color or intensity in a graphics device: color by index, and color by value. Color by index is often used in multiplane bit map displays, or in hard-copy devices whose pens are selected by number. Color by value is used by implementations which have limited knowledge of the drawing hardware being used, but which need to specify colors or intensities to differentiate image components. ReGIS supports both color by index and color by value.

8.6.2.2.1 Color by Index

Virtually all graphics devices actually implement an indexed method of color selection. This index is translated into visible color by some hardware component, which may or may not be modifiable by user commands. Video devices, for example, translate the index value stored as bit map contents into beam intensity or color gun selector by a digital to analog converter and/or a multiplexer. If the converter/multiplexer is hard-wired, the translation is not user modifiable, and the translation from index to color is fixed. If the converter/multiplexer contains an extra stage of indirection to allow the translation to be dynamic, the device is said to have an output map. If this map can be used to modify the translation of index to color under user control, it is a truly dynamic system. In any case, ReGIS requires that the mapping of index values to visible components be known to the implementation. Colors by index may then be executed into the display medium directly. Color by value specifications are referenced against the index to color translation hardware (or software/firmware translation table) and a resulting index deduced. The index is then executed into the display medium as if it had been directly supplied.

In the case of an unsupported direct color or intensity specifier, the color map, whether hard-wired or user modifiable, should be searched to find the closest available color or intensity specifier. The "closest match" indicates the index value to be used. In monochrome systems, this "closest match" decoding is fairly simple, and either rounding or truncation may be used. The approach taken must be documented.

In color systems, the concept of "closest match" is more complex. For example, if a degree of orange is requested but not supported, the goal should be to present some shade of red or yellow to approximate this request. If only shades of blue and green are available, however, the approximation has no clear solution Efforts should be made to prevent mismatches in requests for a foreground from collapsing to the background. If color mismatches are too great, matches based on the monochrome mappings of those colors might be used. The approach to be taken in such cases is device and device class dependent.

In general, specification by index is implemented in ReGIS by the form

I<index>

where I is the suboption identifier for intensity for both the Screen command, where it sets the background color, and for the Write attributes command, where it sets the foreground. The range of values for <index> is device dependent.

When an index exceeds the bounds of the device, the unimplemented index must be mapped back into the supported range. In systems which support only one visible color or intensity, any non-zero index should map to the visible value, and the zero index should remain zero. In systems with more than one bit of index specification, a modulus scheme should be used, wrapping out of bounds values back into the supported range in a non-monotonic mapping. For instance, given a 4-intensity implementation and an 8-intensity stored picture, the following mapping would result:

Ask	Get
0	0
1	1
2	2
3	3
4	0
5	1
6	2
7	3

The device class may determine whether the mapping is purely modular, or whether some other approach is more appropriate.

For example, in a multi-plane bit map, pure modulus arithmetic is the most straightforward. A three plane system can accept indices in the range 0 to 7, as noted in the example table above. Mapping into these three planes by accepting only the low three bits of values greater than 7 is acceptable, though collisions between the wrapped foreground and the background may occur. On the other hand, a pen plotter is less likely to present such a problem, since the color of the paper is not likely to be the same as one of the loaded pens.

8.6.2.2.2 Color by Value

Specification of color by value in ReGIS is more complex than color by index. Not only are colors more complex, but ReGIS supports two types of color specification, RGB and HLS.

1. The HLS form is:

   I(H<ang>L<pct>S<pct>)

   where I is the option to Screen and Write attributes commands as noted above, and H, L, and S are the HLS components noted earlier; <ang> is expressed in degrees, 0 to 360; and <pct> in percentage values, 0 to 100.

   A hue angle of zero degrees specifies blue, 120 degrees specifies red, and 240 degrees specifies green. The additive complementary colors magenta, yellow, and cyan, are positioned at 60, 180, and 300 degrees, respectively.

2. The RGB form is:

   I(x)

   where x is a single letter color specifier for the RGB color alphabet: R, G, B, Y, C, M, W, or D for red, green, blue, yellow, cyan, magenta, white, or dark, respectively.

Color suboptions must be completed within one suboption (one parenthesized range), and RGB and HLS color specifiers must not be mixed in the same suboption. RGB specifiers must not be quantified, nor may RGB specifiers be mixed to define color mixing. For example, the specifier RG does not imply Y.

HLS specifiers have the following defaults:

a. If H or S is specified, but L is not specified or not quantified, L defaults to 50.

b. If H is specified and quantified, and S is not specified or not quantified, S defaults to 100.

c. If H is not specified or not quantified, S defaults to 0; H has no default.

d. If none of H, L, or S is present, there is no default, and no action regarding that color specifier is taken.

   This rule will allow for extensions to the color specification alphabet by standardizing as yet unspecified suboption letters; the "no action" requirement protects backwards compatibility.

Specification by index provides the most control over the device where the hardware use of the index can be known to the user, as in specifying plane-by-plane bit map contents. Specification by value provides the greater degree of transportability where the goal is to differentiate information and where the exact mapping of device capabilities is not known beforehand.

8.6.3 TEXT ATTRIBUTES

Text attributes include a wide variety of features associated with the presentation of textual characters, including (but not limited to) any combination of the following:

a. Variable character size including independent width and height adjustment.

b. Variable character spacing including proportional spacing and letter-spacing.

c. Angular orientation of characters and independent angular character spacing.

d. National language character sets.

e. Alternate representation fonts (Gothic, Futura, and so forth).

f. Superscript and subscript capability.

g. Overstrike (underline, APL characters, and so forth).

ReGIS can be extended to support such features in extension sets. The user should recognize that the degree to which a specific device supports these characteristics is extremely broad and is therefore one of the areas of image generation with the least portability.

To support the majority of these features, an extended ReGIS device may support the definition of the following extended logical device parameters:

1. WIDTH - The width of the hypothetical parallelogram in which a character is written.

2. HEIGHT - The height of this parallelogram.

3. DANG - The direction of the character base line measured as an angle in degrees relative to the horizontal axis.

4. HANG - The direction of the height side of the character parallelogram measured relative to the baseline direction, such as relative slanting of characters, as for an Italic representation.

5. TDX - The relative horizontal spacing of two characters.

6. TDY - The relative vertical spacing of two characters.

7. FONT - A scalar parameter used to identify which of several possible character representations is currently in use.

8.6.4 AREA ATTRIBUTES

This class of extensions refers to the general capability of many devices to associate attributes to areas bounded by ReGIS primitive line images. The visual attributes may include shading patterns, similar in concept to line patterns, and color and intensity variation.

These capabilities are covered syntactically by other ReGIS extensions. ReGIS distinguishes three approaches to identifying the boundaries of an area:

1. SHADEd areas are defined to be the difference between two not-necessarily bounded vector or curve sequences.

2. FILLed areas are defined by either a connected sequence of vectors or a closed curve, or by a combination of connected vectors and open and closed curves.

3. FLOODed areas are defined by boundaries existing in an image without regard to what sequence of commands generated the elements of the boundary.

The first case corresponds to the typical approach used to represent data as a "histogram" image. The second case corresponds to the common concept of bounded surface. The third case corresponds to after-the-fact coloring in of areas not necessarily bounded by a single command.

8.6.5 DYNAMIC ATTRIBUTES

The area of dynamic attributes covers a broad spectrum of features which are illustrated at one extreme by "blinking" attributes and at the other extreme by a fully animated cartoon. The common point in ReGIS for such capabilities is that the foreground/background selector function may vary as a function of time. Using this model, the typical blink attribute is interpreted as a time-based modification of foreground and/or background visual attributes, while more complicated dynamic attributes could be considered as selectors acting on separately accessible foregrounds.

8.7 REQUIRED EXTENSIONS FOR RASTER DEVICES

This section describes a standard set of ReGIS instruction extensions to be used with raster CRT graphics devices. All commands under heading 8.5 Base ReGIS Instruction and all commands under this heading must be implemented as specified for a device to be certifiable as a ReGIS Raster Device. Heading 8.8 Open Extensions to ReGIS describes further extensions to ReGIS, some of which are applicable to raster technology, but which are not part of the required raster set.

The additional capabilities addressed here include:

a. Background and foreground gray scale and color intensity attributes,

b. Image dependent writing attributes (complement), and other extended writing modes (replace, negate, and erase),

c. Area attributes for bounded and unbounded areas, and

d. Text attributes and user definable characters.

All Base ReGIS commands apply to raster devices. Some raster commands modify or extend these base commands. Defaults for such new elements approximate as closely as possible the effects that would be achieved if the extension were not present.

The original intent of ReGIS Raster Extensions was to capture the capabilities found in most raster video devices. Since the original adoption of ReGIS, monochrome bitonal raster printers have become a major presence in the industry. The ReGIS Raster Extensions focus on video-like capabilities, but the limitations of bitonal hard copy is considered as well.

8.7.1 SCREEN INSTRUCTION

Raster scan extensions to the screen instruction allow setting a background intensity (color and/or gray scale value) and the introduction of a time base for ReGIS execution.

8.7.1.1 Position Arguments

There is no significance to the presence of a bracketed position argument in the Screen Instruction Extensions for Raster ReGIS.

8.7.1.2 Numeric Arguments

There is no significance to the presence of a numeric argument in the Screen Instruction Extensions for Raster ReGIS.

8.7.1.3 Quoted Text Arguments

There is no significance to the presence of quoted text arguments in the Screen Instruction Extensions for Raster ReGIS.

8.7.1.4 Options

8.7.1.4.1 S(A[][]) - Screen Address definition

Display address definition in Raster Extensions works as it does in Base ReGIS, but sets to defaults the following Raster Extension state elements:

a. The device writing mode is set to its device default. This is overlay, if possible, with replace as second choice. Negate mode is disabled.

   Equivalent explicit command: W(V,N0) or W(R,N0)

b. Background color (or gray value) is set to its device dependent default; this default is visually distinct from the default foreground color or gray value.

   Equivalent explicit command: typically, S(I0); alternatively, S(I(<default color spec>))

c. The foreground color (or gray value) is set to its device dependent default; this default is visually distinct from the default background color or gray value.

   Equivalent explicit command: typically, W(I<n>), n greater than 0; alternatively, W(I(<default color spec>))

d. The foreground mask is set to all planes enabled.

   Equivalent explicit command: W(F<n>) with n=(2**m)-1, with m the number of planes in the device

e. Shading state is set to disabled.

   Equivalent explicit command: W(S0)

f. The text attribute set is set to restore individual text defaults if T(E) is encountered before T(B).

g. Character baseline angle is set to horizontal.

   Equivalent explicit command: T(D0)

h. Character unit cell and display cell sizes are set to the device dependent standard size 1.

   Equivalent explicit command: T(S[]U[]), with standard size 1 contents of the bracketed extents

i. Character slant (italic) angle is set to rectilinear.

   Equivalent explicit command: T(I0)

j. The alphabet selected for drawing is set to alphabet 0.

   Equivalent explicit command: T(A0)

k. The alphabet selected for loading is set to alphabet 1, the first loadable alphabet.

   Equivalent explicit command: L(A1)

The following states and stores are NOT AFFECTED by executing an S(A):

1. The character stores are not affected.

2. Character set names, as set by L(A'<name>'), are not affected.

8.7.1.4.2 S(I<n>) Or S(I(<color specifier>)) - background Intensity

The intensity parameter in the screen instruction uniformly changes the background intensity of the graphic image. The result is the same as loading a certain colored piece of paper into a plotting device. This parameter may be changed during the process of drawing the image to define the color value of the "off" pixels in a line pattern, or the background color to the rectangle which

encloses a character, but if transportability is to be maintained, no essential information content should be placed in such a dynamically defined background attribute. In either case, the action of a screen erase operation is to use the then current background intensity to uniformly define the background color and/or gray scale.

Refer to subheading 8.6.2 Gray Scale and Color for the distinction between color by index (I<n>) and color by value (I(<color specifier>)), and for the comments on handling values of <n> that exceed the implementation limit.

The default value for background is device dependent, but it must be visually different from the default foreground color. This default value takes effect when a valid S(A) is executed.

8.7.1.4.3 S(T<n>) - Timer

This option connects timing information to graphical display. The numeric value <n> is the number of device dependent timer "ticks" for which ReGIS interpretation should be suspended. The intent is to allow presentation timing to be done at the end of the transmission latency path, in other words, at the device. The number of ticks <n> is measured from the time the S(T<n>) is interpreted by the device. This is not necessarily the time the command is sent or received. Note that S(T<n>) commands may be cascaded, as S(T60,T60,T60), for extended pauses.

The amount of time represented by each increment of <n> is device dependent, but it should be in the tens of milliseconds range. For video devices, <n> could most likely be measured in frame times. The target increment for timer values should be 20ms +/- 20% (15-25 ms). Devices must accept and properly execute values of <n> as large as 2000. At 60 ticks per second, this allows for pauses of up to about 30 seconds. Longer pauses are not expected to be used to correct for device latency, but for software timing purposes, and must be controlled by host software.

8.7.2 WRITING ATTRIBUTES INSTRUCTION

Raster extensions to the writing attributes instruction allow access to a wide range of capabilities of CRT devices, including color, gray scale, the simple blink dynamic attribute, image memory modification, and area filling.

8.7.2.1 Position Arguments

There is no significance to the presence of a bracketed position argument in the Writing Attributes Extensions for Raster ReGIS.

8.7.2.2 Numeric Arguments

There is no significance to the presence of numeric arguments in the Writing Attributes Extensions for Raster ReGIS.

8.7.2.3 Quoted Strings

There is no significance to the presence of quoted string arguments in the Writing Attributes Extensions for Raster ReGIS.

8.7.2.4 Options

8.7.2.4.1 Writing Mode Options

"Writing modes" are the detailed actions of execution into or onto the drawing medium. Some drawing modes are insensitive to the presence of previous image contents, others are intimately coupled to prior contents.

8.7.2.4.1.1 W(C) - Complement Writing

When writing, use ones in the pattern register or cell to complement the contents of the pixels present in the image. Ignore zeros in the pattern register or cell; ignore the current foreground and background intensity specifiers.

8.7.2.4.1.2 W(E) - Erase Writing

When writing, ignore the contents of the pattern register, and Erase each pixel to the background specifier if W(N0) is in force, or write each pixel with the foreground specifier if W(N1) is in force.

8.7.2.4.1.3 W(N<0 or 1>) - Negate Writing

When W(N1) is in effect, the bitwise sense of the linear or area pattern is inverted (complemented) during write operations. In raster devices which use dot matrices for character cells, the bit patterns of these cells are inverted. When W(N0) is in effect, the patterns are used normally. The default value, N0, is set when a valid S(A) is executed.

8.7.2.4.1.4 W(R) - Replace Writing

When writing, replace the current image by input image. Blanks in a dashed line pattern or zeros in a pattern cell erase (to the background specifier defined by S (I)) the area overwritten on the screen. Ones in the pattern write the foreground specifier.

8.7.2.4.1.5 W(V) - oVerlay Writing

When writing, overlay ("OR") input onto current image. Blanks in a dashed line pattern or zeros in a pattern cell have no effect on the area overwritten. Overlay is the intended default writing mode in all extended ReGIS implementations, and is set when a valid S(A) is executed.

Four of the writing modes (C = complement, E = erase, R = replace, and V = overlay) are mutually exclusive, each overriding all others; the last one specified takes effect. The fifth mode, N = negate, may be used in combination with other modes.
Figure 8-6 shows the effects of each of the writing modes. In each case, the solid block was written first, and the patterned block was written over it in the indicated mode.

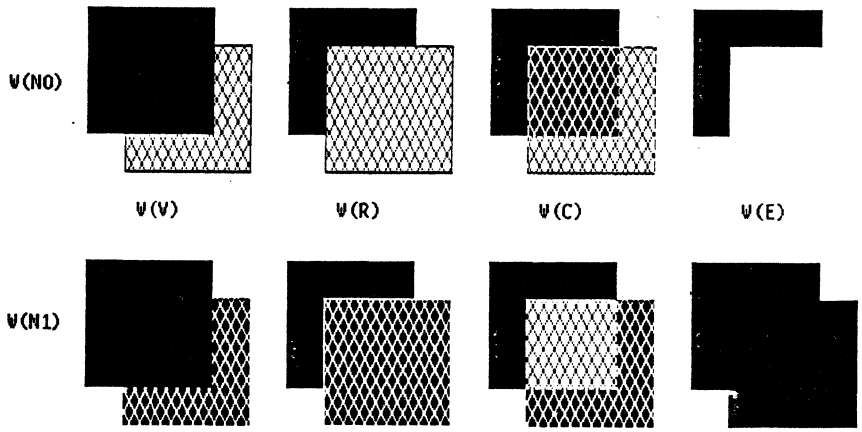


Figure 8-6: Writing Modes

8.7.2.4.2 Non-Writing Mode Options

8.7.2.4.2.1 W(F<integer>) - Foreground mask

Define the foreground field specifier. The bit encoded integer argument to F defines a mask which is used to enable all plane write and erase operations. Only planes represented by a 1 bit in the mask can be written. Ordering of bit assignments and bit planes affected is implementation dependent, but the numbering of bit planes for purposes of write masking and output mapping (S(M...)) must be consistent with each other. The intent of this command is solely to limit write access to a subset of planes in a multiplane device. It does not affect plane readback (S(H)) or those planes which can be moved by screen scrolling (S[]). The default value for this option is "all planes enabled," and this default is set when any valid S(A) is executed.

8.7.2.4.2.2 W(I<n>) or W(I<color specifier>) - foreground Intensity

Refer to the notes under the Screen command S(I) and subheading 8.6.2 Gray Scale and Color for details of <n> and <color specifier>. The default value for the foreground specifier is device dependent, but must be visible and distinguishable from the default value specified for the background by S(I). The default value is set whenever a valid S(A) is executed.

8.7.2.4.2.3 W(S<suboption><arguments>) - Shading

Shading is an area fill operation which is executed in conjunction with the Vector and Curve drawing commands. As indicated by Figure 8-7, shading is executed to a horizontal reference line.

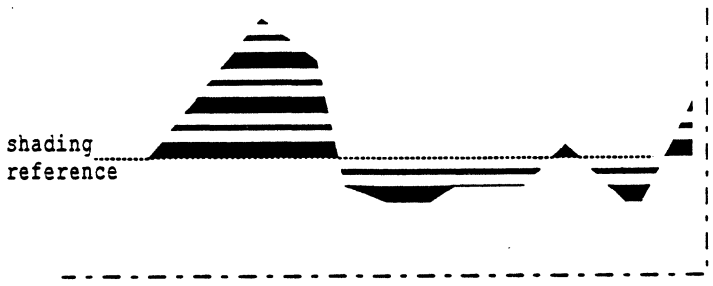


Figure 8-7: Selecting a Shading Reference Line

Shading is different from both fill and flood. Fill colors in a bounded area as it is defined. Flood colors in a bounded area after it is drawn, and is not supported in Raster ReGIS. Shading colors in a solid or textured area that is not explicitly bounded. Shading may be used to execute a fill-like function, but the burden of defining and controlling the area so filled is on the application using ReGIS.

A bracketed position specifier defines the location to which shading will be executed. In Raster ReGIS, it is possible to shade only to a horizontal line, and hence only the Y part of this specifier is significant. Open Extensions to ReGIS allow shading to a vertical line or to a point, so that either or both parts of the specifier may be significant. This mode selection is controlled by suboptions to this option. Pixel vector specifiers cannot be used as addresses for shading reference definition, as the numeric arguments to the S option have a different meaning. If this argument is not specified, the current drawing position is assumed.

A numeric argument to the S option is interpreted as a boolean switch. A non-zero value causes shading to take on the characteristics of the currently specified linear pattern. A zero value turns shading off. Shading is, by definition, an area operation. Use of a linear pattern to shade in an area requires adding a second dimension to the linear pattern. Shading is intended to be implemented by drawing lines from each distinct point on the drawing path, as described by Vector and Curve commands, to the shading reference. It is these lines, not the path line, which must take on the shading pattern. Furthermore, these shading lines should be registered so that the shading pattern is independent of the defining path. Figure 8-7 shows how the shading pattern reflects a vertical orientation of the chosen line pattern. Note how the vertical line next to the shaded area matches the pattern of the shaded area.

A single quoted character to the S option causes the shading pattern to be taken from the two dimensional pattern specified by the cell for that character from the alphabet currently selected by the A option to the T (text) command. The Text unit cell size currently in effect specifies the granularity of the shading pattern as it is applied to shading. Any Text option which affects the cell unit size is reflected in this granularity. These are T(U[]), T(M[]), T(H<n>), and T(S<n>), but not T(S[]). T(S[]) is excluded because it sets only the display cell, not the unit cell size. The text pattern is referenced to a known point in the display so that patterns of adjacent shaded areas match where the areas abut. To facilitate the development of patterns that match reliably when replicated left to right and top to bottom, only the top square part of the storage cell is applied in a shading context. Thus crosshatching patterns may be defined without consideration being given to the storage cell aspect ratio of the characters being used for shading.

When shading is enabled, with either a linear or a cellular pattern, certain state variables are "snapshot." These variables are used with the values they had at the time the shading was enabled, regardless of how these values change during the time shading remains enabled.

If shading is re-enabled without being disabled, then a new snapshot is taken of the state, and the old snapshot state is lost. This allows shading state to be set up and "locked," while the state variables for text are changed to allow interspersed execution of shaded drawing and normal text, such as for drawing labeled bar graphs. Execution of text is not affected by enabling the shading state. Even if shading is disabled, the snapshot established by its enabling remains in effect for use by polygon fill through the F command.

The following states are snapshot for use by shading when shading is enabled:

a. Linear pattern, set by W(P<n>)

b. Linear pattern multiplier, set by W(P(M<n>))

c. Text alphabet selection, set by T(A<n>)

d. Text size, as set by any of T(S<n>), T(H<n>), T(S[]), T(U[]), or T(M[])

The following two states are NOT part of the snapshot, and if they are changed while shading is enabled, they will be appropriately applied in subsequent path-oriented actions:

1. Writing mode, as set by any of W(E, V, R, C, N, W)

2. Writing colors, as set by either of W(I) or S(I)

Shading state is cleared (disabled) on any occurrence of W(S0), S(E), or a valid S(A). If shading is enabled as part of a W option to a drawing command, then the shading state is cleared when the drawing command is terminated by encountering a command keyletter at command level, or by synchronization.

8.7.3 TEXT INSTRUCTION

Extensions for graphics text include user definable character parameters and user definable alphabets.

8.7.3.1 Position Arguments

Position arguments to the Text command in Raster ReGIS are treated as character escapement values, just as in Base ReGIS.

8.7.3.2 Numeric Arguments

Numeric arguments are treated as character oriented pixel vector operations in Raster ReGIS, just as in the Base, but differ from the Base in regard to the effect of option extensions. In Raster ReGIS, a character and character string orientation option is present. The direction of the pixel vector operations in Base Raster ReGIS is referenced against this string direction, not against the absolute horizontal. Hence, Text pixel vector direction 0 is parallel to and in the direction of the character baseline, direction 2 is perpendicular the base line (and NOT affected by the italic angle), direction 1 is forward and up, and so on.

8.7.3.3 Quoted String Arguments

Just as in the Base, the text arguments to be displayed are passed as quoted string arguments, and the four ASCII control codes (carriage return, line feed, horizontal tab, and backspace) are interpreted by ReGIS when encountered within quoted strings.

1. Backspace (BS) moves the drawing position one character escapement increment backwards, both horizontally and vertically.

2. Line feed (LF) moves the drawing position one character height down, perpendicular to the character baseline; LF also saves the resulting drawing position for restoration upon occurrence of a subsequent carriage return in the same quoted extent.

3. Carriage return (CR) moves the drawing position back to the position it had at the start of the current quoted extent, or to that saved at the most recent LF, whichever was more recent.

4. Horizontal tab (HT) moves the drawing position one character increment forward along the baseline.

ReGIS text does not automatically scroll or wrap at screen borders. The drawing point advances by character escapement increments and abides by the addressing constraints outlined in the S(A[][]) Screen Address Setup Command description.

An error character will be displayed for any of the following occurrences:

a. An attempt to print characters from a nonexistent or undefined character set (for example, T(A4)'x', where alphabet 4 has not yet been defined by an appropriate L(A4) command).

b. An attempt to print characters from outside the defined extent of a character set (for example, L(A1,E64) followed by T(A1)'a').

c. An attempt to print not-yet-defined characters from within the defined extent of a character set.

The error character may be device dependent, and need not be the same for each error above. Solid block characters or checkerboard characters are acceptable. The reverse question mark error character used in certain text protocols may not be a good choice, as the rotations and transformations allowed in graphics text may negate the advantages of its otherwise unique shape.

8.7.3.4 Options

8.7.3.4.1 T(A<integer>) - select Alphabet <integer>

The default value for <integer> is zero, and this value is restored on a valid S(A).

If a received value of <integer> exceeds the range implemented in a device, or <integer> is not specified, then the option is ignored and the previous state value is left unchanged. For suboptions R and L (described in the ReGIS Base), <n> is restricted to have a value of 0. [Note that ReGIS alphabets are not now and have never been coupled to ANSI G-set selection; ReGIS alphabet numbering is an independent character set enumeration scheme.] Should the R or L suboptions be specified without a numeric on their A-option, the last set selected by T(A) is still selected. If the <n> in the last T(A<n>) was not 0, then the suboption is ignored.

Refer to the description under the Load alphabet command for possible device dependent contents of the non-default alphabets.

8.7.3.4.2 T(B) - save Text attributes

Start a temporary text attributes range. The state of all Text option settings, including character set selection and the state of the in-use character table, but not including temporary writing options, are saved on a one-deep text attributes stack, to be restored by an E option in the current or a subsequent Text commands.

8.7.3.4.3 T(D<ang>) - set Text baseline Direction

Orient the baseline of each character at <ang> degrees to the horizontal.

An acceptable fallback is to limit angular resolution to a device dependent value, typically either 45 or 90 degree increments. When used in conjunction with the S option, string baseline is established. The default value is 0 degrees to the horizontal. This value is restored when a valid S(A) is executed.

8.7.3.4.4 T(E) - restore saved Text attributes

Restore text attributes that were in effect when the last previous B option was encountered.

If there was no previous B option specified, the default values for all states are restored. Repeated T(E) commands without intervening T(B) commands will restore whatever was saved by the last T(B), or the defaults if there has been no previous T(B) since power up.

8.7.3.4.5 T(H<number>) - set Text Height

Set the character height to be half of <number> times the standard-size-1 cell height, leaving the character width unchanged.

T(S) changes the entire character display cell size; T(H) changes only the height, the width remaining as set by T(S). The value may have a fractional part, but truncation to the next lower integer is allowable. Devices must accept values for <number> in the range of 1 to 25; conforming software will not use values outside of this range. Values larger than 16 should be implemented by use of the T(S[]) construct. This state has no default value, as the default for T(S) applies.

8.7.3.4.6 T(I<ang>) - set Text Italic slant

Slant subsequent characters by <ang> degrees, referenced from the perpendicular to the baseline and measured from the upper left corner of the character cell.

Positive angles cause the characters to appear to slant backwards. A zero <ang> returns the specifier to rectilinear. Resolution of italic angle may be device dependent, and angles greater than 45 degrees (positive or negative) may be taken to be exactly 45 degrees (positive or negative). The default value for this state is 0 degrees (upright); this default is restored when a valid S(A) is executed.

8.7.3.4.7 T(M[<hint,vint>]) - set Text pixel Multiplier

Multiply each pixel from the stored character image width by hint, and each pixel from the stored character height by vint.

This is a non-positional use of bracketed extents. Both the standard-size-1 unit cell (character image) and the standard-size-1 display cell (character image plus intercharacter background) are scaled according to these multipliers. If either <hint> or <vint> is missing, that value is interpreted as whatever value is appropriate for standard size 1. This function provides for compatibility with older implementations of ReGIS. The preferred mode of character scaling is by use of cell unit size and cell display size, T(U) and T(S), respectively.

8.7.3.4.8 T(S[<width,height>]) - set Text display cell Size

Set display cell size in the current screen addressing coordinates, but oriented with respect to the value of T(D) in effect.

This is a non-positional use of bracketed extents. The display cell is that area of the display affected by the writing of a character. It comprises the unit cell (see T(U[]), below) and whatever background area is required to achieve the extent defined as the display cell. Display cell size must be executed to within one physical pixel of the size specified; there is no fallback, as is allowed for the scaling of the unit cell. Refer to "further notes," below, for more on display and unit cells. If either <width> or <height> is missing, the missing values default to the values that correspond to the (device dependent) standard size 1, as would be set by T(S1). Maximum values of <width> and <height> are determined by the arithmetic capacity of the ReGIS device and the current imposed coordinate space, that is, 16 bit integer arithmetic. Default text display cell size is device dependent, and the default is restored on execution of a valid S(A).

8.7.3.4.9 T(S<number>) - set standard Text display cell Size

Use one of the "standard character sizes". Set the character width to be <number> times standard size 1 character width, and the height to be three-quarters of <number> times character height standard size 1. This element also sets a device dependent character escapement value; this escapement is relative to the character direction specified by the D option. "Standard size

zero", S0, is implemented as the smallest possible character rendering on the device. As for the H option, the argument may be fractional, but may be truncated to the next lower integer if the implementor chooses. Devices must accept values for <number> in the range of 0 to 16; conforming software will not use values outside of this range. A value of zero for <number> may have special, device dependent meaning. Values larger than 16 should be implemented by use of the T(S[]) construct.

8.7.3.4.10 T(U[<width,height>]) - set Text Unit cell size

Set unit cell size in the current screen addressing coordinates, but oriented with respect to any T(D) in effect.

This is a non-positional use of bracketed extents. Unit size is the size to which the character image is meant to be set. Allowable fallback is to scale the unit cell down to the next executable size smaller than that specified. If the unit cell size specified is bigger in either dimension than the display cell, the unit cell is truncated to the borders of the display cell. Refer to "further notes," below, for more on display and unit cells. If either <width> or <height> is missing, the missing values default to the values that correspond to the (device dependent) standard size 1, as would be set by T(S1). Maximum values of <width> and <height> are determined by the arithmetic capacity of the ReGIS device and the current imposed coordinate space, that is, 16 bit integer arithmetic. Default text unit cell size is device dependent, and the default is restored on execution of a valid S(A).

8.7.3.4.11 T(W(<suboption>)) - temporary Write options

Set temporary writing controls, as in the Vector command.

8.7.3.5 Further Notes

A text "display cell", specified in terms of height and width (using screen addressing coordinates) by T(S[]), is the screen area to be written for each character.

A text "unit cell", specified similarly by T(U[,]), is the screen area in which the image from character storage is drawn. See Figure 8-8. If these two cells are not of identical size, the unit cell is upper-left justified in the display cell, with any display cell area not filled by the character image explicitly filled with the background color. If the unit cell is larger than the display cell, only the upper left portion of the character image is drawn; that is, the unit cell is clipped to fit in the display cell.

In the computational limit (that is, where the granularity of scaling arithmetic is continuous and not bound by pixel increments), the display cell and the unit cell should map to identical sizes. An acceptable fallback for less capable systems restricts Unit cell sizes to integral multiples of the character storage cell defined for the character set by Load Pattern command, L[,]. A unit size that does not represent such an integral multiple may be rounded down to the nearest integral multiple value. Implementations may choose to round up if the target size is only slightly less than the closest realizable size, and if the resulting unit cell would not exceed the display cell. The display cell must be executed to within one physical pixel of its specified size.

The L command defines bit patterns; these are invoked by the T command and as area shading patterns in the W(S'x') command. Alphabet set 0 contains an 8-bit character in-use set. The content of this set is as determined by the ReGIS Base command T(A(L"*",R"*")). All sets except alphabet 0 may be loaded using the L command. See the description of the Load command for further information.

Different alphabets may be simultaneously available for loading and execution.

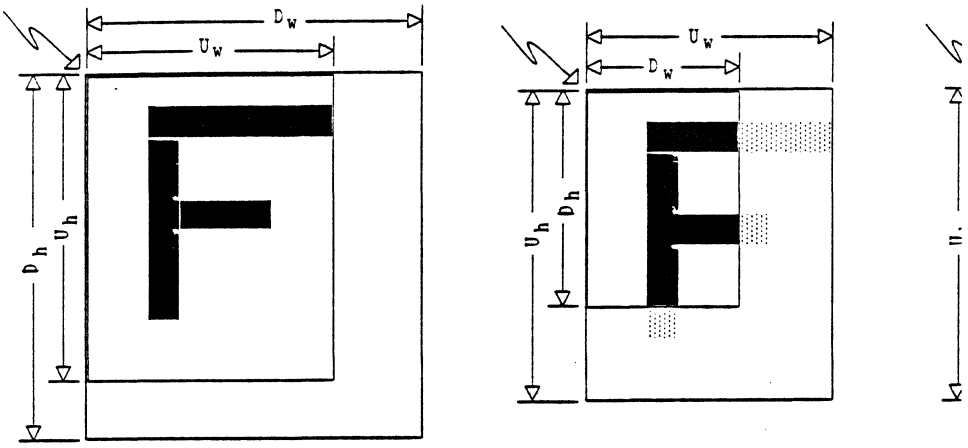


Figure 8-8: Character Display and Unit Cells

8.7.4 REPORT INSTRUCTION

There is one option to the report command in Raster ReGIS not in the ReGIS Base. It reports back the saved names of the loadable character sets.

8.7.4.1 R(L) - Report selected Loading alphabet

Report back the name of the pattern set currently selected for loading. The format of the report is (A'<name>') where <name> is the name provided in the L(A'<name>') command.

8.7.5 LOAD CHARACTER SET INSTRUCTION

A ReGIS device may implement several alphabets. These alternate alphabets are selected for drawing using the "A" option with a numeric subargument in the text instruction. The alphabet numbered 0 may not be modified, though the defined font is implementation dependent.

ReGIS allows the non-Base alphabets to be loadable. If the Load character set command is implemented, then provision for at least three loadable alphabets must be made. Loadable alphabets and name tables, as defined by L(A<n>"name"), may have device dependent initial contents, but they must be volatile and subject to overwriting.

The general syntax of a load instruction is the letter "L" followed by the ASCII character index, as a quoted character, which will be used to refer to the character. This is then followed by a sequence of numeric parameters, each individual numeric specifying a single row in the character pattern.

The most significant bit is the leftmost displayed pixel; the first parameter is the top row of the character. Parameters are hexadecimal integers separated by commas.

8.7.5.1 Position Arguments

A bracketed coordinate specifier defines the storage size, in bits per character, to be used with the selected alphabet. This is a non-positional use of bracketed extents. The selected alphabet is cleared when its storage size is specified. The storage size determines how many bits, and how many rows of bits, are encoded into the character mask arguments of the Load command. The storage size maps to the unit cell size when characters are displayed. Negative values are not allowed in storage cell sizes, and the default values for storage cell sizes are device dependent.

8.7.5.2 Numeric Arguments

Each character cell is redefined by a string of bit patterns expressed as groups of ASCII hex digits (0 ... 9, A ... F or a ... f). A cell is terminated by a semicolon or a successive parenthesized option specifier or quoted character specifier.

Note

The semicolon is required to terminate the Load command because of the nonstandard nature of the numeric arguments that form the bulk of the Load command format.

Groups of hex digits, each of which specify the contents of one scan line of the character cell, are separated by commas, or by exceeding the maximum number of hex digits needed to specify one scan line. If, in a given device, the default storage size were 12 horizontal by 10 vertical, 3 is the maximum number of hex digits needed to specify one scan line. A fourth consecutive hex digit would be applied to the next scan line whether or not an intervening comma were encountered. Where the default storage size is not known, storage size should be specified, or commas should be used to separate scan line specifiers.

Cells are always loaded one row at a time using hex-ASCII characters. A cell map is specified from the top down, more significant bits leftmost in the specifier string.

a. If more hex digits are specified than are needed for a single line of the mask, the extra digits are used on the next line.

b. However, if the width of the character cell is not a multiple of four, any unneeded, low order bits of the last hex digit applied to that row are discarded, and NOT carried over to the next row. Each raster row is begun with a new digit.

c. If too few digits are specified to fill a row, they fill the low order portion of the mask, setting the high order portion to zero.

d. If fewer mask lines are specified than exist in a cell, the unspecified portion of the cell is cleared to zero.

e. If more lines are specified than will fit in the storage cell, these extra lines are ignored - NOT overflowed into the next sequential character.

Figure 8-9 shows the format of the raster-oriented character storage cell.

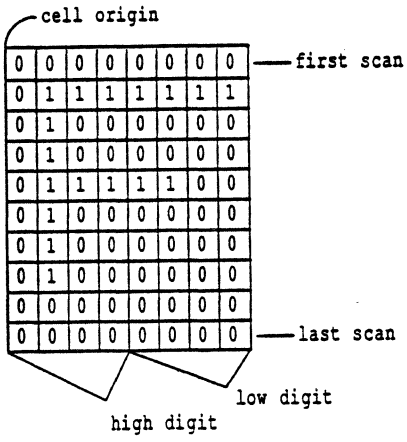


Figure 8-9: Character Storage Cell

8.7.5.3 Quoted String Arguments

A single quoted character serves as the index into the character set selected for storage. It will be the same character used for recovery of the loaded cell in the Text and Write attributes Shading commands. If more than one character is present in a string, the first is used, and the rest are discarded. Separate character arguments are required to specify multiple mappings.

8.7.5.4 Options

8.7.5.4.1 L(A<integer>) - select Loading Alphabet

Select alphabet set <integer> for loading. If loadable alphabets are provided, at least three must be provided, and they must be numbered 1, 2, and 3. Different alphabets may be simultaneously selected for access (by T(A)) and loading (by L(A)). L(A) and T(A) have no effect on each other. The default character set selected for loading is 1, and this default is set when a valid S(A) is executed. If a specified value for <integer> exceeds the implementation limit, the value is ignored and the alphabet selected for loading is the same as the one selected before the out of range value was received.

8.7.5.4.2 L(A"<name>") - associate <name> with Alphabet

Associate <name>, which may be up to ten characters long, with the character set currently selected for loading. Both <name> and <integer> may be specified in the same option, but if <name> is specified first, it is associated with the currently selected set, not necessarily the one specified by <integer>.

8.7.5.4.3 L(E<integer>) - specify character set Extent

The <integer> parameter determines the greatest number of characters which can be stored as part of the selected alphabet. The selected alphabet is cleared when the extent is redefined.

The range of the extent begins with the space character (code value 2/0), and comprises <integer> characters. The ninety-fifth character position is code 7/14, the ninety-sixth is 7/15, the ninety-seventh is 10/0, the 191st is 15/14, and the 192nd is 15/15. Codes progress in order between those values. Extents greater than 96 represent character values beyond the 7-bit set, and these characters must be accessed by 8-bit codes.

Codes 7/15 and 15/15 might not be encodeable in quoted string arguments depending upon the ReGIS user's environment, but devices shall make these code positions available.

The default extent for each loadable alphabet is 96 characters. This default is set only at power up.

8.8 OPEN EXTENSIONS TO ReGIS

The following specifies functions of ReGIS which have been used in product implementations or are standardized for future implementations, but are not required for conformance with any particular level of ReGIS. Extensions related to device technology not necessarily universal to devices which use that technology are described, as well as extensions which are potentially useful but not clearly called for on a cost/benefit analysis.

8.8.1 SCREEN INSTRUCTION

Position arguments to the Screen control command allow the devices the dynamic capability of performing screen motion, as in the case of text scrolling or "strip chart" generation. The range of implementation is very broad, ranging from no implementation at all to the full scale "panning" operation of allowing the operator to move the viewing window through a much larger image definition.

There are two models for the scrolling of data in a display:

a. Coupling the displayed data and display addressing, and decoupling both of these from the display.

   A displayed entity has a fixed relationship to the address at which it is written. A line drawn from location [100,120] to [200,250], for example, will always maintain those end points, regardless of how the display addresses are moved about the display. This model is similar to drawing a picture on graph paper and viewing it through a rectangular window that can expose all or part of the graph paper behind it. Any new information is written on the graph paper, and the lines on the graph paper are used to address the display space. The origin is always fixed in relation to the graph paper, but the paper can be moved about behind the viewing rectangle, modifying the viewer's perception. If the viewing rectangle exceeds the boundaries of the graph paper, either the opposite edge of the graph paper wraps in, or the view is truncated.

b. Coupling the display addressing and the display, and decoupling these from the displayed data.

   The display (the viewing window in the example above) and its addressing scheme are coupled. When data move, they do so in a manner which moves them from the address at which they were executed.

   This follows the addressing scheme of most text terminals. As data are entered at the bottom, for example, on line 24, old data move up through the address space to line 1 and off the top. Line 1 is always at the top, and line 24 is always at the bottom. Recovery of data moved off the display by scrolling in the opposite direction cannot be guaranteed.

In summary, the windowing model may extend into a general window to viewport model, the scrolling model mimics the use of text terminals for display. The appropriateness of either model depends upon the goals and capabilities of the underlying hardware. Neither model is preferable or prerequisite to the other. Neither model is required; devices may elect to not move data in their displays at all.

8.8.1.1 Position Arguments

In the first data movement model above (the windowing model), position arguments specify an address in the display address space where the origin of the viewing rectangle is to be positioned. Data appear to move left as the origin moves to the right, but since the data and addresses move together, data are generally recoverable by reversing the direction of the movement.

In the second data movement model above (the scrolling model), position arguments specify an address in the display address space which will become the new origin. As in the windowing model, data appear to move to the left as the origin is moved to the right, but in this model the data cannot necessarily be recovered by reversing the scrolling, since the data may have been moved off and erased from the display medium.

If data movement is supported in a ReGIS implementation, the choice of models used is device dependent. If the hardware can support both models, the model in effect is chosen by the S(D) option, described below.

8.8.1.2 Numeric Arguments

Numeric arguments to the Screen control command are treated as pixel vector specifiers. They are considered to be equivalent to relative address specifiers and are executed as a relative bracketed position argument. The data movement model chosen is the same for pixel vectors as it is for position specifiers.

8.8.1.3 Quoted String Arguments

There is no significance to quoted string arguments to the S command in Open Extensions to ReGIS.

8.8.1.4 Options

8.8.1.4.1 S(A) - screen Address definition

The prime function of the screen addressing command in Open Extentions to ReGIS is as it is in the Base. The following states representing Open Extensions are reset to defaults as a result of the execution of the execution of a valid S(A).

The equivalent ReGIS command that would institute this setting is shown after the defined action.

a. Blink attribute is set to off.

   Equivalent explicit command: W(A0)

b. Line width is set to nominal.

   Equivalent explicit command: W(L1)

c. Shading state is set to disabled.

   Equivalent explicit command: W(S0)

d. The clip rectangle is set to the range indicated by S(A), and clipping is disabled.

   Equivalent explicit command: S(R[][]0) <[][] as in enabling S(A[][])>

e. The display offset or scroll state is set to match the first coordinate of the S(A[][]) to the upper left hand corner of the display.

f. Data movement control is set to its device dependent value.

   Equivalent explicit command: S(D1), scrolling (if supported; else S(D0), windowing)

g. Cursor visibility is set to ON, and the default cursors are selected.

   Equivalent explicit command: S(C(H1)(I1)1)

h. Screen scaling is set to 1.

   Equivalent explicit command: S(S1)

The color output map is NOT AFFECTED by executing an S(A).

8.8.1.4.2 S(C) - visible Cursor control

Most interactive devices support a visible cursor that identifies either or both of the current drawing position or an input track position. The C option to the Screen control command implements access to this facility. A numeric subargument to the C option enables and disables the visible cursor. S(C1) enables the cursor display, S(C0) disables it. The default value for cursor visibility is enabled.

Suboptions and subarguments to the S(C) command allow for the selection among different cursor representations, and simultaneously different cursor representations for output and input tracking indicators.

Cursor Capabilities are Guidelines

The display actions of cursors should be tailored to the device being designed and to its intended usage. The mechanisms and actions for cursor display described in this section are quite complex and potentially expensive to implement. While the ReGIS syntax describing the selection mechanisms has the force of a standard, the display actions of cursors as regards blinking, multiple colors, and such may be taken to be guidelines, subject to modification as needed for product implementations.

a. Suboption S(C(H<n>)) allows for selection among various styles of output cursor. The value of <n> selects among predefined cursor styles. A value of zero or one for <n> selects a diamond shaped cursor with a small crosshair inscribed. This is the default cursor shape, and is selected after a valid S(A) has been executed. The diamond cursor is normally no more than a half inch or so in size. A value of 2 for <n> selects a full screen cross hair cursor. This cursor is composed of two lines, one horizontal, one vertical, that intersect at the drawing position.

b. Suboption S(C(I<n>)) allows for selection among various styles of input cursor. The value of <n> selects among predefined cursor styles. A value of zero or 1 for <n> selects a the diamond cursor identical to the diamond output cursor. This too is the default cursor, and is selected after a valid S(A) has been executed. A value of 2 for <n> selects a full screen cross hair cursor, again identical to the related output cursor. A value of 3 for <n> selects a rubber band line, and a value of 4 selects a rubber band rectangle as the input echo cursor. The two rubber band styles are anchored (the first end of the line and one corner of the rectangle) at the output drawing position in effect at the start of the input operation. The other end of the line and the diagonally opposite corner of the rectangle track the input position.

c. Additional subarguments to the S(C(I)) and S(C(H)) commands allow for program defined cursors. The cursor is defined by a pair of characters (a single quoted string) and an offset position. The two characters are

overlaid, with each one being rendered in a different color or intensity if color or gray scale is supported. If the implementation chooses to blink its cursors, the two characters should alternate between the two colors in opposite phase. The characters come from the alphabet currently selected for drawing by T(A). The offset is the distance, in character cell pixels, from the origin of the character cell to the point that will be tracked for output or input. This distance is always relative to the displayed location of the overlaid character pair. For example,

```
S(C(I[+4,+8]"?="))
```

defines and selects an input cursor composed of an overlaid ? and =, with an offset of four horizontal pixels and eight vertical pixels from the corner of the overlaid cells to the tracking position. The command

```
S(C(H[+0,+0]"OX"))
```

defines and selects an output cursor composed of an overlaid O and X, with the tracking location at the upper left hand corner of the character cells.

d. Applying the offset and quoted string subarguments of the S(C(H)) and S(C(I)) forms to the S(C) command directly, as by S(C"XO"[+4,+4]), enables the capabilities as described immediately above, but for all possible cursors at once. This is an acceptable fallback for devices that do not have separate output and input tracking cursors, or may be used as a shorthand way of controlling both cursors at once. Note that the anchor point for rubber band output cursors is not specified by this standard. Likely choices might be the screen origin, the screen center, or the top point of the current position stack.

8.8.1.4.3 S(D<integer>) - data movement control

For devices supporting both data movement models, the D option switches between them. The <integer> argument to D is 0 for systems which couple data and addresses, and allows data to be recovered by reverse scrolling. This is the windowing model, and is most useful in conjunction with the Screen Scaling option, S(S), below. An <integer> argument of 1 indicates that the scrolling model should be used, where the address space and the visible display are coupled, and the data change addresses as they move about the screen. If neither model is implemented, the option is ignored. If a device supports both modes, the default value is 1 (scrolling model); this default is set when a valid S(A) is executed.

8.8.1.4.4 S(H(<suboptions>)[][]) - retransmit Hard copy

The hard-copy option retransmits the visible image on the display surface to a second display device. This command is primarily for dot matrix transfer of images to hard-copy devices, but may also be used to store the image on another medium. The format of the retransmission, its encoding, or its media are all excluded from specification by ReGIS, though Sixels are the implementation choice in video terminals and host-based ReGIS systems. The hard-copy option may have up to two bracketed position specifier arguments which specify two opposite corners of a rectangular subset of the screen area. These position specifiers delimit the area of the display surface to be retransmitted. If only one position specifier is given, the rectangle is defined by that position and the current drawing position. If no position specifiers are given, the whole visible screen is output. Numeric arguments to the H option are NOT interpreted as pixel vector specifiers for purposes of specifying an area of the display for output.

A suboption P[,] specifies a starting offset for the graphics image when output is re-directed. The value specified refers to the offset to be executed on the device to which output is redirected. The values specified in the brackets are not necessarily those of the imposed coordinate space, but are implementation- and retransmission protocol- dependent. The upper left corner of the area retransmitted will be offset by the amount indicated in the P[,]. The value set by the suboption remains set until changed by the next occurrence of the P suboption, or until it is returned to its default by power up or execution of a valid S(A).

S(H[][]) is a non-positional use of bracketed extents. The value specified is always interpreted absolutely, whether signed or not. Any missing part of the bracketed extent is interpreted as 0. The default value for the state set by (P[,]) suboption is [0,0].

There is no standardized way to change the hard-copy offset without generating a dump, as the presence of the H option, necessary for the interpretation of the (P) suboption, has the primary meaning of "execute dump."

The amount of graphic information retransmitted may, at the implementor's option, be limited by the visibility of the image on the primary presentation medium. For example, if, in a two plane video device, the output map (controlled by S(M)) is set such that only the data in one of the two planes is visible on the primary output medium, the implementation may choose to only retransmit the visible information. In such cases, the method and limitations chosen must be clearly outlined in the product documentation.

8.8.1.4.5 S(M<index>(<color specifier>)) - modify the color Map

This command modifies the output map for devices with modifiable output maps. Value <index> specifies a device index, and <color specifier> is the HLS or RGB definition of that entry. The suboption alphabet for color specifiers is R, G, B, C, M, Y, W, and D for red, green, blue, cyan, magenta, yellow, white, and dark, respectively, and HLS for hue, saturation, and lightness. The interpretation for <index> is the same as for the indexed values to the foreground and background color index specifiers, W(I) and S(I). If <index> exceeds the number of entries in the selected map, it is mapped into the realized range as are the foreground and background indexes.

Suboption A<n> indicates which map is to be loaded for devices which have multiple maps. If A is not present, the primary map is used. If A is present without subargument, the primary alternate map is used. A color specifier must be complete in one suboption. The selected entry will generally be marked for loading when the terminating ")" is encountered, but the implementation may batch a series of <index>(<color specifier>) sets and load all specified entries at once to reduce undesired image "staging" due to piecemeal map changes.

The initial contents of all maps in a device is device dependent, but each entry in the map should be visually distinct from every other entry (and from the background, if it is mapped separately). This applies unless there are more entries than supported colors, or unless the number of colors supported approaches or exceeds the number visually distinguishable.

Color maps are set to their defaults only at device power up.

8.8.1.4.6 S(R<n>[][]) - define and enable/disable a clip Rectangle

A clip rectangle is a region of the display that can be used to limit the area that can be written. The S(R) command in ReGIS allows such a rectangle to be defined, and to enable and disable the limiting action.

a. The bracketed extents in the S(R) command define the two opposite corners of a rectangle, the sides of which are parallel to the ReGIS coordinate axes. These are positional specifiers, and may use relative addressing. Both corners must be specified. If only one specifier is present, the extent of the clip rectangle is not changed. If more than two are present, only the last two are used.

b. The numeric argument to the option is a switch to enable and disable clipping action. If <n> is equal to zero, then clipping is disabled, and drawing actions can take place everywhere in the display. If <n> is greater than

zero, drawing actions are limited to the interior of the specified clip rectangle. The "interior" of the clip rectangle is defined to include what would be drawn as the boundary of the clip rectangle.

c. As an extension, values of <n> less than zero indicate that drawing activity is limited to the exterior of the clip rectangle. This action is called "shielding." The exterior of the clip rectangle includes all the drawing area not in the interior (in particular, the boundary is NOT part of the exterior).

d. The extent of the clip rectangle can be set by an instance of the command without changing the enabled state. For example, the command S(R[][+200,+200]) sets the clip rectangle to an extent 200 square, starting at the current drawing position, but it does not affect the state of the clipping activity. A subsequent S(R1) will enable clipping to this area, or S(R0) will disable clipping, without changing the extent.

e. The clip rectangle maintains its position in the coordinate space, even if the display supports scrolling. The effect is that the clip rectangle moves with the data if the windowing model, S(D0), is supported, and the data move through the clip rectangle if the scrolling model, S(D1), is supported. In the latter case, data that move out of the clip rectangle never disappear just because of movement out of the clipping rectangle.

f. Screen erase, S(E), always erases the entire display, regardless of the clipping state. Erase mode writing, on the other hand, is bounded by the clip rectangle if clipping (or shielding) is enabled.

g. The default clip area is the extent defined by the last valid S(A) or the entire display surface, if no S(A) has yet been encountered. The default clip state is disabled, allowing writing to the entire display surface. Both defaults are restored by valid S(A) commands.

8.8.1.4.7 S(S<n>(<suboptions>)) - Scale the visible image

This option is used for devices which can scale images after they are displayed. Taking the model of pixel replication, the subargument <n> is treated as a multiplicative scale factor for each pixel in the display. When the S(S<n>) form is used, the display is scaled isotropically. There are two suboptions (X<n>) and (Y<n>) for scaling independently in X and Y. The display can be rescaled only by respecifying a scale factor of 1. This option is most useful with the windowing data movement model described above. The S(S) option causes the display medium (the bit map,

for example) to become bigger than the display surface. S(S) and S[] then combine to move a small window through a large display space.

Negative values for scaling factors are not allowed. A value which exceeds the implementation limit in a device is mapped to the implementation limit. The default value for scaling factors is 1, and is restored by the execution of a valid S(A).

8.8.1.4.8 S(W(<suboptions>)) - temporary Write options

This command sets temporary writing controls just as in the Vector command. The only useful temporary writing control within the Screen command is S(W(M)), which sets the pixel vector multiplier. The pixel vector multiplier affects the granularity of the display motion generated by S<pixel vector> commands.

8.8.2 POSITION INSTRUCTION

8.8.2.1 P(P<n>) - select drawing surface

Some devices have multiple separate drawing surfaces. Programs may select among these separate surfaces with the P option to the P command, P(P<n>). The value of <n> identifies the drawing surface selected. All drawing surfaces share the same state store, that is, current drawing color, selected text alphabet, macrograph store, position stack, and so on.

8.8.3 WRITING ATTRIBUTES INSTRUCTION

There are four options extensions to the Write attributes instruction.

8.8.3.1 W(A<n>) - Alternate (blink)

This option enables the blink (or "alternate") attribute. For values of <n> greater than zero, set the blink attribute on all foreground data drawn until blink is disabled. Application of the blink attribute is disabled by W(A0), S(E), or valid S(A). Clearing the subsequent application of the blink attribute by W(A0) does not change the blinking status of any displayed elements, any more than changing the foreground specifier with W(I) changes the color or intensity of any pre-existing data on the display.

The visible action of blink is implementation dependent. The alternation may be between foreground and background intensities, or between foreground and some other device-dependent setting. Interaction of the blink attribute with Complement and other image-content dependent writing modes is device dependent. Blink frequency and duty cycle are device dependent.

8.8.3.2 W(L<n>) - Line width control

This option controls the width or weight of lines and curves. The value <n> is a multiplier of the device's nominal line width. Nominal line width is device dependent, and is normally the device's default line width. W(L2) sets a width twice the nominal width, W(L3) three times, and so on.

Fractional line widths are allowed. For example, W(L0.5) specifies a width 50% less than nominal, and W(L1.5) specifies a width 50% greater than nominal. Devices that do not support fractions may truncate to the next lower integer or round to the nearest integer, just as they do for other numerics. W(L0) selects the minimum imageable line width.

8.8.3.3 W(S(<suboptions>)) - Shading

The Shading option to the W command accepts subarguments indicating a shading reference line and shading texture. There are also suboptions to the W(S) form which indicate whether shading takes place to a horizontal line (the default), a vertical line, or to a point. The form W(S(Y)...) indicates that the shading takes place to a horizontal reference line; this is the default. See Figure 7. The form W(S(X)...) indicates that the shading takes place to a vertical reference line. The form W(S(P)...) indicates that the shading is to a point. These suboptions are volatile, that is, every invocation of shading assumes that shading is to a horizontal baseline unless a countering (X) or (P) suboption is provided.

8.8.3.4 W(W) - microcoded Writing modes

Detailed control of multiplane writing is enabled with the Write control option to the W command, W(W). The option W accepts a numeric index and suboptions P<j> and N<k>. The form of the instruction extension is W(W<i>(P<j> N<k>)). The values <i>, <j>, and <k> are indices for memory contents, typically bit plane values. If a pixel to be written has value <i> before the write operation, it is written to value <j> if the next value from the pattern register, area pattern, or character cell is a 1, or to value <k> if the next masking value is a 0.

These settings override the last specified write mode control option (R, C, V, N, or E), but only those entries explicitly loaded take effect. Any entries not loaded retain their settings as determined by the last effected Writing mode command.

Any Writing mode command overrides all detailed settings.

The default writing modes are reset when a valid S(A) is executed.

8.8.4 TEXT INSTRUCTION

[This section became blank as a result of the movement of 8-bit character support to Base ReGIS.]

8.8.5 REPORT INSTRUCTION

8.8.5.1 R(E) - Report Error

This command reports back the last error encountered by the ReGIS parser. The format of the error report is "n,m", where n is a decimal digit string representing the error code, and m is a decimal digit string returning supplementary information. Except where noted below, this supplementary information is the decimal equivalent of the character code of the character flagged as causing the error. When it cannot be determined exactly what caused the error, 0 is returned in the error character field. Only the most recent error code is kept in the error report register. The error report register is cleared when ReGIS is synchronized by a ";" or by activation synchronization. The error report register is not cleared when the report is sent. The following are the reportable error codes:

> 0 = no_error: No error has occurred since the last synchronizing operation. The supplementary error field returns 0.

> 1 = ignore_character: An unexpected character was encountered and ignored.

> 2 = extra_option_coordinates: A command option that supports paired bracketed extents (such as S(H[][]) or S(A[][])) contained more than two coordinate pairs. The supplementary error field returns 0.

> 3 = extra_coordinate_elements: The syntax [,] contained more than one comma. Characters after the second comma up to the right square bracket (]) were ignored. The supplementary error field returns 0.

> 4 = alphabet_out_of_range: The syntax L(An) or T(An) contained an alphabet number "n" that was less than 0 or more than the implementation limit.

> 5 = alphabet_attribute_error: At least one of the following is true:

a. In the syntax L(An)[h,w] or L(An,Ex), the alphabet number "n" was zero. The attributes of the default alphabet (alphabet 0) may not be modified.

b. In the syntax L[h,w], the height "h" exceeded the implementation limit.

c. In the syntax L[h,w], the width "w" exceeded the implementation limit.

d. In the syntax L(En), the alphabet extent "n" exceeded the implementation limit.

> 6 = reserved code

> 7 = begin/start_overflow: The implementation limit for position stack depth was exceeded, that is, a V(B), V(S), P(B), or P(S) could not be processed because the stack was full.

> 8 = begin/start_underflow: A V(E) or P(E) was encountered without a matching V(B), V(S), P(B), or P(S) preceding it.

8.8.5.2 R(I<n>) - Input processing mode

This extension selects between separate or overlapped input and output processing. R(I0) selects "one-shot" mode. In this mode, which is the default mode and is set by executing a valid S(A), processing of ReGIS output is suspended by the device while interactive input (R(P(I)) is pending. Furthermore, input data can be transmitted by the device only once per R(P) or R(P(I)) command.

R(I1) selects "multiple input" mode. In this mode, the device can report positions with or without receipt of R(P) or R(P(I)) commands. The device reports on operator actions on the input devices, typically a mouse or graphics tablet. While in multiple input mode, the device can continue to execute output commands as received.

The commands R(I0) or S(A) exit multiple input mode.

8.8.5.3 R(P(I)) - Report Position Interactive

This extension differs from the Report Position command in the Base ReGIS in that it enables local interaction with an input device. This command puts the ReGIS device into input mode. In this mode an input device, such as a data tablet, mouse, or cursor keys cause an echoing cursor to be shown on the display surface. This echo tracks the input device, and generates a position report when the user generates a break condition. This break is usually effected by enabling an auxiliary control, such as a control button, associated with the device.

There are three responses allowed to the R(P(I)) command:

1. The normal bracketed coordinate specifier, just as from the normal R(P) command - This is the result from a non-interactive device.

2. An empty report - Only the terminating CR is sent. This is in response to a special break condition set by the user, indicating that NO INPUT was to be returned.

3. A normal bracketed response preceded by a break condition indicator - This indicator may be a single character code or a numeric string. This passes more details of the break condition to the application.

This command is purposely underspecified because of the wide variety of input device and input/output device couplings.

8.8.6 FLOOD/FILL INSTRUCTION

The Flood/ Fill command, keyletter "F", specifies general area operations in ReGIS. As was the case with Circles and Curves both being expressed by a single keyletter, Fill and Flood are really different commands in similar guises. Mixing of options intended to affect Flood (B) with those intended to affect Fill (P, V, and C) is non-conforming usage, and the effects of such mixing are not standardized.

Flood "colors in" a bounded area already present in the bit map with variable pattern or texture. Refer to 8.6.4 Area Attributes for a discussion of the various area coloring modes. Flooding is implemented in devices with read-write display media, such as bit map raster displays, since the display medium must be interrogated to discern seed values and boundary values.

Fill defines bounded areas by their edges and colors in only and entirely, subject to area texture or pattern in effect, the area (or areas) bounded by the edges presented. Fill is defined in Base ReGIS, and is extended here. Implementations which support the F command may elect to support either Flood or Fill or both at their option.

8.8.6.1 Position Arguments, Numeric Arguments

A position specifier to the F command, either a bracketed coordinate specifier or a pixel vector digit, specifies the location of the "seed" pixel from which a Flood is executed. The border of the flooding operation is as determined by the last previous F(B()) encountered. If more than one specifier is given, each is executed in turn. The current position does not change as the result of any flood operations.

8.8.6.2 Quoted String Arguments

There is no meaning assigned to the presence of quoted character arguments in Fill or Flood commands.

8.8.6.3 Options

8.8.6.3.1 Flood

8.8.6.3.1.1 F(B(<suboption>)) - flood Boundary condition

This option sets Boundary conditions for the definition of the closed area to be flooded. If the <suboption> is F0, a flood will bounded by image areas represented by the background. If it is F1, the flood is bounded by any foreground value different from the seed value, or any foreground if the seed location has the background attribute. If the <suboption> is I<integer> or I(<color/intensity spec>), the flood is bounded by parts of the image with the attribute so specified.

The default boundary for a flood is as set by F(B(F1)), that is, any foreground setting different from the seed value. This default is restored upon execution of a valid S(A).

8.8.6.3.2 Fill

Extended Fill acts as Fill defined in Base ReGIS, with an extension to support complex polygons.

8.8.6.3.2.1 F(...F...) - fill complex polygon

A "complex polygon" is defined as a closed figure described by an explicitly disconnected set of edges. Many letterform shapes fall into this category. The letter "O," for example, may be described as two roughly concentric closed curves defining the outer and inner perimeters of the letter shape. The letter "B" may be described as two polygons within a third. Because of the interior rules for polygons, the letter "i" may be defined as a complex polygon with disconnected but nested definitions for the body of the i and its dot. See Figure 8-11, below.

The F option to the F command provides for a one-level deep recursion of polygon definition to allow for the description of such complex polygons. Further subarguments to the F option are the same V, P, and C options described above. The presence of the F option decouples the edge streams from one another. This allows nested polygons to be defined without the need for an invisible edge to connect the edge list definitions. The recursion of the F option need only be one deep because of the interior rules for polygons. The position of the F option within the F command is not critical as regards the ordering of the subpolygons. This is because of the interior rules and the deferred execution of polygons until the polygon fill terminator is encountered.

Complex polygons must abide by the same constraints as simple polygons as regards the number of entries in the vertex list. The total number of vertices is the critical parameter. There need not be a separate vertex list for the nested polygons. Multiple nested polygons using the F(...F...) option are shown in Figure 8-11. Note that as the inner loop of the complex polygon is moved out of the outer, its interior takes on the interior attribute, until, when entirely outside, it becomes a disjoint polygon.

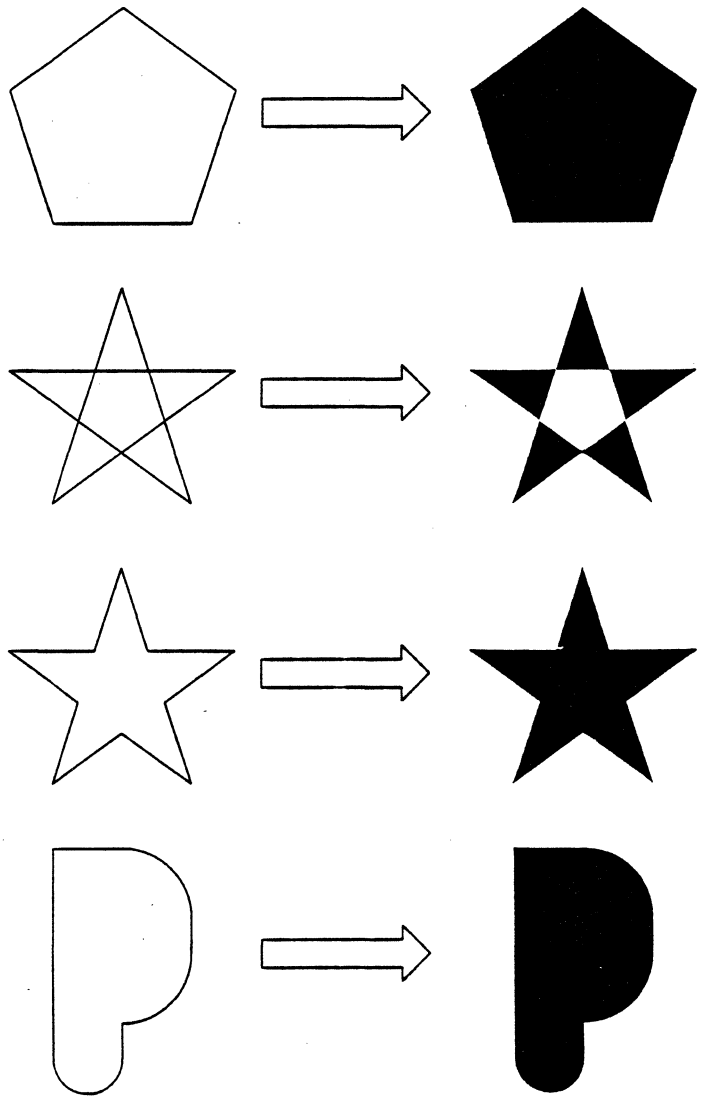


Figure 8-10: Simple Polygons

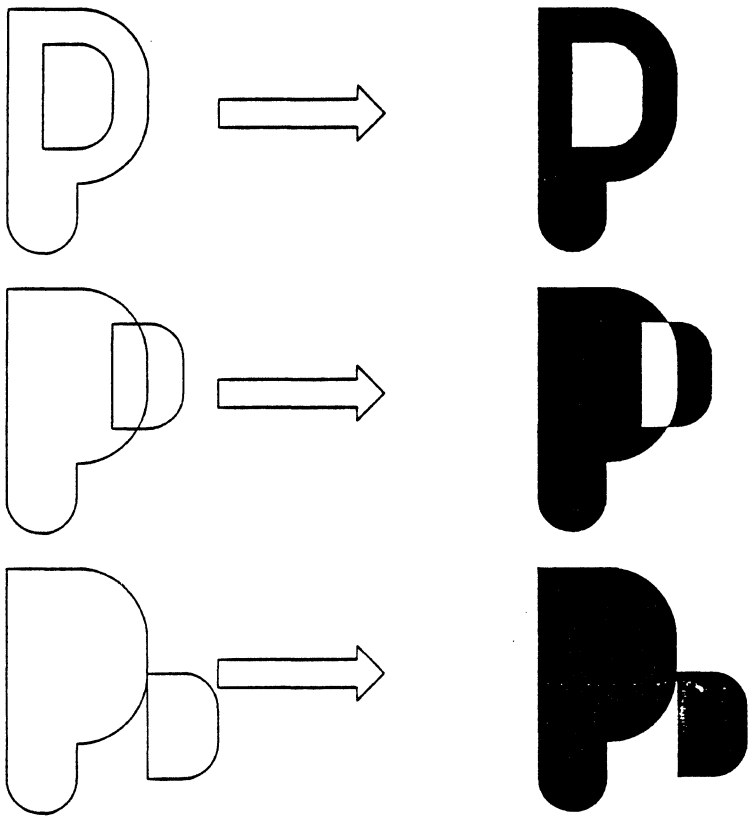


Figure 8-11: Complex Polygons

8.8.6.3.3 Common

8.8.6.3.3.1 F(W(<suboption>)) - temporary Write options

Set temporary write options, as in the other drawing commands.

8.8.6.4 Further Notes

The Flood command uses as a flood pattern the last selected area shading pattern as defined by W(S<n>), which selects a pattern defined by the linear pattern specifier W(P), or W(S"x"). This is true even if shading has been subsequently disabled by W(S0). Shading should be disabled before a flood or fill operation is initiated. The currently selected write mode and intensity specifier are used. These may be changed by the temporary write options to the Flood command.

The boundary conditions for the flood operation may be expressed in one of three ways:

1. As a specific attribute condition.

2. As any change in attribute with respect to the seed location attribute value.

3. In the single plane system, any change between the foreground and the background state.

The flooded area does not include the boundary. For example, if a red (unfilled) circle is on the display, and the current write mode is solid green, a flood of the interior of the circle will leave a green circle with a red circumference. This allows the edge of other objects to be used as a boundary without modifying that other object.

If the area in which the seed location is found is not closed with respect to the boundary conditions set, the flood will "leak out", causing a writing action that could cover the entire display. Note that non-solid vectors written in overlay mode do not necessarily form closed boundaries.

Areas may have complex boundaries, such as the area between two concentric circles, or the area between bounded figures on a background.

A boundary need only be connected by diagonally adjacent pixels to define closure for Floods.

8.9 INSTALLATION ENVIRONMENTS

This section contains an outline of the requirements for environmental support for ReGIS.

For the purposes of this section, assume that ReGIS addresses two environments for the application of the graphics instructions:

1. Descriptor code enveloped in ANSI escape sequences and transmitted on character oriented data paths, or

2. Use in a bounded graphics system.

8.9.1 ANSI ENCODING

The ReGIS command syntax is specified as if it were a seven bit encoding. Allowance for an eight bit encoding is an environmental issue. For example, ReGIS may be encoded into either the left or right half of an eight bit code table, and only character codes of the proper sense could be passed to the parser. However, ReGIS shall accept eight bit codes within quoted strings for use as arguments to text-related or text-indexed commands.

For communication to devices or interpreters that accept ReGIS as, perhaps, one of many device syntaxes, ReGIS commands shall be introduced by a DEC-private parameterized form of the ANSI escape sequences Device Control String (DCS) and terminated the ANSI escape sequence String Terminator (ST).

For communication to interpreters that accept only ReGIS, such as dedicated software for ReGIS translation, or that are capable of identifying the data syntax in a file by other means (such as file type flagging), ReGIS commands need not be delimited in any way.

The DEC-private parameterized form of DCS is

ESC P<n><protocol selector>

where <n> represents an ANSI X3.64-like parameter string encoding device controls, and the <protocol selector> for ReGIS is lower case "p".

A ReGIS interpreter must recognize two flags encoded in the first numeric parameter <n> if one is present in the invocation stream.

If the least significant bit of the numeric value is a '1', then the interpreter shall force an environmental resynchronization before interpreting any following ReGIS instructions. This level of resynchronization takes place regardless of the previous state of the interpreter, and returns the interpreter to the highest command level with no quoted strings or macrograph definitions in process. This resynchronizing mechanism shall be implemented in all ReGIS interpreters that use DCS/ST delimiting.

If the second least significant bit of the numeric is '1', the interpreter shall process the ReGIS instructions as it otherwise would, but should also pass along the source ReGIS stream to a text display process for display as text. This simultaneous display action is useful for debugging software using ReGIS on terminals and other interactive devices. This concurrent display action is not mandatory for ReGIS interpreters, and is obviously not applicable in all instances.

ST is encoded as

ESC \

and is the standard terminator for the collection of ANSI escape sequence strings.

All characters within this sequence following the protocol selector, up to but not including the string terminator are interpreted as ReGIS instructions. Although an entire screen image does not have to be included in one invocation of these sequences, individual instructions should not be broken up across escape sequence boundaries.

Some implementations may require that leaving ReGIS and re-entering via ST and DCS cause the parser to resynchronize, as if a semicolon were encountered within the instruction stream or as if a resynchronizing parameter were included in the invoking DCS.

Some devices scan for and parse ANSI-mode escape and control sequences while interpreting ReGIS. Some of these devices execute escape sequences as if the devices were not in ReGIS mode, and return to ReGIS mode after such execution. Other devices use the occurrence of any escape code, code position 1/11 or 8/* or 9/*, as reason to exit the ReGIS mode. Such use may be necessary to support non-conforming software, but such use is deprecated by this standard.

8.9.2 BOUNDED SYSTEMS

ReGIS is also intended for use in bounded systems even if no remote communications link is involved. Therefore, images defined in a bounded system may be used in distributed systems and vice versa.

In the realm of such bounded systems, ReGIS should be considered for the following purposes:

a. Graphics text file definition of a graphic image.

b. Direct use of ReGIS instructions in high level languages when no embedded graphics statements are provided.

c. Communication of a graphics image to a separate graphics device.

It is assumed that such a device should have a ReGIS interface "driver" to convert the character strings originating at any of these sources to the native graphics hardware instructions of the system.

8.10 ReGIS COMMAND COMPLEMENT

The following sections summarize the assigned ReGIS command elements. Each section outlines a separate command.

In ReGIS, a command keyletter can take four forms of argument: bracketed extent, numeric, quoted string, and parenthesized options.

Numeric arguments are classified by semantic use, as pixels vector, scalar value, boolean, switch, or other use.

Parenthesized options contain keyletters, each of which can be followed by any of the four argument forms.

Indenting of the construct identifiers indicates its applicability and scope, as if in an outline.

Thus indented under each keyletter will be one or more of [] (for bracketed extent), n (for numeric), "" (for quoted string), and () (for options and further nesting). Under a () line or subline will be the subarguments and suboptions that apply in that option.

The text on the right describes the capability encoded by the construct(s) on the left. Part of this description will be an indication of where that construct is defined and what conformance level of ReGIS it applies to. The notation {B} indicates the construct is part of the ReGIS Base; {R} indicates the construct is part of Raster ReGIS, and {O} indicates the construct is part of Open extensions to ReGIS. Unless otherwise indicated, all constructs have the conformance level of their parent construct.

8.10.1 SCREEN INSTRUCTION

```
S                       screen control commands {B}
  []                      screen data movement {O}
  n                       pixel vector, screen data movement {O}
  ()
     A                    screen coordinate definition {B}
       []                   coordinate extent (two [] required) {B}
     C                    cursor control {O}
       n                    boolean - cursor enable/disable
       ""                   text character used for cursor
       ()
          H               drawing position cursor control
            ""              (as for S(C""))
            []              (as for S(C[]))
          I               input tracking cursor control
            ""              (as for S(C""))
            []              (as for S(C[]))
     D                    data movement mode {O}
       n                    data movement mode selector
     E                    erase display {B}
     F                    feed medium/image complete {B}
       n                    value, frames to advance medium
     H                    image retransmit {O}
       []                   area of display to retransmit (1 or 2)
       ()
          P                   destination image offset
            []                  destination image offset vector
     I                    background intensity selector {R}
       n                    value, background index
       ()                   color/intensity specifier
          RGBCYMHLSDW         (the ReGIS color/intensity alphabet)
     M                    load output color map {O}
       n                    value, output map entry index
       ()
          A               select alternate output map
            n               value,alternate map id
          RGBCYMHLSDW         (the ReGIS color/intensity alphabet)
     R                    control drawing clip area {O}
       n                    tri-state value: shield, clip, disable
       []                   define clip area (2 needed)
     S                    scale display {O}
       n                    value, scale factor
       ()
          X                 scale in X direction
            n                 X direction scale factor
          Y                 scale in Y direction
            n                 Y direction scale factor
     T                    suspend interpretation for a time {R}
       n                    value, suspension interval
     W*                   temporary write options
```

*Temporary write options apply as they would if set by a W at command level, but only for the duration of the current command.

8.10.2 POSITION INSTRUCTION

All Position components are defined in the Base, except as noted.

P	Position instruction
[]	drawing point destination
n	pixel vector, drawing point destination
()	
B	position stack push
E	position stack pop
P	change drawing/display surface {0}
S	position stack push
W*	temporary write options

*Temporary write options apply as they would if set by a W at command level, but only for the duration of the current command.

8.10.3 WRITING ATTRIBUTES INSTRUCTION

```
W                       Write attributes instruction {B}
  ()
    A                   set writing mode to blinking {O}
      n                   boolean, enable/disable blink
    C                   set writing mode to complement {R}
    E                   set writing mode to erase {R}
    F                   enable foreground planes {R}
      n                   value, foreground mask
    I                   select foreground color/intensity {R}
      n                   value, foreground index
      ()                  foreground color/intensity value
        RGBCYMHLSDW         (the ReGIS color/intensity alphabet)
    L                   define line width {O}
      n                   value, line width
    M                   set pixel vector multiplier {B}
      n                   value, pixel vector multiplier
    N                   set writing mode negate {R}
      n                   boolean, negate enable/disable
    P                   select line drawing pattern {B}
      n                   value or bit pattern, selects pattern
      ()
        M                   select pattern element multiplier
          n                   value, pattern element multiplier
    R                   set writing mode replace {R}
    S                   set area operation shading {R}
      n                   boolean, enable/disable shading
      ""                  select area pattern for shading
      []                  select shading baseline
      ()                  modify shading baseline selection {O}
        X                   shade to X
        Y                   shade to Y
        P                   shade to point
    V                   select writing mode overlay {R}
    W                   set microcoded writing modes {O}
      n                   before-writing pixel contents
      ()                  after-writing pixel contents
        P                   if pattern element is 1
          n                   value of new pixel contents
        N                   if pattern element is 0
          n                   value of new pixel contents
```

8.10.4 VECTOR INSTRUCTION

All Vector components are defined in the Base.

V	Vector instruction
[]	draw line to end point
n	pixel vector, draw line to end point
()	
B	position stack push
E	draw to position stack top entry, pop
S	push empty entry to position stack
W*	temporary write options

*Temporary write options apply as they would if set by a W at command level, but only for the duration of the current command.

8.10.5 CURVE INSTRUCTION

All Curve and Circle components are defined in the Base.

```
C                   Curve/Circle instruction
  []                operating point of curve or circle
  n                 pixel vector, operating point
  ()
     A                limit circles to circular arcs
       n                value, degrees of circular arc
     B                begin closed curve interpolation
     C                circumferential circles
     E                end curve interpolation
     S                start open curve interpolation
     W*               temporary write options
```

*Temporary write options apply as they would if set by a W at command level, but only for the duration of the current command.

8.10.6 TEXT INSTRUCTION

```
T                   Text Instruction {B}
  []                set character-to-character escapement {B}
  n                 pixel vector, character position offset {B}
  ""                text to draw {B}
  ()
     A                select character set {B}
       n                select alternate character set {R}
       ()
          L             select standard character set into GL {B}
            ""            character set identifier
          R             select standard character set into GR {B}
            ""            character set identifier
     B                save current text state {R}
     D                set character/ line baseline direction {R}
       n                value, angle of baseline direction
     E                restore saved text state {R}
     H                set character drawing height {R}
       n                value, character drawing height
     I                select character slant {R}
       n                value, character slant angle
     M                select character mask pixel multiplier {R}
       []               pixel mask multiplier
     S                set character drawing size {R}
       n                value, character drawing size multiplier
       []               character display cell size
     U                set character unit size {R}
       []               character unit size
     W*               temporary write options
```

*Temporary write options apply as they would if set by a W at command level, but only for the duration of the current command.

8.10.7 REPORT INSTRUCTION

```
R                   Report instruction {B}
  ()
     E                report error status {O}
     L                report alphabet selected for loading {R}
     I                set input mode {O}
       n                boolean, input mode
     M                report macrograph contents {B}
       ()
          x             macrograph name
     P                report drawing position {B}
       ()
          I             report input position {O}
```

8.10.8 FILL/FLOOD INSTRUCTION

```
F                Fill instruction {B} / Flood {O}
  []             flood source position {O}
  n              pixel vector, flood source position {O}
  ()
     B             flood boundary condition {O}
       ()
          F        flood to foreground or background
            n        boolean, foreground/background selector
          I        flood to index
            n        value, index of flood boundary condition
            ()       color/intensity of flood boundary
                  RGBCYMHLSWD  (ReGIS color/intensity alphabet)
     F             fill complex figure {O}
       ()            (as for normal fill option list)
     PVC**         specify filling boundary {B}
     W*            temporary writing options
```

*Temporary write options apply as they would if set by a W at command level, but only for the duration of the current command.

**Filling boundary is specified by the same Position, Vector, and Curve/Circle formats as would be used to draw the outline alone.

8.10.9 LOAD CHARACTER SET INSTRUCTION

All Load character set components are specified in Raster ReGIS.

```
 L               Load character cell
   []              specify character cell size
   n               numeric (special radix), character cell data
   ()
      A            select alphabet for loading
        ""           associate name reporting name with alphabet
        n            value, alphabet to be loaded
      E            set size of character set to be loaded
        n            value, number of characters
```

Section Index

DEC STD 070-9 VIDEO AND PRINTER SYSTEMS REFERENCE MANUAL - SIXEL GRAPHICS EXTENSION

DOCUMENT IDENTIFIER: A-DS-EL00070-09-0000 Rev A1, 03-Aug-1990

ABSTRACT: This section describes the Sixel Graphics Protocol, which forms an extension to the Level 1, Level 2, and Level 3 Character Cell Display service class. Sixel Graphics provide a means of encoding and displaying raster binary data on raster devices of various capabilities.

APPLICABILITY: SARA Formal Cross-Component Standard. Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces.

STATUS: APPROVED 03-Aug-1990; use VTX SMC for current status.

DEC STD 070-9 Video and Printer Systems Reference Manual - Sixel Graphics Extension

DOCUMENT IDENTIFIER: A-DS-EL00070-09-0000 Rev A1, 03-Aug-1990

REVISION HISTORY:	Rev A,	24-Nov-1987	
	Rev A1,	03-Aug-1990	ECO Number CTS01

Document Management Category:	Terminal Interface Architecture (STI)
Responsible Department:	VIPS Architecture Group
Responsible Person:	Peter Sichel

APPROVAL: This document, maintained by the VIPS Architecture Group, has been reviewed and recommended for approval by the General Review group for its category for use throughout Digital.

Peter A. Sichel

Peter Sichel - VIPS Architecture Group

Eric A Williams

Eric Williams - Standards Process Manager

Direct requests for further information to:

Peter Sichel
Use $ VTX ELF for the latest location information.

Use VTX SMC to order copies of this document from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

CONTENTS

1 INTRODUCTION

This document defines a method of defining and transmitting encoded bit map data over a communication line known as the Sixel protocol. All or portions of this protocol may be stored in files. This protocol is to be used in display devices (video and hardcopy) of various imaging (drawing) capabilities.

In addition, it defines the relationship of this protocol to the ANSI Text presentation protocol. While the Sixel protocol is not intended to be a part of the ANSI text protocol, there are certain interdependencies.

This document regroups several extensions made to the "Sixel Protocol", the original 7-bit black and white protocol (LA34/VK100), the 8-bit extension (LA100-V2), the variable aspect ratio extension (LA100-V2), the color extension (VT240), and finally a limited support for software-handled viewporting.

1.1 OVERVIEW

The transmission of bit map data is one way of transferring visual information. It is required for natural images (pictures usually generated by scanning an original), or as a low level protocol for interfacing devices supporting different protocols, or when composition time is long compared to transmission time.

The Sixel protocol allows devices to receive and print black and white or color bitmap data at various sizes over a stream-oriented communications line. Six bits of each 7-bit or 8-bit character code are used to represent bit map data, allowing the remaining values to be used to control the context of the communications line and to fit within the ANSI text syntax.

The areas of interest when considering this protocol are:

1. Defining the data, at a given quality level, with all attributes. Ensuring that an imaging device having the same capabilities as the generating device will produce an identical picture.
2. Positioning the data
3. Defaulting mechanisms when some data cannot be exactly reproduced, or some command cannot be executed.

The structure of the protocol is intended to support a layered system approach, where there are several independent processes that define or interpret portions of the total data. For example, this allows the size of an image to be defined independent of the actual image definition. This implied scaling of bit map data is difficult to implement exactly and realizations will vary from device to device.

The Sixel protocol allows considerable freedom in specifying the value and size of the bit data to be displayed. It is expected that most imaging devices implementing this protocol will not be able to image the bit map data exactly as specified for all possible parameter values. This limitation is usually due to device imaging resolutions, dot sizes and computational limitations. The common fallback will be for devices to image at integer multiples of the device resolution, but this standard does not prohibit more creative solutions. See the section on imaging fallbacks for the definition of proper fallbacks.

2 GOALS OF THE SIXEL PROTOCOL

1. Provide a general purpose image data input for display devices.
2. Be syntactically compatible with ISO 2022 code definition, and further to be compatible with ANSI X3.64 and ISO 6429. This supports the requirement that the protocol be transmittable on 7-bit stream-oriented serial lines, including control code (X-ON, X-OFF) flow control.
3. Support screen size images from VT125, VT24X terminals. This size is 768 pixels by 240 pixels for the VT125 and 800 pixels by 240 pixels for the VT24X series. Both printed at a 2:1 pixel aspect ratio.

   It is a goal that any pixel value combination for rasters of this size be printable by conforming devices. In other words, 'picture too complex' for the above raster sizes is not permitted.

   The protocol should support any size rasters, although not all devices will be capable of displaying them.
4. It is a goal that it be possible to relate the size and placement of the raster to the ANSI text coordinate space, but not that the individual pixels relate to the ANSI text coordinate system or characters. It is not a goal to allow modification to the rendition of characters imaged via ANSI text with pixels imaged via the Sixel protocol.
5. It is a goal to allow independent control over the pixel definitions (shape and value of pixels) and the imaged size of the pixels.
6. It is a goal to be compatible with previous devices supporting limited Sixel printing - LA34, LA12, LA100, VT125, VT240.

3 TERMINOLOGY

Dot - The smallest displayable unit; a light dot on a screen, an ink dot on the paper. Dots can be round, oval, square, rectangular, small or big.

Pixel - The logical rectangular image area defined by each bit of Sixel data as intended by the generation software. The shape of a pixel is defined by an aspect ratio, and has no size.

Pixel Aspect Ratio - Defines the shape of the pixel as a ratio of the vertical side of the rectangle and the horizontal side. For example, a square pixel has an aspect ratio of 1 to 1 (or 10 to 10) and a pixel twice as high as wide has an aspect ratio of 2 to 1 (or 20 to 10).

Pixel-Spot - The area that is actually imaged (printed) for each pixel. It has a shape (Round, oval, square..) and a size. The shape and size are totally device dependent, and may or may not be related to the grid size.

Grid - The positions on the display, where pixel-spots may be placed. These positions are dimensionless points, even though the pixel spots do have some size which may even exceed the grid size. The horizontal distance between the two positions is defined by the horizontal grid size parameter. The vertical distance between two positions is defined by the horizontal grid size parameter and the aspect ratio parameter.

The distance between two positions can be bigger or smaller than the pixel-spot size. This relation is totally defined by the device, and may vary from device to device, and vary on the same device from one set of parameters to another.

The actual grid sizes available on the device may not match the exact grid size specification. In this case, the device will select a grid which best represents the specified grid. This 'best fit' grid is often called the 'actual grid' or 'actual grid size'. Note that most imaging operations deal with the actual grid and not the exact specified grid.

Grid Size - The distance between two adjacent grid points. The grid size may be different in the horizontal and vertical directions.

Dot/Pixel/Pixel-Spot Relationship - An imaging device may use several dots to represent a single pixel via a pixel-spot. Multiple dots can be used to cover an area larger than a dot size, or to produce some level of darkness, or to obtain some level of scaling.

Raster - All of the pixels used to define a single image. For purposes of this document, all of the pixels defined in a single Sixel control string.

Raster Size - The resultant size of the raster after it has been printed based on the actual grid size. When used in the context of pixels not yet printed, it is just the number of horizontal and vertical pixels of the raster.

Overlap - The percentage of the pixel-spot that is larger than the grid size. Overlap may be visually pleasing to smooth certain images, or may be used to cover device registration errors, or may be due to the device having a pixel-spot size more limited than the grid size.

Sixel Active Position - The Sixel active position is that position to which the next Sixel is to be printed. The actual Sixel is printed with a vertical offset to the Sixel active position so that machines which use the baseline of a character for positioning in ANSI text mode will be compatible with devices that position based on the ascender height of the character. This offset is in the vertical direction, the amount of 70 decipoints (.0972 inch). (See section - ANSI Text Interactions, While In Sixel Mode).

Picture Definition - This is all the data needed to describe the image, including colors, size, pixel aspect ratio, and encoded rasters (Sixel Data and Sixel Commands). It does not include formatting information such as position or actual presentation size.

Sixel - A Sixel is a group of 6 vertical pixels represented by 6 bits within a character code.

Sixel Data - This term includes only the encoded raster portion of the picture definitions.

Sixel Commands - These are commands included in the picture definition which provide additional information beyond the encoded raster, such as color, line breaking, etc.

4 LEVELS AND EXTENSIONS

While the Sixel protocol specification allows more device variations than the majority of the other SRM specifications, there are still several specific function sets that must be followed for compatibility. There are two levels and one extension defined for conforming devices. The levels are for variable grid size, while the extension is for color.

4.1 LEVEL 1

Level 1 devices do not support the Set Raster Attribute command, Background Select, Horizontal Grid Size or Macro Parameter commands. Level 1 devices will parse these commands, but only print in one grid size. The level 1 grid size is that defined for Macro Parameter 1.

4.2 LEVEL 2

Level 2 devices support everything a Level 1 device supports, plus the Set Raster Attribute command, Background Select, Horizontal Grid Size and Macro Parameter commands.

4.3 EXTENSION - COLOR

Color is an extension to either level 1 or level 2 devices.

5 SIXEL PRINTING

Sixel printing consists of setting context and attributes for the pixels and then printing each Sixel in received order.

A Sixel is a group of 6 vertical pixels represented by 6 bits within a character code. A one value for a bit indicates that a pixel-spot will be placed at the corresponding grid position, a zero will cause the corresponding grid position to be unchanged (not written).

Upon entering Sixel mode, the current Sixel position is determined from the ANSI text position and is called the graphic left margin.

The Sixel active position is advanced to the next horizontal grid position as each Sixel is printed. Also, there are a few line control commands that modify the Sixel active position. Positioning is always relative to the Sixel active position, and can not go backwards except for returning to the graphic left margin via the graphic carriage return and next line commands.

The color of a pixel-spot is determined by the currently selected color in the color map, as specified with a color specifier control code. The color map is defined by optional parameters of the color specifier. Note that all 1 bits of a Sixel are imaged in the selected color. Therefore, if the image is to have pixels of different color that fall within a single Sixel, only those pixels of the current color should be set to a 1, and the line of Sixel data sent again with a new color selector and the appropriate 1 bits set for this color. A line of sixels must be sent (with a graphic carriage return between them), for each color that falls in the Sixel line.

Horizontal and vertical direction follow the ANSI text horizontal and vertical axes at the time Sixel mode was entered, with Sixel drawing always proceeding from left to right, top to bottom.

6 PROTOCOL STRUCTURE

The data and commands of the Sixel Graphics protocol can be conceptually separated into three primary fields:

1. Picture Definition - the body of the Sixel protocol which contains all the information needed to reproduce the image, regardless of size. The Picture Definition includes Sixel data and Sixel command codes. The Sixel data is the encoded image raster while the Sixel command codes control how the raster will be interpreted.

   The picture definition is used by creation software, editing software, and imaging devices. It is intended that picture generating software will be able to create a file that contains the image definition independent of formatting information.

2. Formatting information - This data is added to the picture definition by the page composition software, defining the size of the picture data. The size is included in the Protocol Selector (see the example below) so that the parameters may be added without affecting the picture definition data.

   In general, composition software need not interpret the picture definition data, although in certain cases it may need to determine the number of pixels in the picture definition by looking at the pixel extent parameters.

3. Positioning data - used by page composition software. The first pixel position is defined by the ANSI text active position at the time that the Sixel mode is entered. All other pixel positions are relative to the first based on the actual grid size and aspect ratio.

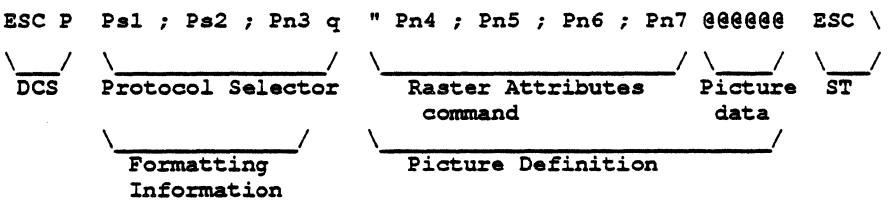
6.1 SIXEL CONTROL STRING

The formatting and picture definition for a Sixel graphic is contained in a Device Control String (DCS) envelope as defined in the "Code Extension" section of this standard, DEC STD 070-3.

```
DCS   Formatting Information   Picture Definition   ST
```

DCS (9/0) is the ANSI C1 control that begins a Device Control String (may also be coded as ESC P (1/11, 5/0) in 7-bits). The Formatting Information is represented by one or more Device Control String parameters separated by semicolons ";" (3/11) as defined by the code extension layer. The final character of the DCS Sequence is q (7/1) which identifies the Control String as containing Sixel data. The DCS parameters and final character together are called the Protocol Selector. The Picture Definition consists of Sixel data and commands as described in this standard. ST (9/12) is the ANSI C1 control that terminates a Device Control String (may also be coded as ESC \ (1/11, 5/12) in 7-bits).

6.1.1 Example Sixel Control String



Where:

Ps1 Macro Parameter
Selects from a list of fixed combinations for other parameters. Included for backward compatibility. New software should always specify this by 0 (zero).

Ps2 Background Select
0 device default action
1 no action (don't change zero value pixels)
2 set zero value pixels to background color

Pn3 Horizontal Grid Size
Given in units specified by ANSI SSU (default is decipoints, 1/720 inch).

q Final Character of DCS Introducer Sequence

" Sixel Control Code for Raster Attributes Command

Pn4 Pixel aspect ratio numerator
(first parameter of Raster Attributes Command)

Pn5 Pixel aspect ratio denominator

Pn6 Horizontal Extent
Number of pixels in image horizontally

Pn7 Vertical extent
Number of pixels in image vertically
(last parameter of Raster Attributes Command)

7 FORMATING INFORMATION

The formatting information currently consists of up to three parameters included in the Sixel DCS Introducer Sequence (Sixel Protocol Selector) by the page composition software. The following sections define each parameter of formatting information.

7.1 MACRO PARAMETER

The first parameter (Macro parameter) selects from a list of fixed combinations of values for other parameters in the Sixel data. This parameter exists for compatibility reasons and is to be set to zero in new software. New software is expected to define the three parameters explicitly. Explicit parameter definitions over-ride the macro settings. The following table defines the intent of the macro value:

Macro value	Horizontal Grid Size (X .001")	Pixel Aspect Ratio Vert pix : Horz pix
0 (default)	7.5	200 : 100
1	7.5	200 : 100
2	3	450 : 100
3	4.5	300 : 100
4	6	250 : 100
5	7.5	200 : 100
6	9	150 : 100
7	10.5	130 : 100
8	12.0	112 : 100
9	13.5	100 : 100

The intended vertical grid size implied by the macro value is always 1/72 of an inch.

7.2 BACKGROUND SELECT

The second parameter (Background Select) specifies what is to be printed in grid positions not explicitly set to a one in the Sixel data. It is device dependent if the grid positions include all possible printable grid positions of the device, or only those grid positions within the horizontal and vertical extent parameters of the Set Raster Attribute command. Hardcopy devices will process this parameter as no action (as if parameter = 1).

The possible values for this parameter are:

0 device default action

1 no action (no change to zero bit value grid positions)

2 set to background color - zero bit value grid positions are set to the background color. The background color is device dependent.

7.3 HORIZONTAL GRID SIZE

The third parameter (Horizontal Grid Size) defines the horizontal grid size in units selected by the ANSI text Set Size Unit (SSU) command. This parameter, in conjunction with the pixel aspect ratio, defines the grid size.

If this parameter is zero or not present, the horizontal grid size is given by the macro parameter, otherwise, the horizontal grid size parameter overrides the macro parameter.

If the device does not implement the SSU command, the units will be the default value for the SSU command which is Decipoints (1/720 inch).

8 PICTURE DEFINITION

The Picture Definition consists of Sixel commands and Sixel data which describe a raster image. Sixel commands are coded as graphic characters in the range 2/1 to 3/14. Sixel data is coded as graphic characters in the range 3/15 to 7/14.

8.1 SIXEL COMMANDS

Sixel commands are used to control how the image raster is interpreted, and may be embedded anywhere in the picture definition.

A Sixel Command consists of a code in the range 2/1 through 3/14 (except parameters and parameter separators) which identifies the command, followed by zero or more parameters separated by parameter separators. Each parameter is formed by one or more consecutive decimal digits 0 through 9 (3/0 through 3/9). Commands having more than one parameter must use the parameter separator ; (3/11) between parameters. A Sixel command is terminated by any non parameter code, that is, any code other than 3/0 through 3/9 (parameter digits 0-9) or 3/11 (parameter separator ;). Note Format Effectors (0/8 to 0/13) and SPACE (2/0) are ignored within sixel control strings (see coding section).

The following list summarizes the currently defined Sixel Commands and their coding. Sixel Commands that are not recognized by the device must be parsed and ignored.

```
 Code       Command
! (2/1)     Repeat Introducer
              1st parameter - repeat count

" (2/2)     Set Raster Attributes
              1st parameter - Pixel aspect ratio numerator
                              (relative height)
              2nd parameter - Pixel aspect ratio denominator
                              (relative width)
              3rd parameter - Horizontal Extent
              4th parameter - Vertical Extent

# (2/3)     Color Introducer
              1st parameter - color number
              (others optional)
              2nd parameter - color coordinate system
              3rd parameter - Hue angle,  0-360   Red,    0-100
              4th parameter -         Lightness,  0-100   Green, 0-100
              5th parameter - Saturation, 0-100   Blue,   0-100

$ (2/4)     Graphics Carriage Return

- (2/13)    Graphics Next Line

; (3/11)    Parameter Separator

0 through 9 (3/0 through 3/9) Parameter digits
```

8.1.1 Repeat Introducer

The Repeat Introducer command allows sixels to be run-length encoded by specifying a repeat count before a sixel. The repeat introducer is followed by a numeric value which indicates the number of times the next sixel is to be repeated. A repeat count of zero implies a repeat count of 1. The maximum value for the repeat count is 32767. If no sixel data character follows the repeat count (before the STRING TERMINATOR or another Sixel Command), the repeat count is ignored. The start of another Sixel Command will cancel a pending repeat count.

```
Format:

    !     Pn1   <sixel-column-code-to-be-repeated>
    2/1         (3/15 - 7/14)

Where:

    Pn1  repeat count
```

8.1.2 Set Raster Attributes

The Set Raster Attributes command defines raster attributes for all of the following data within a sixel string. This command must precede any Sixel, or any sixel positioning command within a single sixel control string or the command will be ignored.

```
Format:

    "    Pn1 ; Pn2 ; Pn3 ; Pn4
    2/2

Where:

    Pn1  Pixel aspect ratio numerator
    Pn2  Pixel aspect ratio denominator
    Pn3  Horizontal Extent
    Pn4  Vertical Extent
```

The first two parameters of the Set Raster Attributes command (Pn1 and Pn2) specify the Pixel Aspect Ratio. This ratio defines the shape of the pixel needed to reproduce the picture without distortion. The ratio is defined by two numbers, a numerator and a denominator, and is the ratio of the vertical to the horizontal shape of the pixel.

For example, if a pixel were to be half as wide as tall, the pixel aspect ratio would be 2:1, or 200:100. The Pixel Aspect Ratio times the horizontal grid size yields the vertical grid size.

The third parameter of the Set Raster Attributes command (Pn3) specifies the Horizontal Extent, the horizontal size in pixels of the intended image defined by the picture data.

This parameter allows trailing background Sixel data to be omitted from the picture definition and still allow for color background, and to provide a summary to avoid a complete scanning of the data for the horizontal limits.

This parameter does not limit the number of pixels a device may image. In the case of receiving more pixels than the extent defines, they will be imaged normally. Thus, the receiving device will NOT clip the image to the horizontal extent. Conforming software will not transmit Sixel data which extends beyond the specified horizontal extent.

The fourth parameter of the Set Raster Attributes command (Pn4) specifies the Vertical Extent, the vertical size in pixels of the intended image defined by the picture data. This parameter has the same functionality vertically as the horizontal extent has horizontally. The receiving device will NOT clip the image to the vertical extent. Conforming software will not transmit Sixel data which extends beyond the specified vertical extent.

Conforming software shall specify all parameters explicitly. Guideline: omitted parameters for Pn1 and Pn2 may be interpreted as '0' (zero) and default to '1' (one). ";0 may be interpreted as "1;1 for example.

8.1.3 Color Introducer

The Color Introducer Command starts a color selection sequence. It is followed by a color number and optional additional parameters. The color number always specifies the color number with which following Sixel data will be presented.

```
Format:

    #    Pc ; Pu ; Px ; Py ; Pz
    2/3     \_________________/
             optional parameters

Where:

    Pc  color number
    Pu  color coordinate system
    Px  Hue angle,  or Red RGB coordinate
    Py  Lightness,  or Green RGB coordinate
    Pz  Saturation, or Blue RGB coordinate
```

If a color identifier appears in the data stream with no additional parameters, it merely selects the color for following Sixel data. If followed by additional parameters, the color specifier represents a new definition for the color number. The parameters select a color in a universal color coordinate system as follows:

```
Pc  is the color number, 0 to 255 (or larger device limit).

Pu  is the universal coordinate system:
        0 - illegal
        1 - HLS (hue/lightness/saturation)
        2 - RGB (red/green/blue)

Px, Py, Pz are the color coordinates in the specified system:
                          HLS                 RGB
        Px      Hue angle, 0-360.       Red, 0-100.
        Py      Lightness, 0-100.       Green, 0-100
        Pz      Saturation, 0-100.      Blue, 0-100.
```

Values for Pu, Px, Py, and Pz other than those specified above cause the entire color specifier command to be ignored, including the color select parameter, Pc. Conforming software shall specify all parameters to be interpreted explicitly. If optional parameters are included to define a new color, they should all be specified. Guideline: omitted parameters for Px, Py, and Pz may be interpreted as '0' (zero); parameter value '0' (zero) for Pu will cause the entire command to be ignored.

Devices must implement color numbers 0 to 255. Devices may implement color numbers larger than 255. Conforming software will use only numbers less than 255, and will use color numbers in sequence starting from 0. In the case of an index which exceeds the bounds of the device, the unimplemented index must be mapped back into the supported range.

Note this command may load or change the device's color table.

8.1.4 Graphics Carriage Return

The Graphics Carriage Return command causes the Sixel active position to move to the graphic left margin. This Sixel control code is the only code which allows rewriting of a Sixel position and thus is required for writing more than a single color into a Sixel position.

```
Format:

    $
    2/4

    (Graphics Carriage Return does not accept any parameters)
```

8.1.5 Graphics Next Line

The Graphics Next Line command causes the Sixel active position to move to the graphic left margin and down one row of sixels (six actual grid units).

```
Format:

    -
    2/13

    (Graphics Next Line does not accept any parameters)
```

8.2 SIXEL DATA

Sixel Data is used to represent the pixels of the raster image in columns of six pixels per character.

Sixel data is coded as graphic characters in the range 3/15 to 7/14. The six pixels to image are derived by subtracting the offset (63 decimal) from the code and assigning each of the low order 6 bits to a grid position. The six pixels are arranged vertically as follows:

```
Top pixel       Bit 0 (LSB)
                Bit 1
                Bit 2
                Bit 3
                Bit 4
Bottom pixel    Bit 5 (MSB)
```

For example, if the character code 4/1 (ASCII A) is received, the offset value (63 decimal) is subtracted from the code value (65 decimal). The resulting value of 2 is mapped into the horizontal scan as follows:

```
                    .
Data Bits:      5  4  3  2  1  0
    2 =         0  0  0  0  1  0
Scan:  (top)    1  o
               .2  x
                3  o
                4  o
                5  o
       (bottom) 6  o
```

where "o" indicates the the pixel spot is not printed and "x" indicates that the pixel spot is printed.

8.2.1 Sixel Column Codes

Character	?	@	A	B	C	D	E	F
Octal	077	100	101	102	103	104	105	106
Decimal	63	64	65	66	67	68	69	70
Hexadecimal	3F	40	41	42	43	44	45	46
Dot Pattern	o	x	o	x	o	x	o	x
	o	o	x	x	o	o	x	x
	o	o	o	o	x	x	x	x
	o	o	o	o	o	o	o	o
	o	o	o	o	o	o	o	o
	o	o	o	o	o	o	o	o

Character	G	H	I	J	K	L	M	N
Octal	107	110	111	112	113	114	115	116
Decimal	71	72	73	74	75	76	77	78
Hexadecimal	47	48	49	4A	4B	4C	4D	4E
Dot Pattern	o	x	o	x	o	x	o	x
	o	o	x	x	o	o	x	x
	o	o	o	o	x	x	x	x
	x	x	x	x	x	x	x	x
	o	o	o	o	o	o	o	o
	o	o	o	o	o	o	o	o

Character	O	P	Q	R	S	T	U	V
Octal	117	120	121	122	123	124	125	126
Decimal	79	80	81	82	83	84	85	86
Hexadecimal	4F	51	51	52	53	54	55	56
Dot Pattern	o	x	o	x	o	x	o	x
	o	o	x	x	o	o	x	x
	o	o	o	o	x	x	x	x
	o	o	o	o	o	o	o	o
	x	x	x	x	x	x	x	x
	o	o	o	o	o	o	o	o

Character	W	X	Y	Z	[	\	]	^
Octal	127	130	131	132	133	134	135	136
Decimal	87	88	89	90	91	92	93	94
Hexadecimal	57	58	59	5A	5B	5C	5D	5E
Dot Pattern	o	x	o	x	o	x	o	x
	o	o	x	x	o	o	x	x
	o	o	o	o	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	o	o	o	o	o	o	o	o

Character	_	`	a	b	c	d	e	f
Octal	137	140	141	142	143	144	145	146
Decimal	95	96	97	98	99	100	101	102
Hexadecimal	5F	60	61	62	63	64	65	66
Dot Pattern	o	x	o	x	o	x	o	x
	o	o	x	x	o	o	x	x
	o	o	o	o	x	x	x	x
	o	o	o	o	o	o	o	o
	o	o	o	o	o	o	o	o
	x	x	x	x	x	x	x	x

Character	g	h	i	j	k	l	m	n
Octal	147	150	151	152	153	154	155	156
Decimal	103	104	105	106	107	108	109	110
Hexadecimal	67	68	69	6A	6B	6C	6D	6E
Dot Pattern	o	x	o	x	o	x	o	x
	o	o	x	x	o	o	x	x
	o	o	o	o	x	x	x	x
	x	x	x	x	x	x	x	x
	o	o	o	o	o	o	o	o
	x	x	x	x	x	x	x	x

Character	o	p	q	r	s	t	u	v
Octal	157	160	161	162	163	164	165	166
Decimal	111	112	113	114	115	116	117	118
Hexadecimal	6F	70	71	72	73	74	75	76
Dot Pattern	o	x	o	x	o	x	o	x
	o	o	x	x	o	o	x	x
	o	o	o	o	x	x	x	x
	o	o	o	o	o	o	o	o
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x

Character	w	x	y	z	{	|	}	~
Octal	167	170	171	172	173	174	175	176
Decimal	119	120	121	122	123	124	125	126
Hexadecimal	77	78	79	7A	7B	7C	7D	7E
Dot Pattern	o	x	o	x	o	x	o	x
	o	o	x	x	o	o	x	x
	o	o	o	o	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x

9 CODING ISSUES

9.1 C0 CODES

The inclusion of C0 Controls 0/8 to 0/13 in control string data is intended for convenience in formatting and storing control strings. These codes shall have no effect on the interpretation of Sixel Control strings.

Codes in the range 0/0 through 0/7 and 0/14 through 1/15 are considered errors. The communications control characters, XON (DC1) and XOFF (DC3) are handled at a different level. Cancel (CAN), Substitute (SUB), Escape (ESC), Bell (BEL), and Enquire (ENQ) are treated as follows:

1. CAN - exit DCS mode processing, exit Sixel mode.
2. SUB - process as if a 3/15 code (blank Sixel) was received - this is to limit the affect of some communication line errors.
3. ESC - exit DCS mode processing, exit Sixel mode, begin processing escape sequence.
4. BEL - same action as in text mode.
5. ENQ - same action as in text mode.

All other C0 Codes are ignored.

9.2 GL CODES

Codes in the range 2/0 through 7/15 are treated as follows:

```
2/0 (SPACE)        Ignored
2/1 - 3/14         Sixel Commands
3/15 - 7/14        Sixel Data
7/15 (DELETE)      Ignored
```

9.3 C1 CODES

Codes in the range 8/0 through 9/15 exit DCS mode processing, exit Sixel mode, and the C1 control code is processed outside of the Sixel mode.

9.4 GR CODES

Codes in range 10/0 through 15/15 are errors. The 8th bit is set to zero and the code is then processed as a GL code. Conforming software shall not transmit GR codes.

10 INITIAL STATES

No Sixel state is preserved from one entering of Sixel mode to the next. All Sixel state is reset when the Sixel mode is entered.

The following states are initialized when Sixel mode is entered:

```
Background color:     Device default background.

Sixel color:          Contrasting to background.

Color numbers:        All unspecified color numbers image as
                        the default Sixel color.

Horizontal extent:    Distance from current ANSI text position
                        to current ANSI right margin.

Vertical extent:      Distance from current ANSI text position
                        to current ANSI bottom margin.
```

Note that the raster attributes (horizontal grid size, pixel aspect ratio) are initialized by the Macro Parameter, even if unspecified and defaulted to zero.

Conforming software will always initialize the horizontal and vertical extent. Conforming software will also initialize any color numbers used in the picture definition.

11 ANSI TEXT INTERACTIONS

The Sixel protocol has certain interactions with the ANSI text portion of the device. Portions of the ANSI text state affect Sixel printing when Sixel mode is entered, and the results of Sixel printing affect portions of the ANSI text state when Sixel mode is exited. The majority of interaction is in the area of active position, although margins are also involved.

11.1 ENTERING SIXEL MODE

The Sixel active position is set equal to the ANSI text active position when Sixel mode is entered. The center of the top dot is printed 70 decipoints (.0972 inch) above the character baseline for compatibility with devices which use cell positioning. (Explanatory note: 70 decipoints is consistent with the standard spacing of multi-wire print heads.) The horizontal and vertical axes are established as the same as the ANSI horizontal and vertical axes. No other information is carried from the ANSI text state; in particular, the ANSI SGR (Select Graphic Rendition) selections DO NOT affect sixels.

11.2 WHILE IN SIXEL MODE

11.2.1 Printing at The Active Position

Sixels are printed relative to the active position such that the top pixel of the Sixel is placed at the current horizontal position and seven actual grid vertical sizes above the current vertical position.

GUIDELINE:

On devices with significantly higher resolution than the vertical grid size, sixels should be processed the same as a font with an above baseline of 70 decipoints, and a below baseline of 6*(vertical grid) - 70 decipoints. If the first character flag is set when printing sixels, the Active Position will first be moved down by 70 decipoints (the above baseline amount). If the Active Position is closer to the Bottom Margin than the below baseline value, a wrap condition will occur. Graphic new line will act like a LF of 6*(vertical grid), and wrap at the page end position.

11.2.2 Right Margin

Sixels defined to be printed past the right margin are not printed.

11.2.3 Top Margin

If the Sixel active position is such that printing would occur above the top margin, then that part of Sixel will image above the top margin.

11.2.4 Bottom Margin

Advancing the Sixel active position past the bottom margin has an effect similar to that of ANSI text. In video devices, advancing past the bottom margin will cause scrolling to occur (if text drawn into this region would have scrolled). In printers, the action will also be similar to that of text causing the active position to advance to the next page consistent with the setting of Form Mode and other parameters (normal Sixel row vertical spacing can be maintained on continuous form machines).

11.3 EXITING SIXEL MODE

The Horizontal Active position of the ANSI text is not affected by Sixel mode and set exactly to what is was when Sixel mode was entered.

The vertical active position of the ANSI text is affected by Sixel mode. The ANSI text vertical position is updated to reflect the vertical motion commands executed in the Sixel printing. Note that on line oriented devices, this may leave the ANSI text active position with a non integer line value, only getting back on line boundaries after an absolute vertical position command. Also, the vertical positioning commands of Sixel mode use the ACTUAL grid size, which may be different than the specified grid size. Therefore, software doing pagination should issue an absolute position command before placing characters to insure placement as expected.

Some devices may require re-alignment to a grid established for ANSI text after leaving Sixel mode. In this case, an adjustment less than one character height may occur in a downward direction. Conforming software will not assume any exact alignment of the ANSI text vertical position following a Sixel image.

12 FALLBACK

As noted in the introduction, this protocol is quite flexible in the range of imaging formats provided. It is recognized that few (if any) devices will be able to faithfully image all of the rasters defined by the protocol. The purpose of this section is to specify fallback and best fit algorithms and techniques.

The commands allowing fallback are the interpretation of Pixel aspect ratio, Horizontal grid size, and color selection.

A device may require that the grid size used for imaging the raster be different than that specified by the Pixel aspect ratio and horizontal grid size. In such a case, the grid size used for imaging the raster is called the "actual grid size". Note that several commands use this actual grid size specifically, not the ideal grid size specified. Conforming devices must clearly state in their product documentation how grid sizes fall back.

The general rule for grid size fall back is that a device selects an actual grid size that will come closest to but does not exceed the specified grid size, but never less than the smallest imageable unit of the device. (That is, if the smallest size that a device can print is larger than the specified grid size, the device prints it anyway, using the smallest unit as the actual grid).

Since some devices may have high frequency grid size errors that do not accumulate, the average grid size over not more than 30 pixels is used to determine the actual grid size. These non-accumulative errors may be due to devices mechanics (two different resolution mechanism working together like a print head and a paper motor), or due to implementation of a scaling algorithm that combines a non-integer number of pixels into an integer number of device print units.

The pixel aspect ratio should be preserved if possible.

GUIDELINE:

Many common grid size specifications were intended for devices with coordinate units based on 1/72" (equal to 1 point or 10 decipoints). When converting to 150/300/600 dpi devices, the proper fallback is to use 1/75" for each decipoint instead of 1/72". This results in a slightly smaller image (4%), but avoids pixel interference patterns from attempting to scale to a non integral number of device pixels.

There are two classes of color fallback - black and white (monochrome), and limited true color.

For monochrome devices, there are several levels of fallback that are device dependent. The most common and the least that conforming devices must do is to write all the one bits in the only color available and not write the 0 bits. Another fallback would be to texture or halftone each pixel, or even texture or halftone an area of pixels.

For color devices that can not image all the colors possible, mapping of the color definitions to the device set is acceptable. The following eight-color mapping shall be used for eight color devices using a subtractive color process.

HLS Coordinate System

Hue Angle	Intended HLS Color	Fallback Color
31 - 90	Magenta	Magenta
91 - 150	Red	Red (Yellow + Magenta)
151 - 210	Yellow	Yellow
211 - 270	Green	Green (Yellow + Cyan)
271 - 330	Cyan	Cyan
331 - 30	Blue	Blue (Magenta + Cyan)

Lightness

0% - 14%	Disregard hue angle, print black
15% - 85%	Print based on hue angle
86% - 100%	Disable Printing (White)

Saturation is ignored, assumed to be 100%

RGB Coordinate System

Red	Green	Blue	Intended Color	Fallback Colors
0	0	0	Black	Black(Yellow+Magenta+Cyan)
1-100	0	0	Red	Red (Yellow + Magenta)
0	1-100	0	Green	Green (Yellow + Cyan)
1-100	1-100	0	Yellow	Yellow
0	0	1-100	Blue	Blue (Magenta + Cyan)
1-100	0	1-100	Magenta	Magenta
0	1-100	1-100	Cyan	Cyan
1-100	1-100	1-100	White	(None)

0 = selected color is not used
1-100 = selected color is used

12.1 DEVIATIONS

12.1.1 VT125

1. Initialization - The VT125 does not reset the Sixel writing color to contrast to the background.
2. Relation to ANSI text position - The VT125 does not start the Sixel position at the ANSI text position but rather at the upper left corner of the display. It does not update the ANSI text vertical position at the termination of a Sixel image.
3. Color numbers - The VT125 accepts color numbers 0, 1, 2, 3 but does not allow color specifiers.
4. Background select - The VT125 does not accept this parameter. It always uses the background color for unwritten sixels by erasing the screen to background (i.e. extent is the entire screen).
5. Scrolling - The VT125 does not scroll when the bottom margin is reached.
6. Raster attributes - The only raster attributes accepted by the VT125 is defined by Macro Parameter value of 1. Thus all other combinations of raster attributes fall back to this selection.

12.1.2 VT240

1. Initialization - The VT240 does not reset the Sixel writing color to contrast to the background. It retains the colors from the previous Sixel transmission.
2. Relation to ANSI text position - The VT240 does not start the Sixel position at the ANSI text position but rather at the upper left corner of the display. It does not update the ANSI text vertical position at the termination of a Sixel image.
3. Horizontal and Vertical Extent - The VT240 assumes that the extent of the image is the entire screen when Background Select is specified to fill in the background; thus the VT240 erases the screen to background in this case.
4. Scrolling - The VT240 does not scroll when the bottom margin is reached.

5. Raster attributes - The only raster attributes accepted by the VT240 is defined by Macro Parameter value of 1. Thus all other combinations of raster attributes fall back to this selection.
6. Horizontal Grid Size parameter - The VT240 interprets this parameter as a different function. The effect of specifying this parameter has very limited effects.

12.1.3 LA50

1. The LA50 has a hardware switch to select between 2:1 and 2.5:1 pixel aspect ratios. 2:1 is required for Level 1 conformance.

12.1.4 LA100 and LA210

1. The LA100 and LA210 are Level 2 devices (recognize the Macro Parameter), but do not recognize the set raster attributes command.

   Guideline: Software that wishes to support these devices explicitly may set the Macro parameter for the desired characteristics, but should also specify the horizontal grid size (Pn3) and raster attributes to over-ride the Macro parameter for use on other Level 2 devices.

12.2 CHANGE HISTORY

12.2.1 Revision AX11 to AX12

General

1. Removed Revision AX11 change bars. Added change bars to any change that could affect conformance or interpretation of the document or should be brought to the attention of terminal implementors or software engineers.
2. Re-organized main body of the standard to combine syntax and semantic information, making it easier for implementors to follow.

Terminology Section

3. Removed definition of "Raster Aspect Ratio" since the term is not used anywhere in the chapter.
4. Clarified "Picture Definition" by indicating it consists of Sixel Data and Sixel Commands.
5. Added definition of Grid Size; the distance between two adjacent grid points.
6. Added definition of SIXEL.

Macro Parameter

7. Added sentence: The intended vertical grid size implied by the macro value is always 1/72 of an inch.
8. Changed the pixel aspect ratio for macro value 5 from 183:100 to 200:100. Macro value 5 is intended to give a 200:100 pixel aspect ratio (as on the LN03). 183:100 is the fallback on the LA12/34/100/210.

Sixel Commands

9. Added format descriptions to each Sixel command.

10. Clarified meaning of "ignoring" unrecognized sixel commands to indicate command and parameters must be parsed and ignored.
11. Clarified behavior of Repeat Introducer when sixel column code does not follow immediately. Starting another sixel command will cancel a pending repeat count.
12. Added material on Set Raster Attributes Command:

    "Conforming software shall specify all parameters explicitly. Guideline: omitted parameters for Pn1 and Pn2 may be interpreted as '0' (zero) and default to '1' (one)."

    Clarified that the Set Raster Attributes command must precede any sixel, or any sixel positioning command within a single sixel string or the command will be ignored.
13. Added material on Color Introducer command:

    "Conforming software shall specify all parameters to be interpreted explicitly. If optional parameters are included to define a new color, they should all be specified. Guideline: omitted parameters for Px, Py, and Pz may be interpreted as '0' (zero); parameter value '0' (zero) for Pu will cause the entire command to be ignored.

    "In the case of an index which exceeds the bounds of the device, the unimplemented index must be mapped back into the supported range."

    "Note this command may load or change a device's color table."

    ANSI Text Interactions
14. Clarified that the ANSI text active position is not changed when Sixel mode is entered. The center of the top dot is printed 70 decipoints above the active position. (Explanatory note: 70 decipoints is consistent with the standard spacing of multi-wire print heads.)
15. Added guideline on printing at the active position for devices with significantly higher resolution than the vertical grid size.
16. Simplified section on Bottom Margin by emphasizing that sixels should be treated the same as text, and removing duplicate material from the PSRM positioning controls chapter.

    Other
17. Added an example of a complete Sixel Control String.
18. Corrected list item numbers (1-3) which were confused with Background Select parameter values (0-2).
19. Removed SPACE (2/0) from the set of sixel control codes. Added statement SPACE shall be ignored in sixel control strings.
20. Added paragraph stating that format effectors (0/8 - 0/13) shall not effect the interpretation of sixel control strings.
21. Added paragraph to Coding section describing how GL codes are handled to make it inclusive of all 8-bit codes. SPACE (2/0) and DELETE (7/15) shall be ignored.
22. Added statement that: Conforming software shall not transmit GR codes.
23. Added table of Sixel column codes.
24. Clarified software conformance requirement for initialization. Conforming software will always initialize the horizontal and vertical extent (not just if used in the picture definition as stated for color numbers).

25. Added statement to fallback section that: Conforming devices must clearly state in their product documentation how grid sizes fall back.
26. Added guideline on fallback for 150/300/600 dpi devices.
27. Fixed legend on RGB fallback table to indicate '0' means selected color is NOT used.
28. Added LA50 to deviations section.
29. Added LA100 and LA210 to deviations section. Included guideline on use of the Macro parameter.

INDEX

```
+-------------------------------------------------------------------+
|                          READER COMMENTS                          |
|                                                                   |
| Your comments and suggestions will help Standards and Methods     |
| Control improve their services and documents.                     |
+-------------------------------------------------------------------+
```

Did you request this document? _____ If so, did it arrive within a satisfactory period of time? _____ Please comment.

What are your impressions of this document? Consider format, organization, completeness, readability, and illustrations.

FOLD ON THIS LINE

Did you find technical or clerical errors in this document? If so, please specify the page number(s) and the error(s).

Are the instructions for the update package clear? _____
Was an index available? _____ If not, is one needed? _____
Do you have other suggestions for improving this document?

The following information is optional:

Name _____ Mailstop _____

Department _____ Node _____

Send your comments to JOKUR::PROJECTS, or fold, staple, and send this page through interoffice mail to:

```
+-------------------------------+
|       READERS' COMMENTS       |
| STANDARDS AND METHODS CONTROL |
|           CTS1-2/D4           |
+-------------------------------+
```

DEC STD 070-10 Video Systems Reference Manual, Dynamically Redefinable Character Sets Extension

DOCUMENT IDENTIFIER: A-DS-EL00070-10-0000 Rev A, 04-Oct-1990

ABSTRACT: This document describes the interface to load and designate Dynamically Redefinable Character Sets (DRCS), which forms extensions to Levels 2, 3 and 4 of the Character Cell Display service classes and Levels 1 and 2 of the Hardcopy Display service classes.

APPLICABILITY: Mandatory for engineers designing hardware for terminal products and software engineers designing programs using terminal interfaces. Additional requirements are defined in the DEC STD 070-1 Concepts and Conformance Criteria.

STATUS: APPROVED 04-Oct-1990; use VTX SMC for current status.

DEC STD 070-10 Video Systems Reference Manual, Dynamically Redefinable Character Sets Extension

DOCUMENT IDENTIFIER: A-DS-EL00070-10-0000 Rev A, 04-Oct-1990

REVISION HISTORY:	Original Draft	30-Oct-1982
	Revision 0.1	25-Dec-1982
	Rev A,	04-Oct-1990
Document Management Category:		Terminal Interface Architecture (STI)
Responsible Department:		Video, Image, and Printing Systems (VIPS) Terminals Architecture
Responsible Person:		Peter Sichel

APPROVAL: This standard, prepared by the Desktop Systems Group, has been reviewed and recommended for approval by the General Review group for its category.

Peter A. Sichel

Peter Sichel, Video, Image, and Printing Systems

Eric A Williams

Eric Williams - Standards Process Manager

Direct requests for further information to:

Peter Sichel
Use $ VTX ELF for the latest location information.

Use VTX SMC to order copies of this document from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

CONTENTS

1 INTRODUCTION

Dynamically Redefinable Character Sets (DRCS) are an extension to levels 2, 3 and 4 of the Character Cell service class and levels 1 and 2 of the Hardcopy Display service class. They provide the capability of loading into the terminal a font description which can be designated and invoked as a resident character set. The font loading format as defined for DRCS is flexible enough to be applicable to a wide variety of devices, including display and printing devices of various resolutions, and varying amounts of font storage capability.

In this document (DEC STD 070-10), the term "font file" refers to a sequence of bit patterns that define the graphic symbols of a coded character set.

1.1 VIDEO CHARACTER CELL DISPLAY REQUIREMENTS

In Level 2 only one 94-character font buffer is required for conformance. Because Level 2 is a Character Cell Display, the font files which can be loaded are fixed width only. A 94- or 96-character font buffer is required for conformance in Level 2 if the device supports a 96-character graphic set designation. The 96-character graphic set designation is part of the 8-bit Interface Architecture Extension. In Level 3 and 4, only one character font buffer is required to support either a 94- or 96-character set for conformance; support of 96-character graphic sets is required in Level 3 or higher. Refer to *DEC STD 070-3 Code Extension Layer* for more information.

For Level 2, 3 or 4 conforming devices, the minimum number of logical pixels per character in the DRCS font buffer is 80, defining a character cell matrix 8 wide by 10 high. This does not include inter-character space for text characters, which is provided automatically when text format is specified.

1.2 HARDCOPY CHARACTER CELL DISPLAY REQUIREMENTS

For devices which conform to Level 1 or 2 of the Hardcopy Display service class, a minimum of three character font buffers is required, each supporting either a 94- or 96-character graphic character set; support of 96-character graphic character sets is required in all levels of the Hardcopy Display service class. The number of logical pixels per character is device-dependent but shall support at least the resolution of the draft-quality resident font files. Hardcopy DRCS fonts may use the whole character cell, including inter-character space.

2 TERMINOLOGY

Cell Matrix Size - The width and height in logical pixels of a character cell.

Character Cell - A rectangular bit pattern which defines the foreground and background values of a single character rendition.

Coded Character Set - A set of named characters and their one-to-one relationship with bit combinations.

Control String - A class of control functions (bit strings) introduced by a special bit combination and terminated by a string terminator, the interpretation of which is dependent on the introductory character sequence.

DCS Introducer Sequence - The sequence of characters in a Device Control String which immediately follows the DCS control code and precedes the command string. Within Digital, DCS Introducer Sequences have the same format and follow the same parsing rules as control sequences.

Designate - To identify a set of characters that are to be represented, in some cases immediately and in others on the further occurrence of a control function, in a prescribed manner.

Font - An instance of a type family having a specific type size, style, characteristic weight, and character proportion.

Font File - A sequence of bit patterns or picture descriptors which define the graphic symbols of a coded character set.

Escape Sequence - A bit string that is for control purposes in code extension procedures consisting of two or more bit combinations, the first of which is an Escape (ESC) character.

Invoke - To cause a designated set of characters to be the prescribed bit combinations whenever those bit combinations occur, until an appropriate code extension function occurs.

Logical Pixel - A single element of a binary image, which may (but does not necessarily) map to a physical pixel in either the transmitting or receiving device.

Scan - One pass across an image in either the encoding or decoding process, consisting of the height of one sixel, or six pixels.

Set Size - A value which defines the relationship of a font definition to physical display parameters.

Sixel - A 7-bit or 8-bit character, the lower six bits of which represent binary image data.

3 FUNCTIONAL DESCRIPTION

3.1 LOADING FONT FILES

The DRCS font buffers may be down-line loaded on command from the host, using the Down-Line Load (DECDLD) control string. The DECDLD command string consists of a sequence of binary data in sixel format which defines the composition of the font file.

3.2 ASSOCIATING FONT FILES WITH CHARACTER SETS

The command string also contains parameters which specify the association between a character set designation string and the font file to be loaded into a buffer. The downloaded font file may be identified as replacing one of the character sets built into the device (for example, ASCII) or as a new character set not present in the device. Refer to *DEC STD 070-3 Code Extension Layer* and *DEC STD 070-5 Character Cell Display*.

When the character set designation string of a loaded font file which replaces a built-in character set is changed, either to another built-in set, or to a new character set, the built-in set that was replaced must be made available in the device without additional interchange.

It is recommended for new devices that when a loaded font file which replaces a built-in character set is completely erased by an application, via the Pe parameter, that the device make the original built-in set available without additional interchange (not mandatory).

3.3 DESIGNATING AND INVOKING CHARACTER SETS

Once a font has been loaded its associated character set, whether DRCS or otherwise, may be designated to any of the four G-sets (G0, G1, G2, or G3), and may invoked into either the left or right hand sides (GL or GR) of the In Use Table, according to the restrictions of ANSI code extension.

NOTE

96-character fonts may not be directly designated into G0. Refer to the "Code Extension Layer" section.

In other words, it may be used in any manner that standard character sets are used including, if applicable, as the User Preference Set.

3.4 NOTE ON FUTURE USE OF FONTS

It should be noted that the font file loading mechanism as defined in this specification for DRCS character sets may have broader application in the future, when fonts are used widely as a means of modifying the graphic rendition of a particular coded character set. In this case, the font number parameter included in the DECDLD load sequence will be used as a font identifier which may subsequently be assigned to a rendition value and selected using the Select Graphic Rendition (SGR) control function. The assignment may be accomplished using the Font Selection (FNT) control as defined in ANSI X3.64-1979 (p.37).

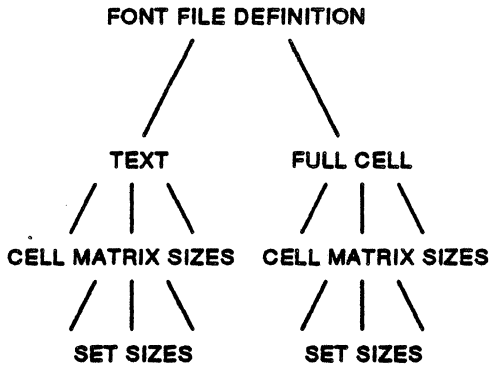
3.5 CELL MATRIX SIZE, SET SIZE, AND FONT USAGE

There are four parameters which affect the relationship between a font file definition as loaded into the font buffer memory and its physical appearance on the display. These parameters are:

1. Character Cell Matrix Size
2. Font Set Size
3. Font Usage

For any given font file, there are two possible Font Usage values, Text and Full Cell. For any given Font Usage, there are appropriate Character Cell Matrix Sizes which are typically device dependent. Furthermore, each Character Cell Matrix Size in a particular device may be limited to rendition in certain Set Sizes (80- or 132-column widths, for example).

The following diagram describes this relationship:



Because of the device dependent nature of these interactions, the product literature must be consulted for each device to determine the appropriate combination of parameters for a particular application.

4 CONTROL FUNCTIONS

DOWN LINE LOAD (FONT FILE) DECDLD

Levels:

```
2X, 3X, 4X Video Character Cell
1X, 2X     Hardcopy Display
```

Purpose: Down-line load font file patterns.

Format: The DECDLD control is a Device Control String consisting of an introducer sequence, followed by a command string. The format of the introducer sequence is:

```
DCS  Pfn ; Pcn ; Pe ; Pcmw ; Pss ; Pu ; Pcmh ; Pcss {
9/0  Parameter String ...                          7/11
```

The introducer sequence is terminated by the final character { (7/11).

The command string format is:

```
Dscs  Sxbp1 ; Sxbp2 ;...; Sxbpn  ST
```

The command string is terminated by the String Terminator character 9/12 (in 7-bits, ESC \, or 1/11 5/12). Refer to DEC STD 070-3 for a complete description of the Device Control String format.

Description:

The DECDLD control string down-line loads one or more characters of a specified 94-character or 96-character (if required) Dynamically Redefinable Character Set (DRCS) with a specified logical pixel pattern.

On video terminals, the affected characters in the DRCS set may also change the presentation of characters from this set currently being displayed on the screen. The DECDLD control does not clear the screen.

In printers, DECDLD will not affect the presentation of characters already on the page.

Notes:

1. Only character positions 2/1 through 7/14 may be loaded in 94-character sets.
2. Character positions from 2/0 through 7/15 may be loaded in 96-character sets.
3. Character cell sizes are given in logical pixels, width by height.
4. Loaded character sets are not required to be loaded into non-volatile memory. Therefore, if the terminal is powered down all loaded sets may be lost.
5. The host cannot read back loaded character sets.
6. The fonts loaded by the DECDLD command are separate and distinct from the ReGIS graphics "Load Pattern Cell" command. The ReGIS character sets may not be used except with ReGIS "text" commands.

7. A Soft Reset function (DECSTR) does not erase font files loaded by the DECDLD command (although the designation of character sets associated with those font files may change).
8. The soft character set may be cleared using any of four techniques: (1) Using the DECDLD function from the host; (2) By result of a power up test sequence; (3) From Setup - the RECALL or DEFAULT functions, (but not RESET TERMINAL); (4) On video terminals via the RIS sequence, ESC c, (but not DECSTR).
9. On video terminals, DECSCL performs a soft reset to put the terminal in a known presentation state without affecting data already displayed (does not clear DRCS). In printers, DECSCL is interpreted as "reset to base state" and will clear any DRCS.

4.1 INTRODUCER SEQUENCE FORMAT

The following selective parameters control the operation of the font load function. All parameters have the same syntax as a control sequence and may be omitted, in which case the default values are used.

4.1.1 Parameter (Pfn): Font Number

The font number parameter, Pfn, specifies the particular DRCS font buffer to be loaded, starting with buffer 1.

On video terminals, only one font buffer is required for conforming products implementing the DRCS extension. An omitted parameter, or a parameter value of zero, will default to the first empty font buffer, or, if none are empty, to font buffer 1.

On printers, three font buffers are required. An omitted parameter, or a parameter value of zero, will default to font buffer 1.

Attempts to load font buffers greater than the required number will produce results which are UNDEFINED by the architecture. Therefore, conforming software will not specify font numbers greater than the number of font buffers required. Hardware will ignore attempts to load font buffers greater than the number implemented.

4.1.2 Parameter (Pcn): First Character In Set To Load

The second parameter selects the first character in the font buffer to be loaded. Parameter value 0 means 2/0, 1 means 2/1, 2 means 2/2, ... 94 means 7/14, and 95 means 7/15.

If the device supports only 94-character buffers, (Level 2 conforming video products that do not include the 8-bit Interface Architecture Extension), attempts to start loading at location zero will cause the first character definition to be ignored, and the second character definition will be used for position 2/1. Any attempt to start loading beyond (or continue beyond) location 94 will also be ignored.

If the device supports 94- or 96-character buffers, (required for printers and Level 3 or higher conforming video products), the Pcss parameter (see below) determines whether the font file is being loaded for a 94-character or 96-character set. If a 94-character set is being loaded, the restricted 94-character buffer interpretation of Pcn applies. If a 96-character set is being loaded, attempts to start loading at location zero will load the character 2/0, and loading at a location greater than 95 will be ignored.

Conforming software loading 94-character sets or font buffers, will not specify values less than one or greater than 94. Conforming software loading 96-character fonts will not specify values greater than 95.

4.1.3 Parameter (Pe): Erase Control

The third parameter controls which characters are erased before loading takes place according to:

0 - Erase all characters in the specified font buffer (default)
1 - Erase only the characters that are loaded
2 - Erase all characters in all font buffers

A parameter value of 1 permits the changing of one or more characters in the specified font without affecting the remaining characters of that font. Characters which have been erased from the buffer and not redefined are represented in the visual display by the error character (reverse question mark).

Guideline:
For video terminals, conforming software should only use the parameter value of 2 (erase all characters in all font buffers) in response to an explicit user request.

4.1.4 Parameter (Pcmw): Character Cell Matrix Width

The fourth parameter defines the limit of the character cell matrix width in pixels of the font being loaded. Note that legal values for this parameter are dependent on the values of the Font Set Size parameter and the Font Usage parameter, and may vary from device to device.

The legal combinations of these three parameters must be clearly defined in the product literature for conforming terminals. Illegal values of the Character Cell Matrix Width parameter cause the remainder of the load sequence to be ignored. Changes to this parameter since the last DECDLD sequence to this buffer will result in the erasure of the entire set, and cause a new load to start.

Guideline:
Implementations may discard pixels beyond the width of the character cell specified by this parameter. Software can use the Character Matrix Width parameter to obtain device independent, predictable results.

Additional Requirements for Video Terminals:
For video terminals, some values of the Character Cell Matrix Width parameter also specify the Character Cell Matrix Height, as shown in the table below. If one of these parameters is specified, the Character Cell Matrix Height (Pcmh) parameter is ignored.

Possible values of the parameter for conforming devices include:

0 - Device default
1 - Not used, illegal
2 - 5 x 10 matrix (width x height)
3 - 6 x 10 matrix (width x height)
4 - 7 x 10 matrix (width x height)

5 - 5 pixel wide matrix
6 - 6 pixel wide matrix
7 - 7 pixel wide matrix
8 - 8 pixel wide matrix
9 - 9 pixel wide matrix
10 - 10 pixel wide matrix
11 - 11 pixel wide matrix
12 - 12 pixel wide matrix

A parameter value of 0 indicates that the width (or matrix) of the character cell is the device default, which is dependent on the implementing device.

Guideline:
7 x 10 is the device default for Level 2 video devices. 5 x 10, 6 x 10 and 7 x 10 character cells are always scaled for VT220 compatibility.

4.1.5 Parameter (Pss): Font Set Size (Video Terminals Only)

This parameter specifies the intended visual rendition of the font file, and is used by the device when such restriction can affect the use of the loaded set. For example, some implementations restrict a loaded font file to use in an 80-column or 132-column screen. Note that legal values for this parameter are dependent on the values of the Character Cell Matrix Size parameter and the Font Usage parameter, and may vary from device to device.

The legal combinations of these three parameters must be clearly defined in the product literature for conforming terminals. Illegal values of the Font Set Size parameter cause the remainder of the load sequence to be ignored. Changes to this parameter since the last DECDLD sequence to this buffer will result in the erasure of the entire set, and cause a new load to start.

Possible values of the parameter for conforming devices include (VT420 Example):

0 - Device default (80 columns, 24 lines, not mandatory)
1 - 80 columns, 24 lines, (normal rendition)
2 - 132 columns, 24 lines, (normal rendition)
11 - 80 columns, 36 lines, (normal rendition)
12 - 132 columns, 36 lines, (normal rendition)
21 - 80 columns, 48 lines, (normal rendition)
22 - 132 columns, 48 lines, (normal rendition)

NOTE

The values for this parameter may be extended in the future to include character pitch and other physical rendition values which may affect the use of the loaded set.

4.1.6 Parameter (Pu): Font Usage

The sixth parameter defines the loaded font file as either a text font file or as a full cell font file. Full cell font files can individually address all pixels in a cell, while text font files may not be able to address all pixels individually. Text font files may also define the centering and spacing characteristics of the area around the character cell. Note that legal values for this parameter are dependent on the values of the Character Cell Matrix Size parameter and the Font Set Size parameter, and may vary from device to device.

The legal combinations of these three parameters must be clearly defined in the product literature for conforming terminals. Illegal values of the Font Usage parameter cause the remainder of the load sequence to be ignored. Changes to this parameter since the last DECDLD sequence to this buffer will result in the erasure of the entire set, and cause a new load to start.

Possible values of this parameter for conforming devices are:

0 - Device default (Guideline: text on VT220)
1 - Text
2 - Full Cell

4.1.7 Parameter (Pcmh): Character Cell Matrix Height

The seventh parameter defines the limit of the character cell matrix height in pixels of the font file being loaded. Note that legal values for this parameter are dependent on the values of the Font Set Size parameter and the Font Usage parameter, and may vary from device to device.

The legal combinations of these three parameters must be clearly defined in the product literature for conforming terminals. Illegal values of the Character Cell Matrix Height parameter cause the remainder of the load sequence to be ignored. Changes to this parameter since the last DECDLD sequence to this buffer will result in the erasure of the entire set, and cause a new load to start.

On video terminals, some values of the Character Cell Matrix Width (Pcmw) parameter also specify the Character Cell Matrix Height. If one of these Pcmw parameters is specified, the Character Cell Matrix Height (Pcmh) parameter is ignored.

The value of this parameter indicates the height of the Character Cell Matrix. For example, a value of 15 indicates a 15-pixel high character cell matrix. If a parameter value of 0 is specified, this indicates that the height of the character cell matrix is the device default, which is dependent on the implementing device.

Guideline:
Implementations may discard pixels beyond the height of the character cell specified by this parameter. Software can use the Character Matrix Width parameter to obtain device independent, predictable results.

4.1.8 Parameter (Pcss): Character Set Size

The eighth parameter controls the size of the associated character set (and modifies the interpretation of the Character Number (Pcn) parameter). The value of the parameter indicates the size of character set associated with the downloaded font according to:
0 - 94-character set (default)
1 - 96-character set

A parameter value of 1 permits the loading of the 2/0 and 7/15 characters in the font. And requires that the 96-character designating escape sequences (refer to the "Code Extension Layer" section) be used to designate the font.

The Final character { (7/11) indicates that this DCS string is the DECDLD control function. The data following it up to the String Terminator (ST) is from 2/0 to 7/14 and represents the actual font load command string.

4.2 COMMAND STRING FORMAT

The syntax of the DECDLD command string is as follows:

1. Dscs - (Select Character Set) The command string begins with a character set designation string, which defines the intermediate and final characters of the escape sequence which will subsequently be used to designate the loaded font as a character set.
2. Sxbp - The final character of the character set designation string is followed by a sequence of sixel bit pattern strings, each string containing the complete definition for one character of the font. Refer to subhead 10.4.2.2. Devices which support only 94-character sets should include only 94-sixel bit pattern strings in a single DECDLD command. Devices which support 96-character sets may include up to 96 sixel bit pattern strings in a single DLD command. Additional patterns will be ignored as necessary, but will not cause the command to be aborted.
3. ST - The String Terminator (ST, 9/12) control.

4.2.1 Character Set Designation String

The Dscs string is a set of zero, one, two, or three ANSI intermediates, followed by an ANSI final. (Dscs = I I I F). The Dscs defines the character set for the loaded font, and is used in the SCS (Select Character Set) escape sequence. Note that the same Dscs can be used to describe a 94-character or 96-character graphic set. The Pcss parameter indicates which initial intermediate character of the SCS sequence is to be used to designate the character set (see below). Refer to the DEC STD 070-3 and DEC STD 070-5 for a complete description of the escape sequence format, and particularly the SCS sequence.

The SCS sequence for designating the loaded font will be of the form:
ESC (Dscs - Load the set into G0 (94-character set)
ESC) Dscs - Load the set into G1 (94-character set)
ESC * Dscs - Load the set into G2 (94-character set)
ESC + Dscs - Load the set into G3 (94-character set)
ESC - Dscs - Load the set into G1 (96-character set)
ESC . Dscs - Load the set into G2 (96-character set)
ESC / Dscs - Load the set into G3 (96-character set)

Changes to the Dscs string since the last DECDLD sequence to this buffer will result in the erasure of the entire set, and cause a new load to start.

Examples of Dscs:

1. Dscs = sp @
   Defines the loaded font file to be a DRCS set. (This particular value is the recommended value for user defined sets for either 94-character or 96-character graphic character sets.)
2. Dscs = B
   Defines the loaded font file to be ASCII (94-character), or ISO Latin Alphabet Nr 2 supplemental (96-character), depending on the value of the Pcss parameter.
3. Dscs = & % C
   Defines the loaded font file to an as yet unregistered set.

4.2.2 Character Font Data

4.2.2.1 Summary

The remainder of the command data is a sequence of sixel bit pattern strings (Sxbp string), each defining the pattern for a single character in the font. Each Sxbp string is separated by a semicolon character (;, 3/11). Each individual Sxbp string consists of several sequences of sixel data (refer to subhead 10.4.2.2.3) which define the individual horizontal scans of the character pattern, each sixel data sequence being terminated by a slash character (/, 2/15).
Bit combinations 0/8 to 0/13 may be included in the command string, and have no affect on the interpretation of the data. They permit convenient formatting of command strings when being created by text editors since TAB, CR, and LF are included in this range. The following bit combinations are considered as error conditions when they occur within the character font data. Because existing implementations differ in how they handle these bit combinations, their processing is UNDEFINED by the architecture. It is not mandatory, but recommended that future implementations adopt the following conventions for these bit combinations.

- C0 controls 0/0 to 0/7 and 0/14 to 1/15 should be ignored except for ESC 1/11 which terminates the DCS and begins a new sequence. ESC followed immediately by \ 5/12 is the 7-bit representation of STRING TERMINATOR and ends the DECDLD normally. Other ESC sequences should cause the entire DECDLD command string to be ignored (aborted).
- 2/0 and 7/15 should be ignored.
- 2/1 to 3/14 with the exception of 2/15 and 3/11 should be ignored.
- C1 controls 8/0 to 9/15 should abort the DECDLD command string except for 9/12 STRING TERMINATOR which ends the DECDLD normally.
- 10/0 and 15/15 should be ignored.
- 10/1 to 15/14 should be interpreted as 2/1 to 7/14.

These characters should not be used by conforming software (except 2/15, 3/11, and STRING TERMINATOR).

4.2.2.2 Sixel Definition

A sixel is an encoded representation of six pixels. It is used to transfer pixel information in Device Control Strings. Sixel data consists of characters from ? (3/15) through (7/14), in which the character code represents a binary bit pattern. Each character represents a binary value plus 63 decimal. Thus 3/15 represents the binary value 000000, and 7/14 represents 111111. The bits are represented as a vertical, 1 by 6 pixel matrix ("the sixel"), with the least significant bit at the top. A reset bit (zero) specifies the font background, and a set bit (one) specifies the font foreground.

NOTE

Refer to *DEC STD 070-9 Sixel Graphics Extension* for a more complete description of the conversion algorithms. Note that the DECDLD command string uses the sixel data format, but does not use the control functions as defined in the Sixel Graphics Extension.

4.2.2.3 Sixel Bit Pattern

To load the logical pixel pattern for an individual character in a font (one Sxbp), a sequence of sixel characters is sent, with each sixel specifying one vertical column of six logical pixels. After each character the font pattern scan advances horizontally by one logical pixel. This sequence continues until either a scan terminator (the / character, 2/15) is received, a sixel bit pattern terminator (the ; character, 3/11) is received, or a STRING TERMINATOR (9/12) is received. If the number of characters in a horizontal scan is greater than the width of the loaded character (as defined in the Character Cell Matrix Size parameter of the introducer sequence), the characters in excess of the loaded character width are ignored.

The scan terminator character (/, 2/15) advances the font pattern scan to the next lower row of six vertical logical pixels. When the height of a loaded character (as defined in the Character Cell Matrix Size parameter of the introducer sequence) is not a multiple of six, the high order bits of the sixel values are ignored on the last row of the font pattern. If the number of horizontal scans is greater than the height of the loaded character (divided by six), the characters in excess of the loaded character height are ignored.

Any pixels of a loaded character that are not specified are set to zero. This might occur either as a result of a horizontal scan being terminated before the pattern has reached the full width of the loaded character, or as the result of the sixel bit pattern string being terminated before the full character matrix has been defined.

The following picture shows the mapping of sixels onto an 8 x 10 logical pixel character cell:

1	2	3	4	5	6	7	8
b0	b0	b0	b0	b0	b0	b0	b0
b1	b1	b1	b1	b1	b1	b1	b1
b2	b2	b2	b2	b2	b2	b2	b2
b3	b3	b3	b3	b3	b3	b3	b3
b4	b4	b4	b4	b4	b4	b4	b4
b5	b5	b5	b5	b5	b5	b5	b5
b0	b0	b0	b0	b0	b0	b0	b0
b1	b1	b1	b1	b1	b1	b1	b1
b2	b2	b2	b2	b2	b2	b2	b2
b3	b3	b3	b3	b3	b3	b3	b3
9	10	11	12	13	14	15	16

The rows of numbers on the top and bottom relate to the order in which sixel values are sent to the terminal.

The following picture shows the mapping of Sixels onto an 10 x 20 logical pixel full-cell character cell:

1	2	3	4	5	6	7	8	9	10
b0	b0	b0	b0	b0	b0	b0	b0	b0	b0
b1	b1	b1	b1	b1	b1	b1	b1	b1	b1
b2	b2	b2	b2	b2	b2	b2	b2	b2	b2
b3	b3	b3	b3	b3	b3	b3	b3	b3	b3
b4	b4	b4	b4	b4	b4	b4	b4	b4	b4
b5	b5	b5	b5	b5	b5	b5	b5	b5	b5
11	12	13	14	15	16	17	18	19	20
b0	b0	b0	b0	b0	b0	b0	b0	b0	b0
b1	b1	b1	b1	b1	b1	b1	b1	b1	b1
b2	b2	b2	b2	b2	b2	b2	b2	b2	b2
b3	b3	b3	b3	b3	b3	b3	b3	b3	b3
b4	b4	b4	b4	b4	b4	b4	b4	b4	b4
b5	b5	b5	b5	b5	b5	b5	b5	b5	b5
21	22	23	24	25	26	27	28	29	30
b0	b0	b0	b0	b0	b0	b0	b0	b0	b0
b1	b1	b1	b1	b1	b1	b1	b1	b1	b1
b2	b2	b2	b2	b2	b2	b2	b2	b2	b2
b3	b3	b3	b3	b3	b3	b3	b3	b3	b3
b4	b4	b4	b4	b4	b4	b4	b4	b4	b4
b5	b5	b5	b5	b5	b5	b5	b5	b5	b5
31	32	33	34	35	36	37	38	39	40
b0	b0	b0	b0	b0	b0	b0	b0	b0	b0
b1	b1	b1	b1	b1	b1	b1	b1	b1	b1

The rows of numbers on the top and bottom relate to the order in which sixel values are sent to the terminal.

4.3 ALGORITHM

State Affected:

```
            FONT: ARRAY [1..MAX_NUM_FONTS] OF FONT_BUFFER_TYPE;
            CELL_NUMBER: CHARACTER_VALUE_TYPE;
            FONT_NUMBER: 1..MAX_NUM_FONTS;
            FIRST_CHARACTER: CHARACTER_VALUE_TYPE;
            ERASE_CONTROL: (ERASE_THIS_FONT,NO_ERASE,ERASE_ALL_FONTS);
            CELL_WIDTH: CELL_WIDTH_TYPE;
            CELL_HEIGHT: CELL_HEIGHT_TYPE;
            SET_SIZE: (EIGHTY,ONE_THIRTY_TWO);
            CELL_USAGE: (TEXT,FULL_CELL);
|           CHARACTER_SET_SIZE: (94,96);
            CHARACTER_SET_DESIGNATOR:
              ARRAY [1..MAX_LENGTH_DESIGNATOR] OF
              DESIGNATOR_CHARACTER_TYPE;
```

Algorithm:

```
   CONST    MAX_NUM_FONTS = 1;

|           MIN_CHARACTER_VALUE = 0;
|           MAX_CHARACTER_VALUE = 95;
|           (* Devices which are not required to support
|              96-character sets may set these values to 1 and 94 *)

            MAX_CELL_WIDTH = 8;
            MAX_CELL_HEIGHT = 10;
|           (* These values may vary from device to device *)

            MAX_LENGTH_DESIGNATOR = 3;

            ON = TRUE;
            OFF = FALSE;

   TYPE     CELL_WIDTH_TYPE = 1..MAX_CELL_WIDTH;
            CELL_HEIGHT_TYPE = 1..MAX_CELL_HEIGHT;

            CHARACTER_CELL_TYPE =
              ARRAY [CELL_WIDTH_TYPE,CELL_HEIGHT_TYPE] OF BOOLEAN;

            CHARACTER_VALUE_TYPE =
              MIN_CHARACTER_VALUE..MAX_CHARACTER_VALUE;

            FONT_BUFFER_TYPE =
              ARRAY [CHARACTER_VALUE_TYPE] OF CHARACTER_CELL_TYPE;

   VAR      FONT: ARRAY [1..MAX_NUM_FONTS] OF FONT_BUFFER_TYPE;
            CELL_NUMBER: CHARACTER_VALUE_TYPE;
            FONT_NUMBER: 1..MAX_NUM_FONTS;
            FIRST_CHARACTER: CHARACTER_VALUE_TYPE;
            DRCS_ERASE_CONTROL:
   (ERASE_THIS_FONT,NO_ERASE,ERASE_ALL_FONTS);
            CELL_WIDTH: CELL_WIDTH_TYPE;
            CELL_HEIGHT: CELL_HEIGHT_TYPE;
            SET_SIZE: (EIGHTY,ONE_THIRTY_TWO);
            CELL_USAGE: (TEXT,FULL_CELL);
|           CHARACTER_SET_SIZE: (94,96);
            VALID_DRCS_SYNTAX: BOOLEAN;
            DRCS_STRING_TERMINATED: BOOLEAN;
            H,V,N: INTEGER;

   PROCEDURE GET_NEXT_EVENT (VAR EVENT: EVENT_BUFFER_TYPE); EXTERN;
   PROCEDURE GET_DRCS_PARAMETERS (EVENT: EVENT_BUFFER_TYPE); FORWARD;
|  PROCEDURE GET_CHARACTER_SET_DESIGNATOR; FORWARD;
   PROCEDURE FILL_CELLS; FORWARD;
```

```
PROCEDURE EXECUTE_FONT_LOAD (EVENT: EVENT_BUFFER_TYPE);
(*
  This routine processes DECDLD font load command strings.
  The input to this routine is the parsed DCS introducer
  sequence. Successive calls are made to the ANSI parser
  (GET_NEXT_EVENT) to input a character at a time for
  processing. The routine will not exit until a String
  Terminator (9/12) is received.
*)
BEGIN
(* first, process the DCS introducer sequence parameters *)
GET_DRCS_PARAMETERS (EVENT);
IF VALID_DRCS_SYNTAX THEN
  BEGIN
  DRCS_STRING_TERMINATED:= FALSE;
  (* if the sequence is valid, get the character set designator *)
  GET_CHARACTER_SET_DESIGNATOR;
  IF VALID_DRCS_SYNTAX THEN
    BEGIN
    (* erase according to the erase parameter *)
    CASE DRCS_ERASE_CONTROL OF
      ERASE_THIS_FONT:
        FOR CELL_NUMBER:=
        MIN_CHARACTER_VALUE TO MAX_CHARACTER_VALUE DO
          FOR H:= 1 TO MAX_CELL_WIDTH DO
            FOR V:= 1 TO MAX_CELL_HEIGHT DO
              FONT[FONT_NUMBER,CELL_NUMBER,H,V]:= OFF;
      ERASE_ALL_FONTS:
        FOR N:= 1 TO MAX_NUM_FONTS DO
          FOR CELL_NUMBER:=
          MIN_CHARACTER_VALUE TO MAX_CHARACTER_VALUE DO
            FOR H:= 1 TO MAX_CELL_WIDTH DO
              FOR V:= 1 TO MAX_CELL_HEIGHT DO
                FONT[FONT_NUMBER,CELL_NUMBER,H,V]:= OFF;
      OTHERWISE IGNORE;
      END;
    (* then process the command string *)
    FILL_CELLS;
    (* on overflow, keep going until String Terminator *)
    WHILE NOT DRCS_STRING_TERMINATED DO
      BEGIN
      GET_NEXT_EVENT (EVENT);
      IF (EVENT.EVENT_CODE = CONTROL_CODE)
      AND (EVENT.CODE_VALUE = 156) THEN
        DRCS_STRING_TERMINATED:= TRUE;
      END;
    END;
  END;
END;
```

```
PROCEDURE GET_DRCS_PARAMETERS;
(*
  This routine gets the DCS introducer sequence parameters
  from the event buffer and sets the appropriate state
  prior to loading the font.
*)
BEGIN
VALID_DRCS_SYNTAX:= TRUE;
(* the first parameter is the font number to be loaded *)
IF (EVENT.PARAMETERS[1] = 0) OR (EVENT.PARAMETERS[1] = 1) THEN
  FONT_NUMBER:= 1
ELSE VALID_DRCS_SYNTAX:= FALSE;
(* the second parameter is the first character to be loaded *)
FIRST_CHARACTER:= EVENT.PARAMETERS[2];
IF (FIRST_CHARACTER = 0) OR (FIRST_CHARACTER > 95) THEN
  FIRST_CHARACTER:= 1;
(* the third parameter is the erase control specifier *)
CASE EVENT.PARAMETERS[3] OF
  0: DRCS_ERASE_CONTROL:= ERASE_THIS_FONT;
  1: DRCS_ERASE_CONTROL:= NO_ERASE;
  2: DRCS_ERASE_CONTROL:= ERASE_ALL_FONTS;
  OTHERWISE VALID_DRCS_SYNTAX:= FALSE;
  END;
(* the fourth parameter is the cell width or size *)
CELL_HEIGHT:= MAX_CELL_HEIGHT;      (* assume no cell height
                                       parameter at start    *)
CASE EVENT.PARAMETERS[4] OF
  0: CELL_WIDTH:= MAX_CELL_WIDTH;   (* 0 is device default *)
  1: VALID_DRCS_SYNTAX:= FALSE;     (* 1 is illegal*)
  2: CELL_WIDTH:= 5;
  3: CELL_WIDTH:= 6;
  4: CELL_WIDTH:= 7;
  OTHERWISE  CELL_WIDTH:= EVENT.PARAMETERS[4];
  END;
IF (CELL_WIDTH > MAX_CELL_WIDTH)
 THEN CELL_WIDTH:= MAX_CELL_WIDTH;
(* the fifth parameter is the set size *)
CASE EVENT.PARAMETERS[5] OF
  0,1: SET_SIZE:= EIGHTY;
  2: SET_SIZE:= ONE_THIRTY_TWO;
  OTHERWISE VALID_DRCS_SYNTAX:= FALSE;
  END;
(* the sixth parameter is the font usage *)
CASE EVENT.PARAMETERS[6] OF
  0,1: CELL_USAGE:= TEXT;
  2: CELL_USAGE:= FULL_CELL;
  OTHERWISE VALID_DRCS_SYNTAX:= FALSE;
  END;
(* the seventh parameter (if present) is the cell height *)
CASE EVENT.PARAMETERS[7] OF
  0: CELL_HEIGHT:= MAX_CELL_HEIGHT;  (* device dependant default
*)
  OTHERWISE CELL_HEIGHT:= EVENT.PARAMETERS[7];
  END;
IF (CELL_HEIGHT > MAX_CELL_HEIGHT)
 THEN CELL_HEIGHT:= MAX_CELL_HEIGHT;
(* the eighth parameter is the character set size *)
CASE EVENT.PARAMETERS[8] OF
  0: CHARACTER_SET_SIZE:= 94;
  1: CHARACTER_SET_SIZE:= 96;
  OTHERWISE VALID_DRCS_SYNTAX:= FALSE;
  END;
END;
```

```
PROCEDURE GET_CHARACTER_SET_DESIGNATOR;
(*
  This routine parses the character set designating escape
  sequence to be associated with the loaded font. This sequence
  must immediately follow the DCS introducer sequence.

  Refer to DEC STD 070-5 for a description of the character set
  table and the subsequent designation procedure.
*)
BEGIN
(* accept characters until final or String Terminator *)
N:= 1;
REPEAT
  GET_NEXT_EVENT (EVENT);
  CASE EVENT.EVENT_CODE OF
    GRAPHIC_CODE:      (* put legal characters in the buffer *)
      BEGIN
      IF (N=1) AND (EVENT.CODE_VALUE >= 32)
      AND (EVENT.CODE_VALUE <= 47) THEN
        CHARACTER_SET_TABLE[4].FIRST_INTERMEDIATE:=
        EVENT.CODE_VALUE;
      IF (N=2) AND (EVENT.CODE_VALUE >= 32)
      AND (EVENT.CODE_VALUE <= 47) THEN
        CHARACTER_SET_TABLE[4].SECOND_INTERMEDIATE:=
        EVENT.CODE_VALUE;
      IF (EVENT.CODE_VALUE >= 48)
      AND (EVENT.CODE_VALUE <= 126) THEN
        CHARACTER_SET_TABLE[4].FINAL:= EVENT.CODE_VALUE;
      IF (EVENT.CODE_VALUE >= 32)
      AND (EVENT.CODE_VALE <= 126) THEN N:= N+1;
      END;
    CONTROL_CODE:      (* terminate on String Terminator, 9/12 *)
      IF EVENT.CODE_VALUE = 156 THEN DRCS_STRING_TERMINATED:= TRUE
      ELSE IGNORE;     (* ignore other control characters *)
    OTHERWISE IGNORE; (* ignore escape and control sequences *)
    END;
UNTIL ((EVENT.EVENT_CODE = GRAPHIC_CODE)
AND ((EVENT.CODE_VALUE >= 64) AND (EVENT.CODE_VALUE <= 126)))
OR DRCS_STRING_TERMINATED;
END;
```

```
PROCEDURE FILL_CELLS;
(*
  This routine processes the sixel command string and loads
  the character cells in succession, starting with the first
  character specified in the introducer sequence.
*)
VAR     NEXT_CELL: BOOLEAN;
BEGIN
(* start with the first character as specified *)
CELL_NUMBER:= FIRST_CHARACTER;
NEXT_CELL:= TRUE;
REPEAT
  (* clear the cell if not already cleared *)
  IF NEXT_CELL THEN
    BEGIN
    IF DRCS_ERASE_CONTROL = NO_ERASE THEN
      FOR H:= 1 TO MAX_CELL_WIDTH DO
        FOR V:= 1 TO MAX_CELL_HEIGHT DO
          FONT[FONT_NUMBER,CELL_NUMBER,H,V]:= OFF;
    H:= 1;
    V:= 1;
    NEXT_CELL:= FALSE;
    END;
  (* now, fill it *)
  GET_NEXT_EVENT (EVENT);
  CASE EVENT.EVENT_CODE OF
    GRAPHIC_CODE:
      BEGIN
      (* convert sixels into the font memory *)
      IF (EVENT.CODE_VALUE >= 63)
      AND (EVENT.CODE_VALUE <= 126) THEN
        BEGIN
        EVENT.CODE_VALUE:= EVENT.CODE_VALUE - 63;
        IF H <= CELL_WIDTH THEN
          BEGIN
          FOR N:= 1 TO 6 DO
            BEGIN
            IF V+N-1 <= CELL_HEIGHT THEN
              BEGIN
              IF ODD(EVENT.CODE_VALUE) THEN
                FONT[FONT_NUMBER,CELL_NUMBER,H,V+N-1]:= ON;
              (* shift in one bit at a time *)
              EVENT.CODE_VALUE:= EVENT.CODE_VALUE DIV 2;
              END
            ELSE IGNORE;
            END;
          H:= H+1;
          END
        ELSE IGNORE;
        END
      ELSE
        BEGIN
        CASE EVENT.CODE_VALUE OF
          (* / terminates a horizontal scan *)
          47: BEGIN H:= 1; V:= V+6; END;
          59: (* ; terminates a cell, go to the next *)
            BEGIN
            NEXT_CELL:= TRUE;
            CELL_NUMBER:= CELL_NUMBER+1;
            H:= 1; V:= 1;
            END;
          OTHERWISE IGNORE;
          END;
        END;
      END;
```

```
    CONTROL_CODE:      (* terminate on String Terminator, 9/12 *)
      IF EVENT.CODE_VALUE = 156 THEN DRCS_STRING_TERMINATED:= TRUE
      ELSE IGNORE;     (* ignore other control characters *)
    OTHERWISE IGNORE; (* ignore escape and control sequences *)
    END;
UNTIL (CELL_NUMBER > MAX_CHARACTER_VALUE)
OR (DRCS_STRING_TERMINATED);
END;
```

5 DRCS FONT LOADING EXAMPLES

The following are examples of DECDLD font loading sequences:

5.1 EXAMPLE ONE

```
DCS 1 ; 1 ; 0 ; 0 ; 0 ; 0 { sp @
~~~~~~~~/~~~~~~~~;
????????/????????;
~@@@@@~?/~WWWWW~?;
TTTTTTTT/TTTTTTTT;
ST
```

Notes:

1. This load is into the first DRCS font buffer.
2. The font is an 80 column text font with a character cell matrix size of 7 x 10.
3. "sp" represents the Space character (2/0).
4. The loaded character set can be designated using the following Select Character Set (SCS) sequences:
   ESC (sp @
   ESC) sp @
   ESC * sp @
   ESC + sp @
5. Carriage return is ignored in the sequence and can be used to facilitate formatting and editing of the command string.
6. The first character loaded is a solid rectangle.
7. The second character loaded is blank.
8. The third character loaded is a rectangular box.
9. The fourth character loaded is a set of horizontal stripes.

5.2 EXAMPLE TWO

```
DCS 5 ; 65 ; 1 ; 2 ; 2 ; 1 { B
???/??????????/?????;
~~~~~/~~~~~;
?????/~~~~~~~~~~~~~~~/??????????????/~?~?~?~?~?~?~?~?~;
??~~??/??~~?
ST
```

Notes:

1. This load is into the fifth DRCS font buffer. Since only one buffer is required for Level 2 conformance, the entire load sequence may be ignored.
2. The load will start with the 65th character in the set, which would be the character 6/1 (ASCII 'a').
3. Only the characters actually loaded will be erased. Thus, this load could be used to redefine only the lower case characters.
4. The font is a 132 column text font with a character cell matrix size of 5 x 10.

5. B is the designator for the ASCII character set - therefore, this font replaces ASCII, and may be designated using the following Select Character Set (SCS) sequences:
   ESC (B
   ESC) B
   ESC * B
   ESC + B

6. The first character loaded is blank.
7. The second character loaded is solid.
8. The third character loaded is top 6/10 blank, bottom 4/10 solid.
9. The fourth character loaded is a thick vertical bar.

5.3 EXAMPLE THREE

```
DCS 1 ; 65 ; 1 ; 4 ; 2 ; 2 ; 16 ; 1 { B
???/??????????/?????;
~~~~~~~~~~~/~~~~~~~~~~/~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
?????/{{{{{/~~~~~/?????;
?~~?/?~~?/?~~?
ST
```

Notes:

1. The font is a 132-column, full-cell font with character matrix size of 4 x 16
2. The first character loaded corresponds to the 6/0 position in the code table.
3. The first character loaded is blank
4. The second character loaded is solid
5. The third character loaded has the top half blank, and the bottom half solid
6. The fourth character is a thick vertical bar
7. The excess Sixel information (outside of the 4 x 16 cell) is ignored
8. "B" could be confused with the ASCII designator, however, since Pcss is 1, making this a 96-character soft character set, this font does not replace the ASCII font. (It is, however, the registered designator for the supplemental character set of ISO Latin Alphabet Nr 2, and would replace that set, if it were present in the device.)
9. The character set can be designated using the 96-character SCS sequence. The Dscs is "B". The SCS sequences are:
   ESC - B
   ESC . B
   ESC / B

6 CHANGE HISTORY

6.1 Revision 0.0 To AX10

1. Added Introduction, Terminology, and Functional Description sections.
2. Revised the DECDLD command format.
3. Restricted the legal values for the Font Number parameter.
4. Restricted the legal values for the First Character parameter.
5. Added a parameter value to erase all font definitions (User Preference Feature).
6. Added parameters for Character Cell Matrix Size, Font Set Size, and Font Usage.
7. Extensively revised the section on Command String Format.
8. Added the DECDLD algorithm.
9. Added examples for DECDLD.
10. Added references to the Character Cell Display section for character set tables of designating escape sequences.

6.2 Rev AX10 to AX11

1. Added statement that starting to load at location greater than 94 will be ignored in level 2.

6.3 Rev AX11 to AX12

1. Added support and Character Set Size parameter for 96-character graphic charcter sets for Level 2 and 3 DRCS.
2. Modified meaning of Character Matrix parameter to mean Character Matrix Width parameter as a modification to Level 2 DRCS.
3. Added Character Matrix Height parameter as a modification to Level 2 DRCS.
4. Added example containing new parameters.
5. Extended behavior of Font Number parameter when parameter is omitted or zero.
6. Clarified behavior of DRCS which replace standard character sets when the DRCS is erased.
7. Updated algorithm to reflect new parameters and 96-character graphic character set support.

6.4 Rev AX12 to AX13

1. Changed title to "Video & Printer Systems Reference Manual".
2. Clarified use of terms "font", "font file", and "character set" to be more consistent with printer terminology. "Font file" refers to a sequence of bit patterns that define the graphic symbols of a coded character set. "Fonts" are styles (an instance of a type family).

3. Added minimum requirements for Hardcopy Display service class (three 96 character buffers).
4. Clarified that DRCS can be used as the User Preference Supplemental Set.
5. Clarifed effect of DECDLD on characters already on the page in printers (do not change).
6. Specified behavior of DECSTR, RIS, and DECSCL with respect to DRCS buffers for both video and hardcopy devices.
7. Clarified differences in behavior of DECDLD parameters for video and hardcopy devices (Pfn, Pcmw, Pss, Pcmh). Printers have three buffers minimum. Printers never deduce Pcmh from Pcmw. Pss (Font set size) is for video terminals only. Extended list of possible values for Pss to reflect VT420 implementation.
8. Specified Dscs may include up to 3 intermediates in addition to the first SCS intermediate which specifies the G-set.
9. Included error handling recommendations for unexpected bit combinations. ESC and C1 controls abort DECDLD (except ST and DCS). 10/1 to 15/14 should be interpreted as 2/1 to 7/14. This reflects latest consensus among hardcopy and video implementations.
10. Clarified that ST may terminate an individual character load.

APPENDIX A REFERENCED DOCUMENTS

A.1 EL-Class Digital Documents

EL-Class Number	Document Title
EL-00070-01	*DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria*
EL-00070-03	*DEC STD 070-3 Video Systems Standard - Code Extension Layer*
EL-00070-05	*DEC STD 070-5 Video Systems Reference Manual - Character Cell Display*
EL-00070-09	*DEC STD 070-9 Video and Printer Systems Reference Manual - SIXEL Graphics Extension*

Use VTX SMC to order copies of EL-class documents from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

INDEX

```
+-------------------------------------------------------------------+
|                        READER COMMENTS                            |
|                                                                   |
| Your comments and suggestions will help Standards and Methods     |
| Control improve their services and documents.                     |
+-------------------------------------------------------------------+
```

Did you request this document? _____ If so, did it arrive within a satisfactory period of time? _____ Please comment.

What are your impressions of this document? Consider format, organization, completeness, readability, and illustrations.

FOLD ON THIS LINE

Did you find technical or clerical errors in this document? If so, please specify the page number(s) and the error(s).

Are the instructions for the update package clear? _____
Was an index available? _____ If not, is one needed? _____
Do you have other suggestions for improving this document?

The following information is optional:

Name _____ Mailstop _____

Department _____ Node _____

Send your comments to JOKUR::PROJECTS, or fold, staple, and send this page through interoffice mail to:

```
+------------------------------+
|       READERS' COMMENTS      |
| STANDARDS AND METHODS CONTROL|
|           CTS1-2/D4          |
+------------------------------+
```

VIDEO SYSTEMS REFERENCE MANUAL

User Defined Keys Extension

Document Identifier: A-DS-EL00070-011-0 Rev. AX10, 15-May-83

ABSTRACT: This section describes the interface to load and invoke User Definable Keys (UDK), which form an extension to the Level 2 Character Cell Display service class.

APPLICABILITY: SARA Formal Cross-Component Standard. Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria".

STATUS: FOR REVIEW ONLY

+---+
| This document has been placed in the SARA "Formal Cross-Component Standard" category. The material contained within this document is assumed to define mandatory standards unless it is clearly marked as (a) not mandatory or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements. The procedures for modifying or evolving this standard are contained within the contents of this document. |
+---+

+---------------------------------------+
| FOR INTERNAL USE ONLY |
+---------------------------------------+

TITLE: VIDEO SYSTEMS - UDK EXTENSION

DOCUMENT IDENTIFIER: A-DS-EL00070-011-0 Rev. AX10, 15-May-1983

REVISION HISTORY: Original Draft 25-Dec-1982

FILES: User Documentation EL070S11.mem

Internal Documentation EL070S11.rno
EL070S11.rnt
EL070S11.rnx

Document Management Group: Terminal Interface Architecture
Responsible Department: Video Architecture Group
Responsible Individual: Peter Sichel

ACCEPTANCE: This document has been approved by the Manager of the Video Architecture Group based on a comprehensive review of its individual sections by the members of the SARA component groups working Terminal Interface Architecture issues. The list of individuals on the review and approval list are on file in Standards and Methods Control.

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, RANI::VIDARCH

Copies of this document can be ordered from:

Standards and Methods Control
APO1/F7, DTN 289-1414, JOKUR::SIMONETTI

CONTENTS

11.1 INTRODUCTION

User Defined Keys are a Level 2 extension, which provide the capability of loading into the terminal key definition strings which subsequently be transmitted using a single key stroke. The format for loading key definition strings is intended to be extensible to future devices which have more complex keyboards, larger memories, or other forms of input processing. Conforming Level 2 devices which implement this extension, however, are required to provide only fifteen programmable keys, with a total buffer space of at least 256 characters.

It should be noted that conformance to this standard does not require the ability to retain the key definitions in non-volatile memory. That is, the loaded definitions may be lost if the terminal is powered off. On the other hand, non-volatile storage and restoration of the key definition strings is not precluded. Furthermore, conformance does not preclude a device from providing the ability to interact locally with the key definitions (input, edit, or delete), provided that it is done in keeping with the intent of this standard.

11.2 FUNCTIONAL DESCRIPTION

11.2.1 Programmable Keys

The keys which may be programmed in Level 2 devices are all of the keys in the Application Function Key Row, when used in combination with the Shift key (see the section "Keyboard Processing" for a complete description of the keyboard). The keys may only be programmed, and are only operational, when the terminal is in Level 2 operation (as selected by the control function Select Conformance Level (DECSCL); see the section "Terminal Management" for a complete description of the DECSCL function). When the terminal is in Level 1 operation (or in VT52 mode), the DECUDK control function is ignored and the keys are "dead" (i.e., they do not transmit and do not keyclick).

11.2.2 Default Definitions

Programmable keys which have not been defined, or have had their definitions erased, do not transmit and therefore do not keyclick. Programmable keys which have been defined do keyclick, and are also subject to all of the requirements of the lower level transmission layers, such as XON/XOFF flow control, rate limiting, and C1 Control Code translation.

11.2.3 Key Definitions

The definition for each key consists of a variable length character string of 0 to 255 7-bit or 8-bit characters. Each character in the string is a code in the range 0/0 through 15/15. Thus, any character in the code table may be included within a key definition.

Buffer space for the key definitions is supplied on a first come first serve basis. Once the supplied buffer space is used up, no more keys may be defined unless space is freed up. Space may be freed up either by redefining previously defined keys using shorter strings than before, clearing key definitions using the DECUDK control function, or clearing the entire definition set (through a terminal power up reset).

11.2.4 Lock Control

The terminal uses a special lock to arbitrate the programming of User Defined Keys. This lock can be turned on or off through Setup mode. It may also be turned on (but not off) from the host using the DECUDK control function. The lock acts globally over all programmable keys.

The state of the Lock Control may be determined from the host using the Device Status Report control function.

11.3 CONTROL FUNCTIONS

REPORT UDK STATUS DSR

Levels: 2

Purpose: Report the status of the UDK Lock Control.

Request Format:	CSI	?	25		n
	9/11	3/15	3/2	3/5	6/14
Report Format:	CSI	?	Ps		n
	9/11	3/15	Ps		6/14

Description: The DSR control function (Device Status Report) may be used to request the current status of the UDK Lock Control. (See the section "Terminal Management" for a complete description of Device Status Report.) The terminal will respond to a UDK status request (CSI ?, 25 n) by transmitting a Device Status Report control sequence containing one of the following selective parameters (preceded by the private parameter value ?, 3/15):

Selective Parameter	Meaning
20	The User Defined Keys are not locked, and all keys may be defined, cleared, or redefined from the host.
21	The User Defined Keys are locked, and may not be defined, cleared, or redefined from the host.
23	No User Defined Keys are present. The device does not support this extension.

Notes:

1. A report will be given even if the UDK extension is not supported. Thus, this request is an optional means of determining if the extension is present (the other means being the Primary Device Attributes request, as described in the section "Terminal Management").

State Affected: None

Algorithm:

```
PROCEDURE REPORT_UDK_STATUS;
BEGIN
IF LEVEL_2_EXTENSIONS[UDK] THEN
  BEGIN
  CASE UDK_LOCK_CONTROL OF
    UDK_LOCK_OFF: WRITE (HOST_PORT,'<CSI>?20n');
    UDK_LOCK_ON: WRITE (HOST_PORT,'<CSI>?21n');
    END;
  END
ELSE WRITE (HOST_PORT,'<CSI>?23n');
END;
```

Known Deviations: None

USER DEFINED KEYS DECUDK

Levels: 2X

Purpose: Down line load User Defined Key sequences.

Format: The DECUDK control is a Device Control String consisting of an introducer sequence, followed by a command string. The format of the introducer sequence is:

DCS Pe ; Pl |

9/0 Pe ; Pl 7/12

The introducer sequence is terminated by the final character (7/12).

The command string format is:

Kyl / Stl ; Ky2 / St2 ; ... ; Kyn / Stn ST

The command string is terminated by the String Terminator character 9/13 (in 7-bits, ESC \, or 1/11 5/12). See the section "Code Extension Layer" for a complete description of the Device Control String format.

Description: DECUDK control down line loads one or more key definitions into the User Defined Keys. This sequence is ignored if the UDKs have been locked, either through a previous DECUDK sequence or as a local terminal function. This sequence executes only when the terminal is in Level 2 operation.

Notes:

1. Software should use the UDK function to reclaim key definition space in the device. This can be done by clearing keys without locking them. Once the keys have been cleared the UDK function can be used to redefine the keys and lock them.

2. Host software should not generally leave the user defined keys unlocked. This may cause a breach of security to the terminal user.

3. The host must keep track of space available for key definitions. No error indication is given if the available buffer space is filled up.

4. If a key is redefined the old sequence is lost. This may free up space if the new sequence is shorter than the previous definition.

11.3.1 Introducer Sequence Format

The following selective parameter values in the introducer sequence may effect the interpretation of the command string. All parameters have the same syntax as a control sequence and may be omitted, in which case the default values are used.

1. Parameter (Pe): Erase Control

   The first parameter controls which key definitions are to be cleared before loading takes place, according to:

   0 - Clear all UDKs before loading new values (default)
   1 - Load new UDK values, clear old only when redefined

   By using Pe=1, it is possible to redefine some keys without reloading them all.

   Note that if the clear parameter is set to 1 (load new, but don't clear old) it is possible that a key load might fail, due to lack of buffer room, even though the final total for all keys would have been 256 bytes or less. This is because all keys are cleared and loaded sequentially. Sequential loading without first clearing the key definitions could result in intermediate storage requirements greater than the maximum provided, even though the final requirement would be less than or equal to the maximum. For example: If F6 contained 120 bytes, F7 contained 110 bytes, and F8 contained 20 bytes, loading F8 with 40 bytes, F6 with 1 byte and F7 with 1 byte works if all UDKs are cleared first, but not if the keys are cleared as they are sequentially redefined. (When the user attempts to load F8 with 40 bytes, the load will fail, because only 26 bytes are free at that time (256-120-110=26)).

2. Parameter (Pl): Lock Control

   The second parameter permits the keys to be locked against subsequent deletion or re-definition. Locking the keys does not effect the definitions included in the current DECUDK sequence (unless the keys were previously locked), but causes all subsequent DECUDK sequences to be ignored (until the lock is cleared by the terminal user under local control). The following selective parameters may be specified:

   0 - Lock the UDKs against future redefinition (default)
   1 - Don't lock the UDKs against future redefinition

Note that Pl=1 does not unlock the UDKs, it just does not lock them.

The Final character | (7/12) indicates that this DCS string is the DECUDK control function. The data following it up to the String Terminator (ST) is from 2/0 through 7/14 and represents the actual key loading command.

11.3.2 Command String Format

The syntax of the DECUDK command string is as follows:

Key definition strings, Kyn/Stn, are included in the data following the Final character and before the String Terminator (ST). Each Key Definition String consists of a Key Selector Number, Kyn, and a String Parameter, Stn, separated by a slash "/". Multiple Key definition strings are separated by semi-colon ";".

Bit combinations 0/8 to 0/13 may be included in the command string, and have no effect on the interpretation of the data. They permit convenient formatting of command strings when being created by text editors since TAB, CR, and LF are included in this range. The following bit combinations are considered as error conditions when they occur within the character font data. Their processing is UNDEFINED by the architecture. It is recommended to hardware implementors that they be ignored for future compatibility.

- 0/1 through 0/7
- 0/14 through 2/14 (with the exception of ESC, 1/11 when it is followed immediately by \, 5/12, to form the two character ESC Fe sequence String Terminator)
- 3/10
- 3/12 through 4/0
- 4/7 through 6/0
- 6/7 through 15/15 (with the exception of ST, 9/12)

These characters should not be used by conforming software.

11.3.2.1 Key Selection Codes

Key selection codes are specified as string of characters in the range 3/0 through 3/9 ("0" through "9"), indicating a single decimal value. The values which may be specified in Level 2 are the parameter values of the control sequences normally transmitted by the programmable keys when they are not pressed in combination with the Shift key (CSI ? Pn ~). (See the section "Keyboard Processing" for a complete description of the codes transmitted by the Application Function Keys.)

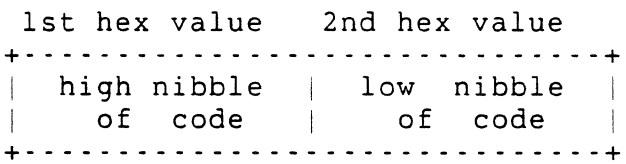
The following table indicates the relationship between the specified value and actual keys on the keyboard:

Key Selection Code	Related Key
17	F6
18	F7
19	F8
20	F9
21	F10
23	F11
24	F12
25	F13
26	F14
28	Help Key
29	Do Key
31	F17
32	F18
33	F19
34	F20

Key selectors may be in any order and may be specified multiple times within the string. If multiple definitions of a key occur, the last definition is preserved. If no key definition is specified for a selected key, that UDK will be cleared of it's definition.

11.3.2.2 String Parameters

The string parameters, Stn, are the encoded contents of the UDKs. They consist of hex pairs in the range 3/0 - 3/9 ("0" through "9"), 4/1 - 4/6 ("A" through "F"), and 6/1 - 6/6 ("a" through "f"). When these hex values are combined they represent an 8-bit quantity. This allows any of the 256 character codes to be used in the UDK sequence. The 8-bit code is loaded high byte first, then low byte, as shown below:



11.3.3 Error Conditions

The following error conditions may occur, and have the specified recovery:

1. Partial Hex Code - If an odd number of hex values is received in any of the string definitions, the current key definition and the entire remainder of the load sequence will be ignored. The Lock Control will also be set (UDK Lock On).

2. Key Code Out of Range - If a key selection code is received which does not represent a programmable key in the device, the entire definition for that key will be processed and ignored. The remainder of the string will be processed normally.

3. Null Key Definition - If two key definition separators (semi-colons) occur together, with no properly terminated key selection code intervening, the characters between the semi-colons will be ignored. The remainder of the string will be processed normally.

11.3.4 Algorithm

NOTE: For the sake of clarity, the following algorithm was coded using an excess of UDK buffer memory and working buffers. It should be recognized that the memory management schemes which will probably be required in many implementations will make this procedure much more complex.

State Affected:

```
UDK: UDK_TYPE;
TOTAL_COUNT: 0..MAX_SIZE_UDK_BUFFER;
UDK_LOCK_CONTROL: (UDK_LOCK_OFF,UDK_LOCK_ON);
UDK_BUFFER_OVERFLOW: BOOLEAN;
```

Algorithm:

```
CONST     MIN_UDK_VALUE = 17;
          MAX_UDK_VALUE = 35;
          MAX_SIZE_UDK_BUFFER = 256;

TYPE      CODE_TYPE = 0..255;

          KEY_DEFINITION_TYPE = RECORD
            CODE_BUFFER:
              ARRAY [1..MAX_SIZE_UDK_BUFFER] OF CODE_TYPE;
            CODE_COUNT: 0..MAX_SIZE_UDK_BUFFER;
            END;

          UDK_RANGE = MIN_UDK_VALUE..MAX_UDK_VALUE;

          LEGAL_UDK_VALUES_TYPE = SET OF UDK_RANGE;

          UDK_TYPE =
            ARRAY [UDK_RANGE] OF KEY_DEFINITION_TYPE;
```

```
VAR       UDK: UDK_TYPE;
          TOTAL_CODE_COUNT: 0..MAX_SIZE_UDK_BUFFER;
          UDK_LOCK_CONTROL: (UDK_LOCK_OFF,UDK_LOCK_ON);
          LEGAL_UDK_VALUES: LEGAL_UDK_VALUES_TYPE;
          LOCK_SET: BOOLEAN;
          UDK_ERASE_CONTROL: (NO_ERASE_UDK,ERASE_ALL_UDK);
          VALID_UDK_SYNTAX: BOOLEAN;
          UDK_STRING_TERMINATED: BOOLEAN;
          KEY_CODE: INTEGER;
          KEY_CODE_TERMINATED: BOOLEAN;
          HEX_CODE_VALUES: SET OF CODE_TYPE;
          WORK_BUFFER:
            ARRAY [1..MAX_SIZE_UDK_BUFFER] OF CODE_TYPE;
          CODE_PAIR_TERMINATED: BOOLEAN;
          DEFINITION_TERMINATED: BOOLEAN;
          UDK_BUFFER_OVERFLOW: BOOLEAN;
          I,N: INTEGER;

PROCEDURE GET_NEXT_EVENT (VAR EVENT: EVENT_BUFFER_TYPE); EXTERN;
PROCEDURE GET_UDK_PARAMETERS (EVENT: EVENT_BUFFER_TYPE); FORWARD;
PROCEDURE GET_KEY_CODE; FORWARD;
PROCEDURE GET_KEY_DEFINITION; FORWARD;
PROCEDURE FILL_KEYS; FORWARD;


PROCEDURE INITIALIZE_UDKS;
(*
  This routine is called on power up reset to initialize
  the UDK state information to its default values.
*)
BEGIN
(* the following codes are in the specified range,
and are the valid programmable key codes for Level 2 *)
LEGAL_UDK_VALUES:= [17..21,23..26,28..29,31..35];
(* the following are valid hex codes, 0-9, A-F, and a-f *)
HEX_CODE_VALUES:= [48..57,65..70,97..102];
UDK_BUFFER_OVERFLOW:= FALSE;
TOTAL_CODE_COUNT:= 0;
FOR N:= MIN_UDK_VALUE TO MAX_UDK_VALUE DO
  UDK[N].CODE_COUNT:= 0;
(* NOTE: The actual value of the Lock Control on
initialization is specified by the user in Setup *)
UDK_LOCK_CONTROL:= UDK_LOCK_OFF;
END;
```

```
PROCEDURE EXECUTE_UDK (EVENT: EVENT_BUFFER_TYPE);
(*
  This routine is called each time a DECUDK control
  string introducer is received. It provides the top
  level control for the key loading sequence.
*)
BEGIN
GET_UDK_PARAMETERS (EVENT);
IF VALID_UDK_SYNTAX THEN
  BEGIN
  UDK_STRING_TERMINATED:= FALSE;
  IF UDK_LOCK_CONTROL = UDK_LOCK_OFF THEN
    BEGIN
    (* clear all udk's if requested *)
    IF UDK_ERASE_CONTROL = ERASE_ALL_UDK THEN
      BEGIN
      FOR N:= MIN_UDK_VALUE TO MAX_UDK_VALUE DO
        UDK[N].CODE_COUNT:= 0;
      TOTAL_CODE_COUNT:= 0;
      END;
    (* then process the command string *)
    FILL_KEYS;
    END;
  (* if locked or overflowed, keep going until the end *)
  WHILE NOT UDK_STRING_TERMINATED DO
    BEGIN
    GET_NEXT_EVENT (EVENT);
    IF (EVENT.EVENT_CODE = CONTROL_CODE)
    AND (EVENT.CODE_VALUE = 156) THEN
      UDK_STRING_TERMINATED:= TRUE;
    END;
  END;
(* set the lock if requested *)
IF LOCK_SET THEN UDK_LOCK_CONTROL:= UDK_LOCK_ON;
END;


PROCEDURE GET_UDK_PARAMETERS;
(* get all of the parameters for the load *)
BEGIN
VALID_UDK_SYNTAX:= TRUE;
CASE EVENT.PARAMETERS[1] OF
  0: UDK_ERASE_CONTROL:= ERASE_ALL_UDK;
  1: UDK_ERASE_CONTROL:= NO_ERASE_UDK;
  OTHERWISE VALID_UDK_SYNTAX:= FALSE;
  END;
CASE EVENT.PARAMETERS[2] OF
  0: LOCK_SET:= TRUE;
  1: LOCK_SET:= FALSE;
  OTHERWISE VALID_UDK_SYNTAX:= FALSE;
  END;
END;
```

```
PROCEDURE FILL_KEYS;
(*
  This routine processes the command string and loads the
  key definition strings into the appropriate key buffers.
*)
BEGIN
UDK_STRING_TERMINATED:= FALSE;
(* get all of the key definitions in the strings *)
REPEAT
  KEY_CODE_TERMINATED:= FALSE;
  DEFINITION_TERMINATED:= FALSE;
  GET_KEY_CODE;
  IF KEY_CODE_TERMINATED THEN
    BEGIN
    (* fill work buffer with code string *)
    GET_KEY_DEFINITION;
    (* ignore out of range key codes *)
    IF (KEY_CODE IN LEGAL_UDK_VALUES)
    AND (CODE_PAIR_TERMINATED) THEN
      BEGIN
      IF TOTAL_CODE_COUNT + I <= MAX_SIZE_UDK_BUFFER THEN
        BEGIN
        UDK[KEY_CODE].CODE_BUFFER:= WORK_BUFFER;
        TOTAL_CODE_COUNT:=
        TOTAL_CODE_COUNT - UDK[KEY_CODE].CODE_COUNT;
        UDK[KEY_CODE].CODE_COUNT:= I;
        TOTAL_CODE_COUNT:= TOTAL_CODE_COUNT + I;
        END
      ELSE UDK_BUFFER_OVERFLOW:= TRUE;
      END
    ELSE IGNORE;
    END;
(* these conditions cause the rest of the string to be ignored *)
UNTIL (NOT CODE_PAIR_TERMINATED) OR (UDK_BUFFER_OVERFLOW)
OR (UDK_STRING_TERMINATED);
END;
```

```
PROCEDURE GET_KEY_CODE;
(*
  This routine parses a single key selector code
  from the command string.
*)
BEGIN
KEY_CODE:= 0;
REPEAT
  GET_NEXT_EVENT (EVENT);
  CASE EVENT.EVENT_CODE OF
    GRAPHIC_CODE:
      BEGIN
      IF (EVENT.CODE_VALUE >= 48)
      AND (EVENT.CODE_VALUE <= 57) THEN
        KEY_CODE:= (KEY_CODE * 10) + (EVENT.CODE_VALUE - 48)
      ELSE
        BEGIN
        CASE EVENT.CODE_VALUE OF
          47: KEY_CODE_TERMINATED:= TRUE;
          59: DEFINITION_TERMINATED:= TRUE;
          OTHERWISE IGNORE;
          END;
        END;
    END;
    CONTROL_CODE:     (* terminate on String Terminator, 9/12 *)
      IF EVENT.CODE_VALUE = 156 THEN UDK_STRING_TERMINATED:= TRUE
      ELSE IGNORE;    (* ignore other control characters *)
    OTHERWISE IGNORE; (* ignore escape and control sequences *)
    END;
UNTIL (KEY_CODE_TERMINATED) OR (DEFINITION_TERMINATED)
OR (UDK_STRING_TERMINATED);
END;
```

```
PROCEDURE GET_KEY_DEFINITION;
(*
  This routine parses a single key definition string from the
  command string.
*)
BEGIN
I:= 0;
CODE_PAIR_TERMINATED:= TRUE;
REPEAT
  GET_NEXT_EVENT (EVENT);
  CASE EVENT.EVENT_CODE OF
    GRAPHIC_CODE:
      BEGIN
      IF EVENT.CODE_VALUE IN HEX_CODE_VALUES THEN
        BEGIN
        (* convert ascii codes to hex values *)
        EVENT.CODE_VALUE:= EVENT.CODE_VALUE - 48;
        IF EVENT.CODE_VALUE >= 65 THEN
          EVENT.CODE_VALUE:= EVENT.CODE_VALUE - 7;
        IF EVENT.CODE_VALUE >= 97 THEN
          EVENT.CODE_VALUE:= EVENT.CODE_VALUE - 22;
        (* then pack into buffer *)
        IF CODE_PAIR_TERMINATED THEN
          BEGIN
          I:= I+1;
          WORK_BUFFER[I]:= 0;
          CODE_PAIR_TERMINATED:= FALSE;
          END
        ELSE CODE_PAIR_TERMINATED:= TRUE;
        WORK_BUFFER[I]:=
        ((WORK_BUFFER[I]) * 16) + EVENT.CODE_VALUE;
        END;
      IF EVENT.CODE_VALUE = 59 THEN
        DEFINITION_TERMINATED:= TRUE;
      END;
    CONTROL_CODE:     (* terminate on String Terminator, 9/12 *)
      IF EVENT.CODE_VALUE = 156 THEN UDK_STRING_TERMINATED:= TRUE
      ELSE IGNORE;    (* ignore other control characters *)
    OTHERWISE IGNORE; (* ignore escape and control sequences *)
    END;
. UNTIL (DEFINITION_TERMINATED) OR (UDK_STRING_TERMINATED);
END;
```

11.4 UDK KEY DEFINITION EXAMPLES

To clear the UDKs send the following sequence:

```
DCS 0 ; 1 | ST
```

To lock the UDKs send the following sequence:

```
DCS 1 ; 0 | ST
```

The following is a generalized example of the recommended way to load the UDKs:

```
DCS 0 ; 1 | Ky1 / St1 ; Ky2 / St2 ; ... ; Kyn / Stn ST
```

11.5 CHANGE HISTORY

11.5.1 Revision 0.0 To AX10

The spec was entirely rewritten from the original draft. The following sections were added:

1. Introduction
2. Functional Description
3. Report UDK Status (control function)
4. Extensively modified the UDK load sequence, DECUDK
5. Pascal coded algorithms
6. UDK load examples

This page deliberately left blank.

Section Index

DEC STD 070-12 VIDEO SYSTEMS REFERENCE MANUAL - TERMINAL SYNCHRONIZATION

DOCUMENT IDENTIFIER: A-DS-EL00070-12-0000 Rev A1, 26-Jul-1990

ABSTRACT: This standard applies to serial asynchronous communications which use XON/XOFF flow control or Break for synchronization. It describes the XON/XOFF flow control protocol including response time and input buffer requirements. It also specifies requirements for implementing the Break function in terminals. This standard supersedes DEC STD 111-0 Terminal Synchronization.

APPLICABILITY: Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in DEC STD 070-1 Concepts and Conformance Criteria.

STATUS: APPROVED 26-Jul-1990; use VTX SMC for current status.

DEC STD 070-12 Video Systems Reference Manual - Terminal Synchronization

DOCUMENT IDENTIFIER: A-DS-EL00070-12-0000 Rev A1, 26-Jul-1990

REVISION HISTORY:	Rev A,	24-Nov-1987	
	Rev A1,	27-Jul-1990	ECO Number CTS01

Document Management Category:	Terminal Interface Architecture (STI)
Responsible Department:	VIPS Architecture Group
Responsible Person:	Peter Sichel

APPROVAL: This document, prepared by the VIPS Architecture Group, has been recommended for approval by the general review group for its category for use throughout Digital. Rev A1 incorporates clerical changes only.

Peter A Sichel

Peter Sichel - VIPS Architecture Group

Eric A Williams

Eric Williams - Standards Process Manager

Direct requests for further information to:

Peter Sichel
Use $ VTX ELF for the latest location information.

Use VTX SMC to order copies of this document from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

CONTENTS

1 INTRODUCTION

This document describes synchronization mechanisms for software which interfaces computers to terminal devices. These mechanisms are used for flow control to prevent devices from losing data received faster than it can be processed; to synchronize keyboard and display processing in terminals with software running on remote hosts; and to reset or initialize the communications context. Currently these mechanisms are defined for use over serial asynchronous communications lines, and include XON/XOFF flow control and Break.

XON/XOFF flow control is a means of using control characters DC1 (XON) and DC3 (XOFF) to alternately suspend and resume the flow of characters over serial asynchronous communication lines.

Break is an out of band signal frequently used to reset the communications environment or data interchange context over serial asynchronous lines. The Break condition is defined as a continuous space (transmission line held in its zero state) for longer than one character time. Although there is no standard use for Break, Digital terminals allow the Break signal to be generated under user control.

1.1 SCOPE

This standard applies to all Digital products which implement XON/XOFF flow control, all products which are intended to be certified as conforming to the Terminal Interface Architecture, and software which is intended to use terminal interfaces in a conforming manner.

1.2 RELATIONSHIP TO TIA

XON/XOFF flow control and break are part of the external communications interface that lies below the code extension layer. Flow control and break are management functions that control the communications link. No conforming applications should depend on the presence of, or performance characteristics of terminal synchronization.

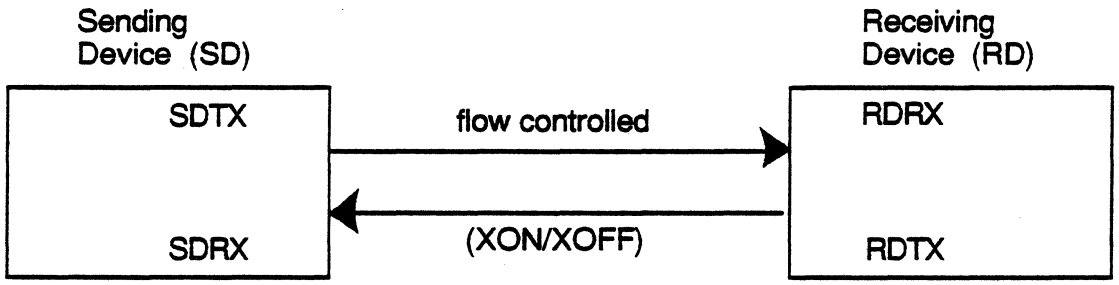
2 XON/XOFF FLOW CONTROL

2.1 PROTOCOL

XON/XOFF flow control is a means of using C0 control characters DC1 (1/1) (XON) and DC3 (1/3) (XOFF) to alternately suspend and resume the flow of characters over serial asynchronous point to point communication lines. Flow control is used to avoid losing data when a sending device is capable of transmitting data faster than the corresponding receiving device can process or use it.

XON/XOFF flow control is only applicable to full duplex communication lines with a specified minimum amount of receive data buffering. The exact buffer requirements depend on the maximum transmission rate and response time of the sender and receiver.

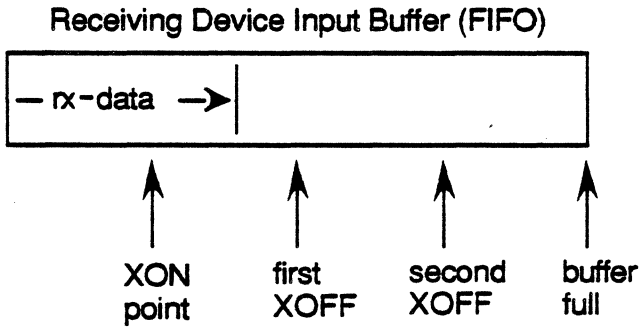
XON/XOFF flow control is a symmetric protocol and may be used independently in either or both directions. Only one direction is described below for simplicity. Most systems will conform as both a sending device and a receiving device. Product documentation should state clearly if an implementation intends to only conform as a sending device or receiving device.



When XON/XOFF flow control is enabled between a sending device and a receiving device, XON and XOFF characters received at the sending device (SDRX) are interpreted as flow control and are not stored in the sending device's input buffer. Because they are used for flow control, characters XON and XOFF may not be passed as part of the normal data interchange from the receiving device to the sending device.

The sending device recognizes XOFF (1/3) as a request from the receiving device to suspend the transmission of characters. When this character is received by the sending device, transmission of all characters (other than XON or XOFF) to the receiving device is suspended. When the sending device's communication port is in this state, it is said to be in the XOFF state. If the device is already in the XOFF state when an XOFF character is received, this character is ignored.

The sending device recognizes XON (1/1) as permission from the receiving device to resume the transmission of characters, if any. When this character is received by the sending device, the XOFF state of the device's communication port is cleared and transmission of characters in the communication port output buffer is resumed. If the device was not XOFF-ed, this character is ignored.



The receiving device transmits XOFF to the sending device if a character other than XON or XOFF that would be stored in the input buffer is received from that device, and (1) the number of characters in the input buffer reaches the first XOFF point for the first time since the last XON was sent; (2) the number of characters in the input buffer reaches the second XOFF point for the first time since the last XON was sent; or (3) the input buffer is full. If the input buffer is full, the received character is discarded. An error indication

should be made in the receiving device if characters are discarded (not mandatory). A terminal might display the error character for example.

If the sending device can respond to XOFF fast enough to avoid buffer overflow from the second XOFF point, reliability is improved on systems where the first XOFF might be lost during transmission. Support for the second XOFF point is not mandatory, but recommended for use over noisy communication channels.

The reason to send an XOFF for each character that arrives while the input buffer is full is to insure that the sending device will receive an XOFF even if it is reset (clearing the XOFF state) after the input buffer was filled. Transmission of XOFF for each character received while the input buffer is full is not mandatory.

The receiving device transmits an XON to the sending device when the receiving device's input buffer empties to the XON point, and the last flow control sent by the receiving device was XOFF. This XON must not be blocked by any data being held in the receiving device's transmit queue. If both devices had transmit queues and did not allow XON characters to bypass them, they could become locked in a "deadly embrace". Special provision shall be made to send XON promptly regardless of the XON/XOFF state of the receiving device.

2.2 XON AND XOFF POINTS

2.2.1 Requirements

The choice of XON and XOFF points will depend on the receive buffer size and the performance characteristics desired. The following requirements apply to the selection of XON and XOFF points. In all cases, the first XOFF point must be higher (closer to buffer full) than the XON point.

Terminals shall provide at least 32 characters of receive buffer space beyond each XOFF point supported (the buffer full condition is not considered an XOFF point).

When an XOFF point is reached, the receiving device should transmit XOFF as soon as possible to maximize the response time available to the sender. It is anticipated that some hardware devices will employ a transmit output buffer and will not be able to insert XOFF into the output stream immediately. In the worst case, XOFF shall be transmitted within five character times from when the XOFF point was reached.

Some implementations allow the first XOFF point to be selected under local control to limit the number of characters ahead of the screen the host can transmit. For interactive terminals with large input buffers, it shall be possible to set the first XOFF point to 64 characters or less (in the receive input buffer).

2.2.2 Guidelines and Examples

The choice of XON and XOFF points is complicated because there are several competing objectives that can be optimized. XON and XOFF points should be chosen to provide a reasonable compromise between these objectives for the intended use of the device. For some devices it may be desirable to make the XON and XOFF points user configurable.

To increase the amount of response time available to accomodate end to end signaling delays (satellite relay of telephone signals can take hundreds of milliseconds for example), it is desirable to provide lots of receive buffer space after the first XOFF point.

To allow a sender to finish a printing task as early as possible, it is desirable to make the XON point high (closer to buffer full).

To reduce the amount of XON/XOFF traffic and/or the number of data transfers, it is desirable to have the XON and (first) XOFF point far apart.

To maximize throughput, it is desirable to balance the XON response time (before the receive input buffer empties) with the XOFF response time so the receiver does not sit idle waiting for the sender to respond to XON.

For interactive video terminals that provide 32 characters of input buffer space beyond the last XOFF point, the XON point will normally be at 32 characters, or at half the number of characters to the first XOFF point, whichever is less.

EXAMPLES

The VT100 provides a 64-character receive input buffer with the first XOFF point at 32 characters, no second XOFF point, and XON point at 16 characters.

The VT220 provides a 254-character receive input buffer with the first XOFF point selectable to 64 or 128 characters. The second XOFF point is 220 characters. The XON point is 32 characters.

The VT100 and VT220 are designed to provide 32 characters of input buffer space to respond to XOFF (from the XOFF point to buffer full on the VT100, and from the second XOFF point to buffer full on the VT220).

2.3 RECEIVE BUFFER SIZE

Terminals shall provide at least 64 characters of receive input buffer space, with at least 32 characters of input buffer space beyond each XOFF point supported.

Although existing Digital Terminals respond to XOFF within two character times, other receiving devices (including host computers) should provide at least 32 characters of input buffer space beyond the XOFF point to be symmetric with the VT100 and VT200 terminal families (not mandatory). This is desirable to support PCs and Workstations running terminal emulator software.

In all cases, the receive input buffer size and XOFF point must be sufficient to prevent buffer overflow under the intended operating conditions.

2.4 BUFFER OVERFLOW PREVENTION

In order to prevent buffer overflow, the sending device must respond to an XOFF within some maximum amount of time which depends on the size of the receiver's input buffer, its XOFF point, and the transmission rate.

2.4.1 Number of Characters to Buffer Overflow

When the sending and receiving transmission rate are the same, the maximum guaranteed number of characters before buffer overflow assuming immediate response is:

```
OVFL = (MXBF - XOFF) - 3
```

Where:

```
OVFL = the number of characters before overflow
MXBF = the receive buffer size
XOFF = the XOFF point
```

The constant '3' derives from the following possible character delays: (1) When the receive buffer reaches the XOFF point, the receiving device may be in the middle of sending a character and unable to send XOFF until the character finishes; (2) It takes one character time to transmit the XOFF; (3) Assuming the transmitting device can respond immediately to the XOFF, it must still finish the character it is currently sending (if any).

Situations that may reduce the number of characters before buffer overflow include:

1. Systems that support different transmit and receive speeds (the slower channel cannot send or receive the XOFF as fast OVFL = (MXBF-XOFF) - [3(RCDR/XMDR)]).
2. Systems that utilize transmit output buffers and cannot insert XOFF into the output stream immediately.
3. Systems that employ "transmit rate limiting" to limit the number of characters per second below the maximum allowed by the baud rate.

DEVIATION NOTE

The number of characters before buffer overflow on the VT220, and VT300 family is calculated differently because received characters are only handled during V Blank time. The actual formula is:

```
OVFL = (MXBF-XOFF)-[3(RCDR/XMDR)]-(RCDR/600)
```

The last term takes into account the worst case for when the received character would get serviced.

The VT220 and VT300 series will not consistently transmit XOFF within five character times of reaching an XOFF point (but they still provide more than 32 characters of buffer space from when the first XOFF is sent). This delay until V Blank time should be discouraged in future devices.

2.4.2 Response Time to XOFF

The maximum response time available to the sending device to stop sending (in order to guarantee that the buffer will not overflow) is the time it takes to transmit the number of characters until buffer overflow (OVFL).

```
    MRST = OVFL x [(DATA + STOP + PRTY + 1)/XMDR]
where:
    MRST = maximum response time
    OVFL = the number of characters before overflow
    DATA = the number of data bits per character
    STOP = the number of stop bits per character
    PRTY = the number of parity bits per character
    '1'  = the start bit
    XMDR = the transmit data rate (bits per second)
```

It is recommended (but not mandatory) that the sending device stop transmitting immediately upon receipt of an XOFF (after the current character).

XON/XOFF flow control is intended for use over point to point full duplex links. The time available to respond to XOFF will be reduced by any end to end signaling delays.

2.4.3 Response Time Example

If the sending device is a host computer intended to work with a VT100 at 9600 bits per second, the maximum response time can be computed as follows:

```
OVFL = (MXBF - XOFF) - 3
     = (64   - 32)   - 3
     = 29 characters until overflow
```

Assuming 8-bit characters with no parity and one stop bit:

```
MRST = OVFL x [(DATA + STOP + PRTY + 1)/XMDR]
     = 29   x [(8    + 1    + 0    + 1)/9600]
     = 0.030 seconds
```

For this example, the sending device including transmission delays in either direction must respond to XOFF within 30 milliseconds to avoid buffer overflow.

2.5 INITIALIZATION

When a device first powers on, it has no way of knowing the XON/XOFF state of a communication link. Devices shall send a single XON at power-up when they are ready to receive data, and assume their transmitter is in the XON state.

Devices that can be locally controlled (that is, have some form of keyboard or local control panel) shall provide a means for the operator to re-initialize the XON/XOFF state. This "Clear Communications" function will include the following actions: (1) clear the device's transmit queue; (2) clear the XOFF state of the device's communication port; (3) transmit a single XON to the remote device.

GUIDELINE

The VT100 sends XON at power-up, and allows the XON/XOFF state to be re-initialized by entering and exiting Set-Up.

The VT220 sends XON at power-up. A "Clear Comm" function is provided in Set-Up to re-initialize the XON/XOFF state.

2.6 OTHER EVENTS THAT AFFECT XON/XOFF STATE

2.6.1 Special Characters

Some devices interpret other characters besides XON to mean "okay to resume transmission". These characters are not synonymous with XON and shall be treated as special conditions that re-initialize the XON/XOFF state. It is up to the device receiving data to decide what, if any, special characters will clear its XOFF state. A device sending data shall not assume any characters other than XON will clear the XOFF state of the receiving device.

GUIDELINE

Some operating systems have interpreted CTRL/Y and CTRL/C as events that re-initialize the XON/XOFF state. Upon receipt of CTRL/Y (when CTRL/Y handling is enabled), VMS clears the communication port's output queue, clears the XOFF state of the communication port, and transmits an XON to the terminal. Terminals should not assume, however, that sending CTRL/Y will clear the XOFF state of the remote device, and should not re-initialize their own XON/XOFF state upon sending CTRL/Y.

2.6.2 Local Events (Guidelines)

Some devices generate XON or XOFF flow control as a result of local events which affect the processing of incoming data.

The VT100 sends XOFF when entering Set-Up, and sends XON upon exit from Set-Up. The VT220 enters the hold screen state (if not already held) when entering Set-Up, and leaves the hold screen state if not previously held upon exiting Set-Up.

The LN03 sends XOFF when the paper cover is lifted and sends XON when the paper cover is replaced. It is suggested that printers send XOFF when they are unable to process incoming data (including going off-line), and send XON when they are again ready to process incoming data (including going on-line).

2.7 RESERVED CHARACTERS

DC1 (1/1) (XON) and DC3 (1/3) (XOFF) are to be used for synchronization in the manner described in this standard. DC2 (1/2) and DC4 (1/4) are reserved for future use, likely for synchronization. These characters should not be used to convey meaning to application software.

2.8 TERMINAL KEYBOARD SYNCHRONIZATION

When XON/XOFF flow control is enabled, terminals may be XOFF-ed to suspend transmission of characters to the host. Terminals will maintain a silo for keyboard output. In the event that the keyboard produces characters faster than they can be sent, the silo will buffer at least nine keystrokes. For the purpose of buffering, a compose sequence or function key counts as one keystroke. If the silo becomes full, keystrokes may be discarded. Terminals shall provide some indication that the silo is full and/or keystrokes are being discarded.

GUIDELINE

In the VT200 family, the Wait LED comes on when the keyboard silo is full, and the keyboard stops clicking (if keyclick was enabled) when keystrokes are being ignored.

2.9 TERMINAL SCREEN SYNCHRONIZATION (HOLD SCREEN)

When XON/XOFF flow control is enabled, terminals shall allow the data on the screen to be held for easy reading (stop scrolling) even though data is still being received from the host computer. The "Hold Screen" function is a toggle that switches the terminal screen between the HELD and NOT_HELD states. The Hold Screen function does not transmit XON or XOFF, but may cause the terminal's input buffer to fill when the screen is in the HELD state, and empty again when the screen is returned to the NOT_HELD state.

When XON/XOFF flow control is enabled, typing CTRL/S (XOFF) or CTRL/Q (XON) at the keyboard will not transmit codes to the host computer. Instead, these keystrokes will toggle the hold screen state between HELD and NOT_HELD respectively.

When XON/XOFF flow control is disabled, the Hold Screen function does not operate and typing CTRL/S (XOFF) or CTRL/Q (XON) at the keyboard will transmit the corresponding codes to the host computer.

2.10 EMERGENCY MESSAGES

At times, one device may need to send another a high-priority or emergency message. If the intended receiver has sent an XOFF however, it has indicated that it is not ready to receive more data. This may be because the receiver is literally unable to receive or process a message (a printing terminal with a blown fuse, for example), or it may just be that the operator has activated the Hold Screen function.

At the time of this update to the standard, there is no clear consensus on how to handle emergency messages.

DEVIATION NOTE

Some printers respond to device status inquiries immediately even when they are in the XOFF state (using the XOFF state as a control channel). This mechanism should be used only by explicit, device dependent agreement between the printer and supporting host. It is not part of the XON/XOFF protocol, and may not be supported in future devices.

2.11 IMPLIED XOFF RULE

Certain terminal functions during which the terminal is unable to respond to host data use an implied XOFF rule. The implied XOFF rule allows the host to suspend transmission until the terminal is again able to respond. When a host computer invokes one of these functions, it is to enter the XOFF state as if an XOFF had been received from the terminal (implied XOFF). Upon completing the requested function, the terminal must send an XON to re-enable host transmission.

Currently, only RIS (Reset to Initial State) and DECTST (DEC Test) use the implied XOFF rule. These functions are intended for diagnostics and shall not be used by conforming application software. Under normal operation, terminal devices should not stop monitoring the communication line, or should send XOFF before going "off line".

3 BREAK

Break is an out of band signal frequently used to reset the communications environment or data interchange context over serial asynchronous lines. The Break condition is defined as a continuous space (transmission line held in its zero state) for longer than one character time. See DEC STD 52 for additional information on physical generation of the break signal. Although there is no standard use for Break, Digital terminals allow the Break signal to be generated under user control.

The remainder of this section describes how the Break signal is transmitted, when it may be transmitted, and operation of the break key.

3.1 THE BREAK SIGNAL

Although Break is defined as a continuous space exceeding one character time, the duration of the Break signal generated by Digital terminals has been standardized for reliability. The Break signal will hold the transmission line to its zero state for 275 milliseconds (plus or minus 25 ms).

3.1.1 Long Break Disconnect

Some modems use a longer Break of 3.5 seconds to signal a remote data set to disconnect from the communications line. When used, this signal is generated by the modem. Terminals indicate disconnect to a modem using the DTR modem control signal. There is no known reason for a terminal to generate the 3.5 second long Break.

GUIDELINE

The VT220 and newer terminals do not generate the long break signal. On the VT100, pressing SHIFT/BREAK would generate a long break in addition to dropping DTR.

3.2 WHEN BREAK MAY BE TRANSMITTED

A Break signal may be transmitted by terminals only upon activation of the break key. Break shall never be transmitted during power-up, power-down, self-test, or terminal reset operations.

Previous terminals (VT100, VT220) have transmitted Break during power-up, and self-test. This use of Break should be eliminated for the following reasons:

- It does not conform to DEC STD 052-1 which requires TXD (transmit data) be held in the mark state during power-up.
- Break is used by some Digital systems to halt the console CPU. Transmitting Break at power-up is inappropriate for these systems.
- Sending Break during self-test precludes the use of remote diagnostics. If a host requests a terminal self-test, the resulting Break may cause a disconnect.

3.3 OPERATION OF THE BREAK KEY

When the break key is activated, special provision shall be made to transmit a Break signal immediately regardless of keyboard locking (KAM), XON/XOFF flow control, or the state of the keyboard input silo. If a character is currently being transmitted, the Break signal will begin as soon as the stop bit of the current character has completed to avoid loss of data. Activating the break key does not clear the keyboard input silo, reset KAM, or change the XON/XOFF flow control state.

3.4 RECEIPT OF BREAK

Terminals shall detect break explicitly and do nothing at all when break is received at the host communication port.

APPENDIX A REFERENCED DOCUMENTS

A.1 EL-Class Digital Documents

EL-Class Number	Document Title
EL-00052-00	*DEC STD 052-0 Operational Requirements for Serial Terminals and Computer Interfaces Operating as DTEs Connected to EIA RS-232-C or CCITT V.28 Modems*
EL-00052-01	*DEC STD 052-1 Operational Requirements for Serial Data Terminal Equipment and Serial System Interfaces Operating as DTEs Connected to EIA RS-232-C or CCITT V.28 Point-To-Point Modems or Data Service Units*

Use VTX SMC to order copies of EL-class documents from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

INDEX

```
+-------------------------------------------------------------------+
|                        READER COMMENTS                            |
|                                                                   |
| Your comments and suggestions will help Standards and Methods     |
| Control improve their services and documents.                     |
+-------------------------------------------------------------------+
```

Did you request this document? _____ If so, did it arrive within a satisfactory period of time? _____ Please comment.

What are your impressions of this document? Consider format, organization, completeness, readability, and illustrations.

FOLD ON THIS LINE

Did you find technical or clerical errors in this document? If so, please specify the page number(s) and the error(s).

Are the instructions for the update package clear? _____
Was an index available? _____ If not, is one needed? _____
Do you have other suggestions for improving this document?

The following information is optional:

Name _____ Mailstop _____

Department _____ Node _____

Send your comments to JOKUR::PROJECTS, or fold, staple, and send this page through interoffice mail to:

```
+------------------------------+
|       READERS' COMMENTS      |
| STANDARDS AND METHODS CONTROL|
|           CTS1-2/D4          |
+------------------------------+
```

DEC STD 070-13 Video Systems Reference Manual—Text Locator Extension

DOCUMENT IDENTIFIER: A-DS-EL00070-13-0000 Rev A, 30-Sep-1991

ABSTRACT: This specification defines support for locator input devices on ANSI text and graphics terminals as an extension to the Character Cell Display service class.

APPLICABILITY: Mandatory for engineers designing hardware for terminal products and software engineers designing programs using terminal interfaces. Additional requirements are defined in DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria.

STATUS: APPROVED 30-Sep-1991; use VTX SMC for current status.

The material contained within this document is assumed to define mandatory standards unless it is clearly marked as: a. not mandatory; or b. guidelines.

Material that is marked as "not mandatory" is considered to be of potential benefit to the Corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques that are considered to be good practice, but should not be considered as requirements.

DEC STD 070-13 Video Systems Reference Manual—Text Locator Extension

DOCUMENT IDENTIFIER: A-DS-EL00070-13-0000 Rev A, 30-Sep-1991

REVISION HISTORY:	Rev A,	30-Sep-1991
Document Management Category:		Terminal Interface Architecture (STI)
Responsible Department:		VIPS Terminals Architecture
Responsible Person:		Peter Sichel

APPROVAL: This standard, prepared by the Desktop Systems Group, has been reviewed and recommended for approval by the General Review group for its category.

Peter A Sichel

Peter Sichel - VIPS Terminals Architecture

Eric A Williams

Eric Williams - Standards Process Manager

Direct requests for further information to:

Peter Sichel
Use $ VTX ELF for the latest location information.

Use VTX SMC to order copies of this document from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 234-4423.

CONTENTS

FIGURES

1 INTRODUCTION

This specification defines support for locator or pointing devices on ANSI text and graphic terminals as an extension to the Character Cell Display service class. Pointing devices are used for entering coordinates or selecting items on a display.

A conforming device shall be supplied with a locator port used to connect an optional mouse or tablet. When a locator device is present and enabled, a separate input cursor appears that is moved by the terminal to follow the locator without host intervention. Individual locator events, such as locator button transitions or movements, can be programmed to send locator reports to the host.

Once enabled, locator reports can be unsolicited (event driven) and mixed freely with keyboard input.

Controls are provided to enable locator reporting, select which events shall generate reports, and interrogate the status of the locator device.

This specification is primarily concerned with the Text Locator Extension, but also documents a simplified locator interface that forms the Locator Port Extension. Requirements apply to the Text Locator Extension unless explicitly indicated in the text.

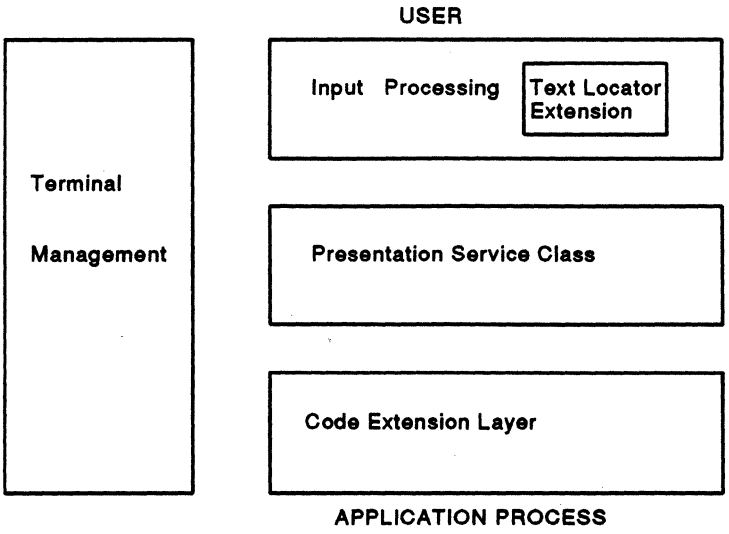
1.1 SCOPE

This standard applies to all Digital products that implement support for a locator device and are intended to be certified as conforming to the Terminal Interface Architecture (TIA) as shown in Figure 1. It also applies to software intended to use the locator interface in a conforming manner.

1.2 Relationship to TIA

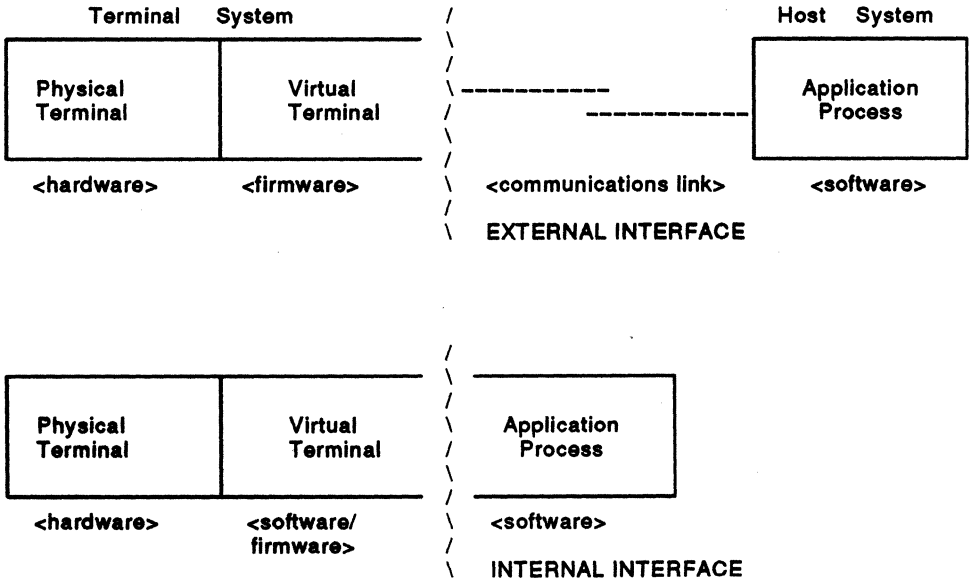
The locator interface forms two extensions to the Character Cell Display service class: Text Locator and Locator Port. Both can be implemented at any conformance level. The primary Device Attributes (DA) response will report the locator extensions as parameter value 15 (Locator Port), and parameter value 29 (Text Locator). The Locator Port Extension supports DECLKD. If the device includes the ReGIS or sixel graphics extensions, additional interactions with these protocols are defined. The Text Locator Extension supports the text locator features described in this document (DECELR, DECLRP, DECRQLP, DECSLE, DECEFR). Both extensions allow the status of the locator port to be interrogated using a Device Status Report (DSR) function.

Figure 1: Structuring of the Terminal Interface Architecture



The interfaces defined within this specification apply to both internal and external product interfaces (as shown in Figure 2). External interfaces are interfaces between a terminal, personal computer or workstation and a remote system. Internal interfaces are interfaces between a terminal subsystem and software processes running within a terminal, personal computer, or workstation.

Figure 2: External and Internal Terminal Interfaces



2 TERMINOLOGY

Filter rectangle—A rectangular area in a video display used for selecting (filtering) which locator movements will be reported to the host. Moving the locator so that the input cursor is moved outside the filter rectangle causes a locator report to be transmitted to the host.

Locator—A device such as a mouse or tablet used for entering position coordinates (pointing) on a video display.

Session—A connection between a single virtual terminal and a host computer.

Virtual terminal—Consists of all the context (state information) of a real terminal but is able to share physical resources (such as the keyboard, display, and communication lines) with other virtual terminals. This allows a single physical device to appear as more than one terminal to a host computer.

Window—An area of the screen used to display information from a virtual terminal.

Window manager—Software or firmware that controls the use of windows.

3 LOCATOR REPORTING MODEL

When a locator device is present and enabled, a separate input cursor appears that is moved by the terminal to follow the locator without host intervention. Individual locator events such as locator button transitions or movement can be programmed to send locator reports to the host.

Each locator report shall include: (1) the specific event that initiated the report; (2) the current state of the locator keys; and (3) the coordinates of the input cursor at the time of the event. The events that can initiate a locator report are a button transition (up or down), locator movement, or an explicit host request.

Reporting both the event that triggered the report, and the current button state allows software to interpret locator activity without keeping track of previous events or button state. In a multiprocess shared locator environment, an application may not know the previous button state. This dual reporting also allows applications to recover from lost locator reports.

Each locator event generates a single report. In the rare situation where two events occur simultaneously (within a single-sampling period), the terminal shall report this as two separate events. The order of reporting shall be by increasing event code number (left button first).

Locator events are queued in the keyboard input silo just like keystrokes. Each locator event occupies one position in the silo (the keyboard silo currently must have at least nine positions). If the input silo becomes full, the locator and keyboard are locked until there is again room in the silo. The sequential order of keystroke and locator events shall be strictly maintained.

Guideline: It is the responsibility of the host to accept data fast enough to avoid locking the locator unintentionally. The limited buffering inside the terminal gives the host a little more time to process locator events smoothly.

When the locator is locked, the terminal continues to track the locator but the input cursor changes shape to indicate the locked condition (not mandatory). The alternate cursor shape indicates that locator button transitions will be ignored, but allows the user to continue positioning in anticipation of the locator being unlocked. The locator is automatically locked any time the input silo is full.

Locator-ahead, analogous to keyboard type-ahead is supported by having each report include the locator position at the time of the event, and maintaining the sequential order of keystroke and locator events.

3.1 INTERACTION WITH SESSIONS AND WINDOWS

Since locator events are queued in the keyboard input silo along with keystrokes, locator and keyboard input should be associated with the same session (virtual terminal) at all times. The session to receive these events is sometimes called the *active session*.

In a multisession windowing environment, the input cursor is allowed to roam freely over the entire screen in response to locator movement. The input cursor is never occluded when locator reporting is enabled in one or more sessions. Each session enables locator reporting independently. The following requirements apply to locator interaction with session windows and scroll regions in a multisession windowing environment.

1. The input cursor is within the active session's window. Pressing a button on the locator sends a locator report when enabled.
2. The input cursor is inside the active session's window but outside the range of defined coordinates for that session. Pressing a button on the locator shall generate a report with omitted coordinates (position undefined). For example, the input cursor is outside the active scrolling region, and the origin mode is set to relative.
3. The input cursor is not contained in any virtual terminal's window. Pressing a button on the locator shall have no effect on any of the virtual terminals.
4. The input cursor is within a window of a session that is not the active session. Pressing a button on the locator will normally make the session containing the locator cursor the active session, possibly changing the occlusion order of windows and the shape of the locator cursor (not mandatory).

   In this case, the *window manager* is free to define its own user interface. Two recommendations (not mandatory) are: (1) no locator report should be sent to the previous active session, because the locator is not in its window; and (2) a locator report should not be transmitted if locator reporting is enabled in the new session, to avoid application side effects when selecting another window.

   Implementation Guideline: DECterm Version 1.0 followed recommendation (1) above, but did not follow recommendation (2). The result is that selecting a new DECterm window often repositions the cursor in that window. Personal computer windowing systems usually require the user to click once to select the window and click again in the window to move the active position.

4 CONTROLLING LOCATOR REPORTS

ENABLE LOCATOR REPORTS DECELR

Levels: 1x, 2x, 3x, 4x (Text Locator)

Purpose: To enable locator reporting

Format:

CSI	Ps	;	Pu	'	z
9/11	3/?	3/11	3/?	2/7	7/10

The value of Ps controls locator reporting as follows:
0—Locator disabled (default)
1—Locator reports enabled
2—One shot (allow one report, then disable)

Pu specifies the coordinate units for locator reports:
0—(or omitted) Default to character cells
1—Device physical pixels
2—Character cells

Description: When disabled (the power-up default), the locator cursor does not appear, and the locator buttons are inactive. When enabled, the locator cursor is visible, and the terminal tracks the locator locally with no host intervention. Individual locator events, such as locator button transitions or movements, may be programmed to send locator reports to the host.

One shot mode is provided for applications that desire simple graphics input similar to Tektronix GIN mode (no unsolicited reports). If parameter value 2 is selected, the next trigger event that occurs generates a single locator report. No further locator reports occur (the locator is disabled), until another DECELR sequence is received.

The coordinate units for locator position reports may be selected to either of two coordinate systems used by terminal software at the lowest level. Device physical pixels are useful when the device is capable of addressing individual dots on the display, such as with the sixel graphics extension. On devices that support character cell addressing only, device physical pixels (picture elements) may be character cells.

NOTE

Undefined values for Ps or Pu causes the entire sequence to be ignored.

State Affected:

locator_mode_t locator_mode;
locator_coordinate_mode_t locator_coordinate_mode;

LOCATOR REPORT **DECLRP**

Levels: 1x, 2x, 3x, 4x (Text Locator)

Purpose: To transmit locator information to the host

Format:

```
CSI     Pe     ;     Pb     ;     Pr     ;     Pc     ;     Pp     &     w
9/11                                                              2/6   7/7
```

Pe is the event code.
Pb is the button code.
Pr is the row coordinate.
Pc is the column coordinate.
Pp is the third coordinate (page number).

Description: When a selected trigger event occurs, such as a button press or release, the terminal transmits a locator report to the host as a DECLRP sequence.

Pe, the event code indicates what event caused this report to be generated. The following event codes are defined:

0—Request; the terminal received an explicit request for a locator report, but the locator is unavailable (not enabled, or physically not present)
1—Request; the terminal received an explicit request for a locator report
2—Left button down
3—Left button up
4—Middle button down
5—Middle button up
6—Right button down
7—Right button up
8—Fourth button down
9—Fourth button up
10—Locator outside filter rectangle

If Pe is 0 or omitted, all other parameters shall not be sent because the locator is unavailable. If Pe > 10, the entire sequence shall be ignored.

Pb is the button code, ASCII decimal 0 - 15, indicating which, if any, of up to 4 buttons are down. The state of up to 4 buttons on the locator corresponds to the low 4 bits of the decimal value; a 1 means button is depressed.

0—No buttons down
1—Right
2—Middle
4—Left
8—Fourth

If Pb is omitted, it shall be interpreted as Pb = 0, no buttons down.

Pr is the row coordinate of the locator position in the page encoded as an ASCII decimal value. If Pr is omitted, the locator position is undefined (outside the terminal window for example).

Pc is the column coordinate of the locator position in the page, encoded as an ASCII decimal value. If Pc is omitted, the locator position is undefined (outside the terminal window for example).

Pp is the page coordinate of the locator position encoded as an ASCII decimal value. The page coordinate may be omitted if the locator is on page one (the default).

NOTES

1. **If locator reporting has been enabled to *one shot* mode (DECELR with Ps = 2), a single locator report (DECLRP) shall reset locator reporting mode to disabled.**
2. **The parameter values Pr, Pc, and Pp, are expressed in the units selected by the last received Enable Locator Reports (DECELR) command.**
3. **The parameter values Pr and Pc shall be omitted, indicating locator position undefined, if a position report (DECLRP) is generated when the locator cursor is outside the range of defined coordinates for the reporting virtual terminal.**

State Affected:

locator_reporting_mode_t locator_reporting_mode;

Levels: 1x, 2x, 3x, 4x (Text Locator)

Purpose: Request the terminal transmit a locator position report

Format:

CSI	Ps	'	|
9/11		2/7	7/12

The value of Ps controls the locator report sent as follows:
0—(or omitted) Default to 1.
1—Transmit a single locator report (DECLRP).

Description: The host may explicitly request a Locator Position Report (DECLRP) at any time locator reporting is enabled through Enable Locator Reporting (DECELR). Upon receipt of DECRQLP, the terminal shall respond with a single DECLRP with an event code of 1.

If the session receiving the request is the active session, but the locator is not within the defined coordinate range for that session, the terminal shall respond with a single DECLRP with omitted coordinate parameter values.

If the session receiving the request is not currently active (the locator is being used in another session), the terminal shall respond with omitted coordinates (locator position undefined) even if the input cursor is physically over the inactive session's window. Locator state from the active session shall not be made available to inactive sessions.

If locator reporting is disabled, the terminal shall respond with a single DECLRP with an event code of 0 (indicating locator not available to avoid timing out the application).

APPLICATION NOTE

If the terminal responds that the locator position is undefined (omitted coordinates), the application can ask to be notified the next time that session is active, and the locator is within the defined coordinate range. See DECEFR (Enable Filter Rectangle).

Levels: 1x, 2x, 3x, 4x (Text Locator)

Purpose: To select which locator events will generate reports

Format:

```
CSI          Ps...Ps          '           {
9/11                          2/7         7/11
```

Ps ... Ps is one or more selective parameters that may assume the following values:
- 0—Respond only to explicit host requests (default, also cancels any pending filter rectangle)
- 1—Report button down transitions
- 2—Do not report button down transitions
- 3—Report button up transitions
- 4—Do not report button up transitions

Description: DECSLE selects which locator events shall generate unsolicited reports according to the parameter values.

Levels: 1x, 2x, 3x, 4x (Text Locator)

Purpose: To cause locator movement to be reported to the host.

Format:

```
CSI    Pt    ;    Pl    ;    Pb    ;    Pr    '    w
9/11                                          2/7  7/7
```

Pt—Top boundary of filter rectangle
Pl—Left boundary of filter rectangle
Pb—Bottom boundary of filter rectangle
Pr—Right boundary of filter rectangle

Description: The DECEFR control sequence defines the coordinates of a filter rectangle and activates it. The next time the locator is detected to be outside that filter rectangle, an outside rectangle event is generated and the rectangle is de-activated. Filter rectangles are always treated as *one-shot* events.

Defining a new rectangle cancels the previous definition, if any, and activates the new filter rectangle.

Pt, Pl, Pb, and Pr are in coordinates units specified by the last DECELR sequence. The filter rectangle includes the boundaries, similar to other rectangular area operations. The origin is coordinate pair 1:1 in the upper-left corner. If any parameters are omitted, they are defaulted to the current locator position. Sending DECEFR with no parameters will cause the application to be notified for any locator movement within the defined coordinate space for that virtual terminal (*unfiltered movement event*).

If a rectangle that does not contain the locator is specified, and the locator is within the defined coordinate space for that session, the terminal will generate an outside rectangle report immediately and de-activate it.

NOTES

1. **DECELR cancels any previous filter rectangle definition. This guarantees that when an application enables locator reports, there will never be an outstanding filter rectangle.**
2. **If a filter rectangle lies on the edge of the defined coordinate space for the active session, and the locator crosses that edge, the rectangle shall be triggered to send a report with omitted coordinates (locator position undefined).**

   If the active session receives a filter rectangle with explicit coordinates while the locator is outside the defined coordinate space, the rectangle shall be triggered to send a report with omitted coordinates (locator position undefined).

If the active session receives a filter rectangle with omitted coordinates (that is, uses the current position) while the locator is outside the defined coordinate space (position undefined), the rectangle shall be triggered the next time the locator is within the defined coordinate space. This allows applications to be notified the next time the locator is within the defined coordinate space without polling repeatedly.

3. **If a session that is not the active session receives a filter rectangle with explicit coordinates, it shall trigger immediately with position undefined. If a session that is not the active session receives a rectangle with omitted coordinates, it shall trigger the next time the locator is within the defined coordinate space for that session, which cannot happen until the session becomes active.**
4. **If the terminal receives a filter rectangle when locator reporting is not enabled, the rectangle shall be triggered to send a report indicating that the locator is not available. This is to avoid hanging the application while it waits for a locator report that will never occur.**

5 LOCATOR SUPPORT FOR GRAPHICS (LOCATOR PORT EXTENSION)

The locator can also be used with the ReGIS Extension, and in Tektronix GIN mode during Tektronix 401x emulation. See *DEC STD 070-8 Video and Printer Standards Reference Manual - ReGIS Graphics Extension* for more information on ReGIS commands.

5.1 ReGIS ONE-SHOT GRAPHICS INPUT MODE

ReGIS One-Shot Graphics Input Mode is selected with R(I0). Once selected, the application may request graphics input using the report position interactive command, R(P(I)).

ReGIS One-Shot Graphics Input Mode is provided for backward compatibility with the VT240. When an application requests graphics input, a cross hair cursor appears which may be positioned using either the arrow keys (as on the VT240) or a locator device. Pressing any non-arrow key that is not dead on the keyboard or a button on the locator will cause the following actions:

1. The ASCII code or codes for the non-arrow key that was pressed is sent to the host. Each locator button can be programmed to send a sequence of 0 to 6 characters. This allows the locator to be programmed to work with existing applications (see DECLKD, paragraph 5.4).
2. The coordinates of the input cursor at the time the non-arrow key was depressed are sent to the host, expressed as an absolute bracketed extent in user coordinates.
3. The graphics input cursor disappears from the screen and the terminal exits graphics input mode.

While the terminal is in graphics input mode, no further data from the application (screen output) is processed until the locator input is complete and a report is generated.

The mouse and arrow keys may be intermixed freely in ReGIS One-Shot Graphics Input Mode. Since the tablet is an absolute positioning device, use of the arrow keys in combination with the tablet would cause inconsistent and confusing behavior. The arrow keys are disabled for input cursor positioning when the tablet is in proximity because the tablet establishes an absolute screen position.

5.2 ReGIS MULTIPLE INPUT MODE

ReGIS Multiple Input Mode is selected with R(I1). In this mode, a graphics input cursor is displayed continuously, and individual locator events may be programmed to send unsolicited position reports to the host. Characters received from the host are executed instead of buffered, so graphics output and graphics input may occur simultaneously. In Multiple Input Mode, the input cursor continues to be displayed and the mode continues to be in effect until explicitly exited by command R(I0). The position of the input cursor may be polled with R(P(I)) at any time while in this mode. If report position interactive is received, the report will be prefixed by the *null button* because no button transition initiated the report.

Since host input is still being processed, it is possible to exit ReGIS (terminal receives String Terminator) while in Multiple Input Mode. If this happens, the ReGIS locator will be disabled until ReGIS is re-entered, at which time multiple input mode resumes with the input cursor at its last known position.

In Multiple Input Mode, locator key transitions for which the key definition is not empty (see DECLKD) send a position report as follows:

1. The locator key definition string for the corresponding button transition is sent.
2. The coordinates of the input cursor at the time of the key transition are sent to the host expressed as an absolute bracketed extent in user coordinates.
3. The report ends with a carriage return.

Examples:

```
<ESC>[241~[102.5,200]<CR>
```

The left mouse button was pressed. <ESC>[241~ is the default sequence for left button down transitions. Input cursor is at 102.5,200.

```
<ESC>[240~[100,100]<CR>
```

The terminal received R(P(I)) in multiple input mode. <ESC>[240~ is the *null button* sequence.

5.3 TEKTRONIX GIN MODE

Tektronix GIN Mode is only available during Tektronix 4010/4014 emulation. GIN mode is similar to ReGIS One-Shot Graphics Input mode with minor exceptions. The GIN request and resulting report sequences are compatible with the Tektronix 4010/4014 series terminals. Only the cross hair graphics cursor is supported. Tektronix 4010/4014 emulation is an exception to the architecture implemented by some graphic terminals.

5.4 DEFINING LOCATOR KEYS FOR GRAPHICS INPUT

LOCATOR KEY DEFINITION **DECLKD**

Levels: 1x, 2x, 3x, 4x (Locator Port)

Purpose: To assign character strings to the locator keys for ReGIS One-Shot or Multiple Graphics Input Mode

Format:

```
DCS  Pc   $    w   Ky1 / Std1 / Stu1 ; ... ; Kyn / Stnd / Stnu  ST
9/0       2/4  7/7                                             9/12
```

Pc—Clear parameter
0—(or omitted) Clear all LKDs before loading new values.
All button definition strings are empty (not the default).
1—Clear old definition only for keys that are redefined.

Kyn / Stnd / Stnu—Key selection code n, slash delimiter (2/15), and string parameter n for down and up key transitions. Kyn is a single ASCII digit, and Std/Stu is a series of ASCII hex pairs. If Std or Stu is omitted, the corresponding button string will be empty. Key definition strings are delimited with 3/11 (";").

Description: The DECLKD device control string downline loads one or more key definitions for use by the Locator device during ReGIS One-Shot or Multiple Graphics Input Modes. These definitions are called Locator Key Definitions (LKDs).

There are six bytes available to each of the locator keys (up to four). A power-up restart or Reset to Initial State (RIS) will cause the LKDs to be set to their default value. LKDs are not affected by Soft Terminal Reset (DECSTR).

Default Locator Key Definitions

No button transition ESC [240 ~
Ky1 down transition ESC [241 ~
Ky2 down transition ESC [243 ~
Ky3 down transition ESC [245 ~
Ky4 down transition ESC [247 ~

Where:

Ky1 = B1 = left = barrel button
Ky2 = B2 = middle = tip button
Ky3 = B3 = right
Ky4 = B4 (tablet cursor only)

Up Transitions are ignored by default. The following locator key up definitions are recommended:

Ky1 up transition ESC [242 ~
Ky2 up transition ESC [244 ~
Ky3 up transition ESC [246 ~
Ky4 up transition ESC [248 ~

NOTES

1. **DECLKD is intended for older applications that use ReGIS one-shot input mode with keystroke commands like A, B, C, and so forth. These commands can be re-bound to buttons on the mouse by sending DECLKD (possibly before running the application).**
2. **A locator transition for which the key definition is empty will not terminate ReGIS One-Shot Graphics Input Mode.**

5.5 COMBINING GRAPHICS AND ANSI LOCATOR INPUT

Support for graphics input in ReGIS is extended by allowing ANSI locator input (Text Locator Extension) to operate normally regardless of whether the host output stream is being interpreted as ANSI or ReGIS. If ANSI locator reporting is enabled, keystrokes and locator position reports can be generated without being solicited from the host. Characters received from the host are executed instead of buffered so graphics output and graphics input can occur simultaneously.

If One-Shot Graphics Input Mode is entered (ReGIS report position interactive) while ANSI locator reporting is enabled, the locator cursor should change to a cross hair (not mandatory). Pressing a non-arrow key on the keyboard shall send the character for that key followed by the ReGIS user coordinates as a bracketed extent. If a locator button had been pressed instead, the characters for the locator button (if any) would be sent followed by the ReGIS position report, and finally the ANSI locator report (DECLRP) if enabled. Upon sending the ReGIS position report, the terminal shall exit One-Shot Graphics Input Mode, and the input cursor shall return to its previous state.

The ANSI locator report is sent in the above situation (if enabled) to keep the two modes independent. This allows filter rectangles to continue operating, and also keeps the ANSI state (last DECLRP seen) consistent with the last button transition. The order of reports is chosen to allow a subroutine to send R(P(I)) and read the ReGIS response that follows without regard to whether ANSI locator reports are enabled.

Since the ANSI locator report is in the form of a control sequence, it should not interfere with ReGIS reports. No special procedure is used to indicate ReGIS data from the terminal. This is backward compatible with the VT240 since ANSI locator reports can only occur if ANSI locator reporting is enabled.

The suspension of graphics output during ReGIS One-Shot Graphics Input Mode remains the same. This is necessary to support rubber band cursors. Rubber band cursors use the ReGIS output position as one reference point, and the locator input position as the other. Rubber band cursors are specified as an option on the ReGIS report position interactive command.

6 LOCATOR DEVICE STATUS

DEVICE STATUS REPORT (Locator Status) **DSR**

Levels: 1x, 2x, 3x, 4x (Locator Port or Text Locator)

Purpose: Cause the terminal to return the current functional status of the Locator

Format:

CSI	?	Ps	n
9/11	3/15	Ps	6/14

Description: The DSR request with private selective parameter 55 or 56 causes the terminal to respond with a similar DSR control function, using a parameter from the list below to indicate its current functional status.

?55 Request locator status (command from host)
?53 No locator
?50 Locator ready
?58 Locator busy

?56 Host requests identity of device on locator port
?57;0 "Unknown Locator" response
?57;1 Device is Digital Corporate Mouse
?57;2 Device is Digital Corporate Tablet

The *no locator* response indicates no locator is currently plugged in, or the last device plugged in could not be initialized or has not transmitted a successful self-test message.

The *locator ready* response indicates a locator device is plugged in and has transmitted a successful self-test message.

The locator busy response can occur when an alternate session has selected locator controller mode (see following).

7 LOCATOR CONTROLLER MODE (GUIDELINE—DOCUMENTED EXCEPTION)

Locator Controller Mode allows the host to communicate directly with the locator device without terminal intervention (similar to printer controller mode). When locator controller mode is set, all data received at the host port is transferred directly to the locator port without interpretation by the display terminal. The only exceptions are the communications control characters XON/XOFF (if enabled), and the control sequence to disabled locator controller mode. All characters received at locator port are transferred to the host port without interpretation. The host assumes full responsibility for the locator device.

Locator controller mode is desirable for two reasons:

1. It allows the host to explicitly initialize or configure locator devices. A foreign locator device might not wake up in Digital format, for example.
2. It allows the locator port to be used for other auxiliary input devices.

Locator controller mode uses the Media Copy control sequence for a secondary auxiliary device:

Format:

```
CSI         6         i          (Turn off locator controller mode)
9/11        3/6       6/9

CSI         7         i          (Turn on locator controller mode)
9/11        3/7       6/9
```

APPENDIX A REFERENCED DOCUMENTS

A.1 EL-Class Digital Documents

EL-Class Number	Document Title
EL-00070-01	*DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria*
EL-00070-08	*DEC STD 070-8 Video and Printer Standards Reference Manual - ReGIS Graphics Extension*

Use VTX SMC to order copies of EL-class documents from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 234-4423.

INDEX

```
+-------------------------------------------------------------------+
|                        READER COMMENTS                            |
|                                                                   |
| Your comments and suggestions will help Standards and Methods     |
| Control improve their services and documents.                     |
+-------------------------------------------------------------------+
```

Did you request this document? _____ If so, did it arrive within a satisfactory period of time? _____ Please comment.

What are your impressions of this document? Consider format, organization, completeness, readability, and illustrations.

FOLD ON THIS LINE

Did you find technical or clerical errors in this document? If so, please specify the page number(s) and the error(s).

Are the instructions for the update package clear? _____
Was an index available? _____ If not, is one needed? _____
Do you have other suggestions for improving this document?

The following information is optional:

Name _____ Mailstop _____

Department _____ Node _____

Send your comments to JOKUR::PROJECTS, or fold, staple, and send this page through interoffice mail to:

```
+------------------------------+
|       READERS' COMMENTS      |
| STANDARDS AND METHODS CONTROL|
|           NRO4/D4            |
+------------------------------+
```

CHAPTER 14

STATUS DISPLAY EXTENSION

VIDEO SYSTEMS REFERENCE MANUAL

STATUS DISPLAY EXTENSION

Document Identifier: A-DS-EL00070-14-0 Rev AX01, 26-Oct-1990

ABSTRACT: This document describes the interface to video terminals implementing a distinct on screen status display. It specifies both the application program interface to select and control the status display, and the user interface including general appearance of the status display. The Status Display forms an extension to Level 2 and is required at Level 3 or higher of the Character Cell Display service classes.

APPLICABILITY: Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the

DEC STD 070-1 Concepts and Conformance Criteria.

STATUS: FOR REVIEW ONLY

type $ VTX SMC for expiration date.

The material contained within this document is assumed to define mandatory standards unless it is clearly marked as: (a) not mandatory; or (b) guideline. Material that is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guideline" defines approaches, and techniques that are considered to be good practice, but should not be considered as requirements.

TITLE: VSRM - STATUS DISPLAY EXTENSION

DOCUMENT IDENTIFIER: A-DS-EL00070-14-0 Rev AX01, 26-Oct-1990

REVISION HISTORY: Original Draft
VT340 Functional Spec 23-Oct-1985
Revision AX01 26-Oct-1990

Document Management Group: Terminal Interface Architecture (STI)
Responsible Department: VIPS Terminals Architecture
Responsible Person: Peter Sichel
SMC Writer:

APPROVAL: This standard, prepared by the Desktop Systems Group, has been reviewed and recommended for approval by the general review group for its category.

_____
Peter Sichel, Standard Owner
Video, Image, and Printing Systems

_____
Eric Williams, Standards Process Manager

Direct requests for further information to Peter Sichel, DSG1-2/C7, DTN 235-8374, HANNAH::TERMARCH

Use VTX SMC to order copies of this document from Standards and Methods Control. Send distribution questions to JOKUR::SMC or call DTN: 287-3724.

CONTENTS

14.1 INTRODUCTION

14.1 INTRODUCTION

The Status Display is a Level 3 feature (Level 2 extension) which provides the ability to display at least one additional line of 80 characters underneath the main display. The Status Display may be configured as Host Writable to form a separately addressed Logical Display, or as an Indicator Line to display local terminal state information.

14.2 FUNCTIONAL DESCRIPTION

14.2 FUNCTIONAL DESCRIPTION

Level 3 or higher conforming devices shall allow an additional line of characters to be displayed on the screen as a Status Display. The Status Display shall be one or more lines high and contain the same number of columns as the main Logical Display (usually 80 or 132).

The type of Status Display can be selected by host control function or from Set-Up as one of the following: (1) No Status Line (factory default); (2) Indicator Status Line (not mandatory for workstation or PC terminal emulators); (3) Host Writable Status Line. The type Status Display selected is saved in non-volatile memory (if available), so it becomes the power-up default.

The Indicator Status Line is required on traditional video terminals, but is not mandatory for workstation or PC terminal emulators because local device information is often maintained separately from the terminal in these implementations.

The Indicator Status Line is always selected while in Set-Up. If the terminal needs to display an error or warning message, the Indicator Status Line is temporarily replaced with the Error Line until the next keystroke.

The Host Writable Status Line can be considered a permanent separate logical display, that contains its own ANSI text state and settings. In general, controls which affect some portion of the Main Display do not affect the contents or the ANSI state of the Status Display. Exceptions are noted in this section (DEC STD 70-14). The cursor may not be positioned within the Status Display, unless the DECSASD (Select Active Status Display) command is issued.

14.2.1 Appearance of the Status Display

14.2.1 Appearance of the Status Display

When the Status Display is disabled, the line where it would appear shall blend into the display border. Guidelines: On a non-over-scanned non-windowed display, the area should appear

black; On an over-scanned non-windowed display, the area should appear the same as the screen background; On a windowed display, the size of the terminal window may change to include the status line. Visual side effects caused by enabling or disabling the Status Display should be minimized.

When the Indicator Status Line is enabled, it's background is the reverse of the Main Display background as indicated by the setting of Screen Mode (DECSCNM).

When the Host Writable Status Line is enabled, it has the same background as the main display. The reverse video character attribute may be used to cause text on the status line to appear in reverse video with respect to the main display. Screen Mode (DECSCNM) changes the background of both the status line and the main display.

If Column Mode (DECCOLM) is changed, the width of the Status Display changes. This will leave 52 blank spaces on the right hand side of the Indicator Status Line in 132 Column Mode. The spacing between indicators will not be increased in 132 Column Mode, however, the host will be able to place more than 80 characters into the Host Writable Status Line in 132 Column Mode.

14.2.1.1 Error Line
14.2.1.1 Error Line

The Error Line appears when it is necessary to indicate the result of some local terminal operation that would not otherwise be evident. An error in Set-Up for example, or a change in session management state. The text of the message replaces the currently displayed Status Line (Indicator or Host Writable) or appears if No Status Line is being displayed. The text of the message continues to be displayed until the user presses a key on the terminal keyboard, which restores the contents of the Status Display. The host may not directly program or affect the Error Line.

14.2.1.2 Indicator Status Line
14.2.1.2 Indicator Status Line

The Indicator Status Line shall contain a number of indicators reflecting the state of the terminal and options attached to the terminal (e.g. printer). The text of the indicators on the Indicator Status Line will be displayed in the Set-Up language. The indicators shall be placed in fixed positions on the Indicator Status Line, and there shall be at least one space (SP, 2/0) character between indicators for legibility.

Indicators are updated asynchronously, as the state of the feature they are monitoring changes. The interval between the change in

the state of a feature and the update of the appropriate indicator shall not exceed 80 milliseconds. Indicator changes are not queued: when an indicator is updated on the Status Display, it shall reflect the state of that feature at the time of the update.

The following indicators are required:

1. Active Session Indicator if the Session Management Extension is present and the indicator is not otherwise available. Indicates the session to which keyboard input is currently directed.
2. Active Position Indicator. Displays the current cursor position, row and column. Includes the page number if more than one page is available.
3. Graphics Input Position Indicator if some graphic input mode is enabled through ReGIS, Tektronix, or the Text Locator extension. Indicates the pixel row and column of the input position.
4. Edit Mode Indicator if the Local Editing Extension is present. Indicates whether Edit Mode (DECEDM) is enabled.
5. Insert-Replace Mode Indicator if the Local Editing Extension is present and the terminal is in Edit Mode. Indicates whether new characters typed into the Logical Display will be inserted, or overstrike existing characters.
6. Printer Status Indicator if the Printer Port Extension is present. Indicates the status of the attached printer if any, and the print mode.
7. Modem Status Indicator if modem control is supported and enabled. Indicates the status of the DSR signal on the communications port.

14.2.1.3 Implementation Guidelines for Indicators

14.2.1.3 Implementation Guidelines for Indicators

This subsection gives suggested formats for Indicators on the Indicator Status Line based on existing implementations. The VT340 Indicator Status Line might appear as follows (spacing is approximate):

```
2  1(24,008)  Edit  Overstrike Mode  Printer: None  Modem: DSR
```

The Active Session Indicator is a single digit at the left most edge of the status line indicating the session to which keyboard input is currently directed. Session 2 in the example above.

The Active Position Indicator gives the current page number within Page Memory followed by the cursor row and column number in parenthesis. Page 1, line 24, column 8 in the example above.

The Graphics Input Position Indicator is only visible when

graphics input is enabled. The indicator might appear as "[123,456]" pixel row 123, column 456.

The Edit Mode Indicator is only visible when DECEDM is active. The Indicator might appear as "Edit".

The Insert-Replace Mode Indicator toggles between "Overstrike Mode" or "Insert Mode". This indicator need only be displayed while Edit Mode is in effect.

The Printer Status Indicator shows the state of an attached printer if any (see Printer Port Extension). The Printer Status Indicator can take the following values:

Printer: None
Printer: Ready
Printer: Not Ready
Printer: Auto Print
Printer: Controller

The Modem Status Indicator shows the status of the DSR signal when Modem Control is selected on the communications port. This indicator is not displayed if Modem Controls are not selected on the communications port. The Modem Status Indicator can take the following values:

Modem: DSR
Modem: No DSR

14.2.2 Operation of Host Writable Status Line
14.2.2 Operation of Host Writable Status Line

If the Status Display is enabled to a Host Writable Status Line, the host is able to treat the Status Display as a separate Logical Display, generally 1 line by 80 or 132 columns.

To write text into the Host Writable Status Line, the host must issue the DECSASD (Select Active Status Display) command. A typical operation by the host would be to send a DECSASD command to switch to the Host Writable Status Line, send a line of text to the line, and then send another DECSASD command to return to the Main Display.

A separate ANSI state is maintained for the Host Writable Status Line. For example, the Status Line could have the USASCII invoked into G0, while the Main Display has the Line Drawing character set invoked into G0. When the DECSASD command is recognized, the terminal must switch and use the correct state information for the Host Writable Status Line or the Main Display. The following information is maintained separately for the Host Writable Status Line and the Main Display:

- Active Position
- Current Graphic Rendition (and other character attribute values)
- Origin Mode
- the currently designated G0, G1, G2, and G3 sets
- the currently invoked GL and GR sets
- the single shift (SS2, SS3) states
- the Auto-Wrap flag
- Cursor Save Buffer (DECSC and DECRC)

14.2.3 Effect of ANSI controls on the Status Display

14.2.3 Effect of ANSI controls on the Status Display

Several ANSI control functions affect both the Main Display and the Status Display. When the Host Writable Status Line is selected, as described above, any data received by the terminal is placed into the Status Display, and most ANSI control functions received will affect only the Status Display. There are some control functions which are executed differently, depending on whether the active position is in the Main or Status Display.

1. C1 Transmission
   The 7-bit or 8-bit transmission selection affects both the Main and Status Displays.

2. Character Set Selection
   An independent state is maintained for the invoked and designated character sets, as well as the associated shift states.

   The User-Preference Supplemental Character Set selection affects both the Main and Status Displays.

3. Cursor Position Controls
   If there is only one row in the Status Display, only the column parameters in cursor positioning control functions (e.g. CUP, etc.), or commands which affect areas dependent on character positions (e.g. ED, DCH, etc.) are significant. Horizontal cursor positioning commands (e.g. CUF, CUB) are recognized, but vertical cursor positioning commands (e.g. CUD, CUU) are ignored. The line number shall be reported as 1, when necessary.

4. DECANM - ANSI/VT52 Mode
   This sequence is ignored if received in the Status Display. If received in the Main Display, the contents of the Host Writable Status Line are left unchanged. If and when the VT52 Mode Enter ANSI Mode command is issued, the contents of the Host Writable Status Line are left unchanged.

5. DECCOLM - Colomn mode
DECCOLM affects both the Main Display and the Status Display simultaneously. The number of columns in the Host Writable Status Line cannot be independently controlled. Receipt of this sequence will clear both the Main Display and the Host Writable Status Line. The Status Display cursor is moved to the home position.

6. DECRQSS - Request Selection or Setting
The Status Display Type (DECSSDT) and Active Status Display (DECSASD) may be queried through DECRQSS.

7. DECSASD - Select Active Status Display
This control function is the recommended method to exit the Status Display.

8. DECSCL - Select Conformance Level
Select Conformance Level performs a Soft Terminal Reset (see DECSTR below).

9. DECSCLM - Scrolling Mode
This mode affects both the Main and Status displays.

10. DECSCNM - Screen Mode
The Main Display and the Status Line are affected by the setting of this Mode. The screen mode of the Host Writable Status line cannot be independently controlled.

11. DECSCPP - Set Columns Per Page
DECSCPP affects both the Main Display and the Status Display simultaneously. The number of columns in the Host Writable Status Line cannot be independently controlled.

12. DECSSDT - Select Status Display Type
If the Status Display Type is chosen to be something other than "Host Writable", the Status Display will be exited, and the selected Status Display Type will be imaged (either "Indicator" or "None").

13. DECSTR
The Status Display will be exited, but not erased.

14. DECTCEM - Text Cursor Enable Mode
Each of the Main and Status Display cursors may be individually enabled or disabled.

15. Format Effectors (BS, LF, VT, FF, IND, NEL, etc.)
Affect only the Status Display. If scrolling occurs as a result of a Format Effector, a single line Host Writable Status Line is erased.

16. IRM - Insert Replace Mode
    Insert/Replace Mode affects both the Main and Status displays.

17. Keyboard Functions
    All keyboard functions are executed regardless of the selected display (Main or Status).

18. Printer Functions
    All printer functions are executed regardless of the selected display (Main or Status).

19. RIS - Reset to Initial State
    The Status Display will be erased and exited.

20. Tabs
    Tabulation stops affect both the Main and Status displays.

14.2.4 Transmission of the Status Display

14.2.4 Transmission of the Status Display

The characters in the Status Display can be transmitted to the printer by "Print Composed Main Display". Print Page and Auto Print Mode will never transfer characters on the Status Display.

14.2.5 Behavior of the Status Display in VT100 (Level 1) and VT52 Mode

14.2.5 Behavior of the Status Display in VT100 (Level 1) and VT52 Mode

When VT100, or VT52 mode is entered, the Status Display will remain visible, if it was enabled. The sequences which program or change the type of the Status Display will not be recognized, therefore the type of Status Display may not be changed from the host in these modes. Sequences which globally affect the Status Display will still affect the Status Display, and the Status Display type may be changed from Set-Up.

The Indicator Status Line will remain visible in Set-Up mode.

14.3 CONTROL FUNCTIONS

14.3 CONTROL FUNCTIONS

SELECT ACTIVE STATUS DISPLAY DECSASD

Levels: 2x, 3, 4

Purpose: Select whether new characters are written to the Main Display or the Status Display.

Format:

CSI	Ps	$	}
9/11		2/4	7/13

Description: Causes the selection of the active status display, either the Main Display, or Status Display (also called the Host Writable or Remote Status Line). The display is specified by the selective parameter, Ps:

Ps	Display
0	Main Display (default)
1	Host Writable Status Line

Note that the Host Writable Status Line must be selected through DECSSDT for DECSASD to be able to select the Host Writable Status Line.

SELECT STATUS DISPLAY TYPE DECSSDT

Levels: 2x, 3, 4

Purpose: To select the type of status line to be displayed: indicator; host-writable; or none.

Format:

CSI	Ps	$	~
9/11		2/4	7/14

Description: This sequence allows the host to choose the type of Status Line that is displayed. The type of Status Display depends on the value of the selective parameter:

Ps	Status Line Type
0	No Status Line
1	Indicator (Local) Status Line
2	Host Writable (Remote) Status Line

Notes:

- If Ps is omitted, it is assumed to be 0, which will disable the Status Display.
- When the Status Line Type is changed, it is initialized. Specifically, this means that the Host Writable (Remote) Status Line is cleared, and the Indicator (Local) Status Line is set to the Set-Up indicators, and will show the current status of the terminal.
- This control has no effect on the Status Line displayed in Set-Up Mode; the Indicator Status Line is always displayed in Set-Up Mode.
- The type of Status Line can also be changed through a Set-Up field.
- The Status Display type may not be "locked" as a User-Preference Feature.

14.4 CHANGE HISTORY

14.4 CHANGE HISTORY

14.4.1 Original draft to Rev AX01

14.4.1 Original draft to Rev AX01

1. Added Abstract and Introduction.

2. Status Display is an extension to Level 2. DECSSDT and DECSASD do not work in Level 1 (VT100 mode).

3. Listed all sequences which affect status line including DECCOLM and DECSCPP.

VIDEO SYSTEMS REFERENCE MANUAL

VT52 Emulation

Document Identifier: A-DS-EL00070-0A-0 Rev. AX11, 18-Mar-1985

ABSTRACT: This document is an architectural statement of future terminal support for VT52 emulation. It includes the features of VT52 mode as documented in the VT102 User Guide. It does not include features of the VT100 family implementation which are a result of cross-over with ANSI mode and are not documented. These features should be considered undefined conditions in terminal products, and cannot be relied on by software. Furthermore, the architecture does not include all of the features of the actual VT52 products, because the intention of the architecture is to define the level of support necessary to maintain software products which already run compatibly on the entire DEC terminal product set.

APPLICABILITY: SARA Formal Cross-Component Standard. Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria".

STATUS: FOR REVIEW ONLY

This document has been placed in the SARA "Formal Cross-Component Standard" category. The material contained within this document is assumed to define mandatory standards unless it is clearly marked as (a) not mandatory or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements. The procedures for modifying or evolving this standard are contained within the contents of this document.

FOR INTERNAL USE ONLY

TITLE: VIDEO SYSTEMS - VT52 Emulation

DOCUMENT IDENTIFIER: A-DS-EL00070-0A-0 Rev. AX11, 18-Mar-1985

REVISION HISTORY: Original Draft 22-Oct-1982
Revision 0.1 16-Nov-1982
Revision 0.2 19-Nov-1982
Revision 0.3 25-Dec-1982
Revision 1.0 03-Feb-1983
Revision AX01 28-Feb-1983
Revision AX10 15-May-1983

FILES: User Documentation EL070SA.mem

Internal Documentation EL070SA.rno
EL070SA.rnt
EL070SA.rnx

Document Management Group: Terminal Interface Architecture
Responsible Department: Video Architecture Group
Responsible Individual: Peter Sichel

ACCEPTANCE: This document has been approved by the Manager of the Video Architecture Group based on a comprehensive review of its individual sections by the members of the SARA component groups working Terminal Interface Architecture issues. The list of individuals on the review and approval list are on file in Standards and Methods Control.

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, RANI::VIDARCH

Copies of this document can be ordered from:

Standards and Methods Control
APO1/F7, DTN 289-1414, JOKUR::SIMONETTI

CONTENTS

A.1 INTRODUCTION

A.1.1 Purpose

VT52 Emulation defines a display service class for character cell display operations. The emulation defined here is a consistent and compatible subset of the functions provided by the actual VT52 family of DEC terminals and VT52 emulation mode as documented in the VT102 User Guide. It does not include features of the VT100 family implementation which are a result of cross-over with ANSI mode and are not documented. These features should be considered undefined conditions in terminal products, and cannot be relied on by software. Furthermore, the architecture does not include all of the features of the actual VT52 products, because the intention of the architecture is to define the level of support necessary to maintain software products which already run compatibly on the entire DEC terminal product set.

A.1.2 Scope

This specification will be included as an appendix to the Video Systems Reference Manual (Video SRM). As an appendix, it is not considered a formal part of the architecture, and is not a required part of conforming product implementations. DEC products which implement this mode, however, should do so in accordance with this specification to insure compatibility with existing software. Software conforming to the architecture will not use any of the functions defined in this appendix.

NOTE

> The algorithms defined in this appendix apply only to the display logic which exists when VT52 emulation is entered correctly (i.e., all UNDEFINED conditions are preset to their prescribed values). For example, VT52 emulation assumes that tab stops are preset to modulus 8 intervals. If this is not done, the logic as described here is not guaranteed.

A.2 STATE DESCRIPTIONS

A.2.1 Display

The display logic of the VT52 emulator provides the following features:

- Character cell array with 80 columns X 24 lines
- Home (1,1) at the top left corner
- No character attributes
- No line attributes
- No screen attributes
- ASCII character set
- Line Drawing character set (* extension *)
- Tabs at 8 column intervals.

```
LINE_TYPE   = 1..MAX_NUM_LINES;   (* 1 to 24 lines *)
COLUMN_TYPE = 1..MAX_NUM_COLUMNS; (* 1 to 80 columns *)

(* character = character code, character set *)

CHARACTER_TYPE = RECORD
  CODE: CHARACTER_CODE_TYPE;
  CHARACTER_SET:
    GRAPHIC_CHARACTER_SET_TYPE; (* extension *)
  END;
```

The character set selection is an extension because the VT52 and other terminals do special graphics differently.

```
(* display = two-dimensional array of 24 lines by 80 columns *)

DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
```

A.2.2 Active Position

The device maintains a position register known as the "Active Position", which serves as a reference point for graphic character replacement and control functions.

```
CHARACTER_POSITION_TYPE = RECORD
  LINE: LINE_TYPE;
  COLUMN: COLUMN_TYPE;
  END;

ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

A.2.3 Tab Stops, Fixed

The device provides a set of horizontal tabulation stops whose positions are fixed in hardware and cannot be changed.

```
HORIZONTAL_TAB_STOPS: ARRAY [COLUMN_TYPE] OF BOOLEAN;
```

tab stops = columns 9,17,25,33,41,49,57,65,73

A.3 PARAMETERS AND CONSTANTS

This section defines the default values and ranges applicable to the terminal state information.

```
(* Display Parameters *)
        MAX_NUM_LINES = 24;
        MAX_NUM_COLUMNS = 80;
(*
  Note: an empty character is a character position in the
  logical display in which no character code appears, and
  thus no character is rendered in the visual display for
  this position.
*)
        EMPTY_CHARACTER = 0;

TYPE    LINE_TYPE = 1..MAX_NUM_LINES;
        COLUMN_TYPE = 1..MAX_NUM_COLUMNS;
        CHARACTER_POSITION_TYPE = RECORD
          LINE: LINE_TYPE;
          COLUMN: COLUMN_TYPE;
          END;
        CHARACTER_CODE_TYPE = 0..127;
        GRAPHIC_CHARACTER_SET_TYPE = (ASCII,LINE_DRAWING);
        CHARACTER_TYPE = RECORD
          CODE: CHARACTER_CODE_TYPE;
          CHARACTER_SET:
            GRAPHIC_CHARACTER_SET_TYPE; (* extension *)
          END;
```

A.4 SUMMARY OF STATE VARIABLES

The following values represent the state information necessary to maintain a VT52 Character Cell Display.

```
VAR     ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
        CURRENT_SET: GRAPHIC_CHARACTER_SET_TYPE;
        KEYPAD_MODE: (NUMERIC,APPLICATION);
        EMULATION_MODE: (VT52_OFF,VT52_ON);
```

A.5 COMMAND SUMMARY

A.5.1 Required Commands

This is a list of the VT52 commands supported in the architectural specification:

Identify
Enter VT52 Emulation Mode
Exit VT52 Emulation Mode
Enter Alternate Keypad mode
Exit Alternate Keypad mode
Enter Graphics Mode
Exit Graphics Mode
Cursor Up
Cursor Down
Cursor Right
Cursor Left
Cursor to Home
Direct Cursor Address
Backspace
Horizontal Tab
Line Feed
Carriage Return
Reverse Line Feed
Erase to End of Line
Erase to End of Screen

A.5.2 Character Set Extension

This command is in an extension to the VT52 specification because in the VT52 the character set is VT52 Special Graphics and in the VT100 and all future terminals it is the Line Drawing set.

Select Special Graphics Set

A.5.3 VT52 Printer Port Extension

The following commands are in an extension to the VT52 architectural specification because they are not included in the standard VT52 (they are in the VT52 with printer option):

Print Screen
Enter Auto Print Mode
Exit Auto Print Mode

A.5.4 ANSI Printer Port Commands

The following commands control modes that are not in the VT52 with printer port but are carry-overs from VT100 printer port modes.

Print Cursor Line

Enter Print Controller Mode
Exit Print Controller Mode

A.5.5 Undefined Functions

The following modes and commands have undefined results and states both when entering and exiting VT52 Mode emulation.

Insert/Replacement Mode
Printer Extent Mode
Printer Form Feed Mode
New Line Mode
Origin Mode
Column Mode
Screen Mode
Keyboard Action Mode
Cursor Key Mode
Line Attributes
Character Attributes
Character Sets (The VT100 VT52 emulator special graphics and) (the VT52 special graphics are not the same.)
Scrolling Region
Scrolling Mode
Auto Wrap Mode
Auto Repeat Mode
Enter Hold-Screen Mode (VT52 function not emulated)
Exit Hold-Screen Mode (VT52 function not emulated)
Tabs

NOTE

The VT52 has hardware set tabs. The VT52 emulator uses the tabs that were set in the terminal before entering the emulator. For correct use of the VT52 emulator, application software should set the tabs to the VT52 settings before entering the emulator.

A.5.6 Programming Guidelines

Because of the undefined interactions listed above, the terminal should be switched between VT52 emulation and ANSI mode only at program initialization. Frequent switching between modes will leave the terminal in an undefined state.

A.6 GRAPHIC CHARACTER REPLACEMENT

REPLACE GRAPHIC CHARACTER

Purpose: To place a graphic character in the display at the Active Position.

Format: implied on receipt of graphic character data

Description: This procedure is called on receipt of graphic character data to place the data in the appropriate position in the display structure. The character code and the current character set are stored in the display list. The Active Position is advanced by one column each time this operation is performed.

State Affected:

```
DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE REPLACE_GRAPHIC_CHARACTER
VAR       X: COLUMN_TYPE;
BEGIN
DISPLAY[ACTIVE_POSITION.LINE,ACTIVE_POSITION.COLUMN]
  .CODE:= EVENT.CODE_VALUE;
(* character set extension only *)
DISPLAY[ACTIVE_POSITION.LINE,ACTIVE_POSITION.COLUMN]
  .CHARACTER_SET:= CURRENT_SET;
IF ACTIVE_POSITION.COLUMN < MAX_NUM_COLUMNS THEN
  ACTIVE_POSITION_COLUMN:= ACTIVE_POSITION_COLUMN + 1;
END;
```

Known Deviations: None

A.7 CONTROL FUNCTIONS

IDENTIFY

Purpose: Request terminal type identification.

Format:

ESC	Z
1/11	5/10

Description: The Identify command is a request from the computer for an identifying sequence. The VT52 emulator response is ESC / Z (1/11 2/15 5/10) (same as VT52 terminal).

Notes:

1. Most terminals also respond to this request when they are in Level 1 or Level 2 operation. Then they respond with an ANSI Device Attributes sequence that includes a DEC private registered value that indicates the actual terminal type. Conforming software should not rely on this response to the DECID sequence. (See IDENTIFY in the chapter "Terminal Management".)

State Affected: None

Algorithm:

```
PROCEDURE IDENTIFY;
BEGIN
IF EMULATION_MODE = VT52_ON THEN
  WRITE (* to host *) (ESC,'/Z');
END;
```

Known Deviations: None

ENTER VT52 EMULATION MODE

Purpose: Disable ANSI controls and turn on VT52 emulator.

Format:	CSI	?	2	1
	9/11	3/15	3/2	6/12

Description: The Enter VT52 Emulation Mode control disables further recognition of ANSI standard control functions in the terminal and turns on the VT52 emulator. This control is only recognized in Level 1 and Level 2 Character Cell Display operation.

Notes:

1. Execution of this control from Level 1 or Level 2 operation does not cause the physical state of the device to change (i.e., the display is not cleared, and the Active Position does not move, etc.).

2. In Conformance Levels 1 and 2, most VT52 control sequences are errors (except Identify and Exit VT52 Emulation Mode), and in general are ignored. They should not be sent by conforming software.

3. This sequence will not be treated as a control function when the terminal is in VT52 mode, and the effect on the display will be unpredictable, since it is dependent on the transmission environment.

State Affected:

```
EMULATION_MODE = (VT52_OFF,VT52_ON);
```

Algorithm:

```
PROCEDURE ENTER_VT52_EMULATION_MODE;
EMULATION_MODE:= VT52_ON;
END;
```

Known Deviations: None

EXIT VT52 EMULATION MODE

Purpose: Turn off VT52 emulator and allow ANSI controls to work.

Format:

ESC	<
1/11	3/12

Description: The Exit VT52 Emulation Mode control turns off the VT52 emulator and turns on the ANSI parser of the terminal, causing the terminal to return to Level 1 operation.

Notes:

1. In Conformance Levels 1 and 2, most VT52 control sequences are errors (except Identify and Exit VT52 Emulation Mode), and in general are ignored. They should not be sent by conforming software.

State Affected:

```
EMULATION_MODE = (VT52_OFF,VT52_ON);
```

Algorithm:

```
PROCEDURE EXIT_VT52_EMULATION_MODE;
EMULATION_MODE:= VT52_OFF;
END;
```

Known Deviations: None

ENTER ALTERNATE KEYPAD MODE

Purpose: Auxiliary keypad keys generate application sequences.

Format:

ESC	=
1/11	3/13

Description: The auxiliary keypad keys generate sequences used by application programs.

Notes:

1. The codes produced by the keypad in VT52 mode are shown in Table 1.

State Affected:

```
KEYPAD_MODE: (NUMERIC,APPLICATION);
```

Algorithm:

```
PROCEDURE ENTER_KEYPAD_MODE;
KEYPAD_MODE:= APPLICATION;
END;
```

Known Deviations:

The minus, comma, and PF4 keys are not present on the VT52 keyboard.

In VT52 Emulation Mode, the VT125 does not transmit a code for the PF1 key until a second key is pressed.

Table 1 VT52 Keypad Codes

Key	Alternate Keypad Mode Off (Numeric Keypad Mode)	Alternate Keypad Mode On (Application Keypad Mode)	
0	0 060	ESC ? p 033 077 160	
1	1 061	ESC ? q 033 077 161	
2	2 062	ESC ? r 033 077 162	
3	3 063	ESC ? s 033 077 163	
4	4 064	ESC ? t 033 077 164	
5	5 065	ESC ? u 033 077 165	
6	6 066	ESC ? v 033 077 166	
7	7 067	ESC ? w 033 077 167	
8	8 070	ESC ? x 033 077 170	
9	9 071	ESC ? y 033 077 171	
-(minus)	-(minus) 055	ESC ? m 033 077 155	**
,(comma)	,(comma) 054	ESC ? l 033 077 154	**
.(period)	.(period) 056	ESC ? n 033 077 156	
ENTER*	CR CR LF 015 or 015 012	ESC ? M 033 077 115	

(continued)

Table 1 VT52 Keypad Codes (continued)

Key	Alternate Keypad Mode Off (Numeric Keypad Mode)	Alternate Keypad Mode On (Application Keypad Mode)	
PF1	ESC P 033 120	ESC P 033 120	
PF2	ESC Q 033 121	ESC Q 033 121	
PF3	ESC R 033 122	ESC R 033 122	
PF4	ESC S 033 123	ESC S 033 123	**

* When numeric keypad mode is select (alternate keypad mode is off), the ENTER key generates the same characters as the RETURN key. The RETURN key character code may be affected by the setting of New Line Mode. When off, this mode causes the key to generate a single control character (CR, octal 015). When on, this mode causes the key to generate two characters (CR, octal 015 and LF, octal 012). This interaction is UNDEFINED by the architecture, and should not be depended upon by conforming software.

** These sequences are not generated by the VT52.

EXIT ALTERNATE KEYPAD MODE

Purpose: Keypad generates codes for the characters shown on the keys.

Format: ESC >

1/11 3/14

Description: Auxiliary keypad keys generate characters corresponding to the numeric, comma, period, and minus sign keys on the main keyboard.

Notes:

1. See Table 1 for the codes.

State Affected:

```
KEYPAD_MODE: (NUMERIC,APPLICATION);
```

Algorithm:

```
PROCEDURE EXIT_KEYPAD_MODE;
KEYPAD_MODE:= NUMERIC;
END;
```

Known Deviations:

The minus, comma, and PF4 keys are not present on the VT52 keyboard.

ENTER GRAPHICS MODE

Purpose: Display special graphics instead of lower case ASCII.

Format:

ESC	F
1/11	4/6

Description: Selects the special character and line drawing character set.

Notes:

1. Table 2 lists the characters displayed and their codes, and compares the characters with the VT52 special graphics set.

State Affected:

```
CURRENT_SET: GRAPHIC_CHARACTER_SET_TYPE;
```

Algorithm:

```
PROCEDURE ENTER_GRAPHICS_MODE;
CURRENT_SET:= LINE_DRAWING;
END;
```

Known Deviations:

The character set displayed is not the same as the set that was in the VT52. However, all terminals since the VT52 have the same VT100 special graphics set.

Table 2 Special Character and Line Drawing Set and Graphics Mode Comparison

Row/ Column	US or UK Set	Special Character and Line Drawing Set	In Graphics Mode (VT52 only)
5/15	_	Blank	blank
6/0	'	<> Diamond	reserved
6/1	a	blotch (error indicator)	solid rectangle
6/2	b	ht Horizontal tab	1/
6/3	c	ff Form feed	3/
6/4	d	cr Carriage return	5/
6/5	e	lf Line feed	7/
6/6	f	o Degree symbol	degrees
6/7	g	+- Plus/minus	plus or minus
6/8	h	nl New line	right arrow
6/9	i	vt Vertical tab	ellipsis (Dots)
6/10	j	_| Lower-right corner	divide by
6/11	k	~| Upper-right corner	down arrow
6/12	l	|~ Upper-left corner	bar at scan 0
6/13	m	L Lower-left corner	bar at scan 1
6/14	n	+ Crossing lines	bar at scan 2
6/15	o	- Horizontal line - Scan 1	bar at scan 3
7/0	p	- Horizontal line - Scan 3	bar at scan 4
7/1	q	- Horizontal line - Scan 5	bar at scan 5
7/2	r	- Horizontal line - Scan 7	bar at scan 6
7/3	s	- Horizontal line - Scan 9	bar at scan 7
7/4	t	|- Left "T"	subscript 0
7/5	u	-| Right "T"	subscript 1
7/6	v	_|_ Bottom "T"	subscript 2
7/7	w	T Top "T"	subscript 3
7/8	x	| Vertical bar	subscript 4
7/9	y	<= Less than or equal to	subscript 5
7/10	z	=> Greater than or equal to	subscript 6
7/11	{	i-i Pi	subscript 7
7/12	|	/= Not equal to	subscript 8
7/13	}	L- UK pound sign	subscript 9
7/14	~	. Centered dot	paragraph

EXIT GRAPHICS MODE

Purpose: Display ASCII lower case characters.

Format: ESC G

1/11 4/7

Description: Select the ASCII character set.

Notes:

1. Many terminals have included a "UK" character set, selectable in SET-UP. The Exit Graphics Mode command returns to the UK set if it was selected.

State Affected:

```
CURRENT_SET: GRAPHIC_CHARACTER_SET_TYPE;
```

Algorithm:

```
PROCEDURE EXIT_GRAPHICS_MODE;
CURRENT_SET:= ASCII;
END;
```

Known Deviations: None

CURSOR UP

Purpose: Move the Active Position upward by one line.

Format: ESC A

1/11 4/1

Description: The Cursor Up control moves the Active Position upward by one line in the display.

Notes:

1. The Active Position will not move above the first line of the display.
2. No scrolling will occur as a result of this control.
3. This is the same sequence that the up arrow key generates.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE CURSOR_UP;
BEGIN
IF ACTIVE_POSITION.LINE > 1 THEN
  ACTIVE_POSITION.LINE:= ACTIVE_POSITION.LINE - 1
ELSE (* ignore *);
END;
```

Known Deviations: None

CURSOR DOWN

Purpose: Move the Active Position down by one line.

Format:

ESC	B
1/11	4/2

Description: The Cursor Down control moves the Active Position down by one line in the display.

Notes:

1. The Active Position will not move below the bottom line of the display.
2. No scrolling will occur as a result of this control.
3. This is the same sequence that the down arrow key generates.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE CURSOR_DOWN;
BEGIN
IF ACTIVE_POSITION.LINE + 1 <= MAX_NUM_LINES THEN
  ACTIVE_POSITION.LINE:= ACTIVE_POSITION.LINE + 1
ELSE (* ignore *);
END;
```

Known Deviations: None

CURSOR RIGHT

Purpose: Move the Active Position right by one character position.

Format:

	ESC	C
	1/11	4/3

Description: The Cursor Right control moves the Active Position right by one character position.

Notes:

1. The Active Position will not move beyond the right margin of the screen.
2. This is the same sequence that the right arrow key generates.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE CURSOR_FORWARD;
BEGIN
IF ACTIVE_POSITION.COLUMN < MAX_NUM_COLUMNS THEN
  ACTIVE_POSITION.COLUMN:= ACTIVE_POSITION.COLUMN + 1
ELSE (* ignore *);
END;
```

Known Deviations: None

CURSOR LEFT

Purpose: Move the Active Position left by one character position.

Format:

ESC	D
1/11	4/4

Description: The Cursor Left control moves the Active Position left by one character position.

Notes:

1. The Active Position will not move beyond the left margin of the display.
2. This is the same sequence that the left arrow key generates.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE CURSOR_BACKWARD;
BEGIN
IF ACTIVE_POSITION.COLUMN > 1 THEN
  ACTIVE_POSITION.COLUMN:= ACTIVE_POSITION.COLUMN - 1
ELSE (* ignore *);
END;
```

Known Deviations: None

CURSOR HOME

Purpose: Move the Active Position to the home position.

Format:

ESC	H
1/11	4/8

Description: The Cursor Home control moves the Active Position to the home position at line 1, column 1.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE CURSOR_HOME;
BEGIN
ACTIVE_POSITION.LINE:= 1;
ACTIVE_POSITION.COLUMN:= 1;
END;
```

Known Deviations: None

DIRECT CURSOR ADDRESS

Purpose: Move the Active Position to the specified position.

Format:	ESC	Y	line	column
	1/11	5/9	line	column

Description: The Direct Cursor Address control moves the active position to the specified line and column. The line and column numbers are sent as the ASCII characters whose codes are the decimal numbers of the line or column plus 31. For example, decimal 32 (SPACE, 2/0) refers to the first line or column, and decimal 111 (o, 6/15) refers to the eightieth column.

Notes:

1. This command has range limits of 1 to 24 lines with parameters "SPACE" (2/0) to "7" (3/7), and 1 to 80 columns with parameters "SPACE" (2/0) to "o" (6/15). An out of range parameter in either dimension causes no action in that dimension, but if the other parameter is in range it will still be operated on.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE CURSOR_POSITION (LINE_VALUE,COLUMN_VALUE: CHAR);
VAR       Y: LINE_TYPE;
          X: COLUMN_TYPE;
BEGIN
(* convert ascii characters to position values *)
Y:= ORD(LINE_VALUE) - 31;
X:= ORD(COLUMN_VALUE) - 31;
(* conditionally update line *)
IF ((Y >= 1) AND (Y <= MAX_NUM_LINES)) THEN
  ACTIVE_POSITION.LINE:= Y;
IF ((X >= 1) AND (X <= MAX_NUM_COLUMNS)) THEN
  ACTIVE_POSITION.COLUMN:= X;
END;
```

Known Deviations: None

BACK SPACE BS

Purpose: Move the Active Position one column to the left.

Format: BS

0/8

Description: The BS control moves the Active Position backward in the display by one column.

Notes:

1. The Active Position will not move beyond the beginning of the Active Line.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE BACK_SPACE;
BEGIN
IF ACTIVE_POSITION.COLUMN > 1 THEN
  ACTIVE_POSITION.COLUMN:= ACTIVE_POSITION.COLUMN - 1
ELSE (* ignore *);
END;
```

Known Deviations: None

HORIZONTAL TAB HT

Purpose: Advance the Active Position to the next Horizontal Tab Stop.

Format: HT

0/9

Description: The HT control moves the Active Position forward in the display to the next Horizontal Tab Stop in the Active Line.

Notes:

1. If no tabulation stop is reached before the end (last column) of the Active Line, the Active Position will be set to the end of the Active Line.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE HORIZONTAL_TAB;
BEGIN
REPEAT
  IF ACTIVE_POSITION.COLUMN < MAX_NUM_COLUMNS THEN
    ACTIVE_POSITION.COLUMN:= ACTIVE_POSITION.COLUMN + 1;
UNTIL (HORIZONTAL_TAB_STOPS[ACTIVE_POSITION.COLUMN])
  OR (ACTIVE_POSITION.COLUMN = MAX_NUM_COLUMNS);
END;
```

Known Deviations: None

LINE FEED LF

Purpose: Move the Active Position downward one line, scrolling if necessary.

Format: LF

0/10

Description: The LF control moves the Active Position downward in the display by one line. If the Active Position is already at the Bottom Margin the display will scroll upward by one line.

State Affected:

```
DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE LINE_FEED;
VAR       Y: LINE_TYPE;
          X: COLUMN_TYPE;
BEGIN
IF ACTIVE_POSITION.LINE < MAX_NUM_LINES THEN
  ACTIVE_POSITION.LINE:= ACTIVE_POSITION.LINE+1
ELSE (* scroll up *)
  BEGIN
  FOR Y:= 1 TO MAX_NUM_LINES - 1 DO
    FOR X:= 1 TO MAX_NUM_COLUMNS DO
      DISPLAY[Y,X]:= DISPLAY[Y+1,X];
  FOR X:= 1 TO MAX_NUM_COLUMNS DO
    DISPLAY[MAX_NUM_LINES,X].CODE:= EMPTY_CHARACTER;
  END;
END;
```

Known Deviations: None

CARRIAGE RETURN CR

Purpose: Move the Active Position to the first column of the Active Line.

Format: CR

0/13

Description: The CR control moves the Active Position to the first column in the Active Line.

State Affected:

```
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE CARRIAGE_RETURN;
BEGIN
ACTIVE_POSITION.COLUMN:= 1;
END;
```

Known Deviations: None

REVERSE LINE FEED

Purpose: Move the Active Position upward one line in the same column, scrolling if necessary.

Format:

	ESC	I
	1/11	4/9

Description: The Reverse Line Feed control moves the Active Position upward by one line in the same column in the display. If the Active Position is at the first line in the display, a scroll down is performed.

State Affected:

```
DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
```

Algorithm:

```
PROCEDURE REVERSE_LINE_FEED;
VAR       Y: LINE_TYPE;
          X: COLUMN_TYPE;
BEGIN
IF ACTIVE_POSITION.LINE > 1 THEN
  ACTIVE_POSITION.LINE:= ACTIVE_POSITION.LINE - 1
ELSE (* scroll down *)
  BEGIN
  FOR Y:= MAX_NUM_LINES DOWNTO 2 DO
    FOR X:= 1 TO MAX_NUM_COLUMNS DO
      DISPLAY[Y,X]:= DISPLAY[Y-1,X];
  FOR X := 1 TO MAX_NUM_COLUMNS DO
    DISPLAY[1,X].CODE := EMPTY_CHARACTER;
  END;
END;
```

Known Deviations: None

ERASE TO END OF LINE

Purpose: Erase all characters from the Active Position to the end of the line.

Format: ESC K

1/11 4/11

Description: The Erase to End of Line control erases all characters from the Active Position to the end of the current line including the cursor position. The cursor does not move.

Notes:

1. The Active Position must be placed on the first position of the line to erase the complete line.

State Affected:

```
DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
```

Algorithm:

```
PROCEDURE ERASE_TO_END_OF_LINE;
VAR       X: COLUMN_TYPE;
BEGIN
FOR X:= ACTIVE_POSITION.COLUMN TO MAX_NUM_COLUMNS DO
  DISPLAY[ACTIVE_POSITION.LINE,X].CODE:= EMPTY_CHARACTER;
END;
```

Known Deviations: None

ERASE TO END OF SCREEN

Purpose: Erase all characters from the Active Position to the end of the screen.

Format: ESC J

1/11 4/10

Description: The Erase to End of Screen control erases all characters from the Active Position to the end of the screen including the cursor position. The cursor does not move.

Notes:

1. The Active Position must be placed on the first position of the screen to erase the complete screen.

State Affected:

```
DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
```

Algorithm:

```
PROCEDURE ERASE_TO_END_OF_DISPLAY;
VAR       Y: LINE_TYPE;
          X: COLUMN_TYPE;
BEGIN
FOR X:= ACTIVE_POSITION.COLUMN TO MAX_NUM_COLUMNS DO
  DISPLAY[ACTIVE_POSITION.LINE,X].CODE:= EMPTY_CHARACTER;
FOR Y:= (ACTIVE_POSITION.LINE + 1) TO MAX_NUM_LINES DO
  BEGIN
  FOR X:= 1 TO MAX_NUM_COLUMNS DO
  DISPLAY[Y,X].CODE:= EMPTY_CHARACTER;
  END;
END;
```

Known Deviations: None

A.8 EXTENSIONS TO THE VT52 EMULATION ARCHITECTURE

The following functions are not part of the architecture because many terminals, including the VT52 without printer, do not support them. However, all new terminals with both VT52 emulation and printer ports support these commands.

A.8.1 Print Functions

Some terminals have serial printer ports to allow local printing. All print operations can be selected using sequences. Two of the print operations can be selected from the terminal keyboard (auto print and print screen).

When printing characters from the screen of the terminal, the tab stops of the terminal and printer are ignored. Characters printed are spaced using the space (SP, octal 040) character. After the last printable character of a line is transmitted, a carriage return (CR, 015) and line feed (LF, 012) are transmitted. The terminal does not transmit the space characters after the last printable character of a line.

If a line contains double height characters, the characters are printed as two identical lines of standard width characters. Double width characters are printed as standard width characters on a single line.

A.8.1.1 Modes

```
AUTO_PRINT_MODE: (AUTO_PRINT_OFF,AUTO_PRINT_ON);
PRINTER_CONTROLLER_MODE:
  (PRINTER_CONTROLLER_OFF,PRINTER_CONTROLLER_ON);
```

ENTER AUTO PRINT MODE

Purpose: Turn on Auto Print Mode.

Format: ESC ^

1/11 5/14

Description: Turns on Auto Print Mode. A line on the screen is printed when the cursor is moved off the line. The cursor is moved off the line by a Line Feed, Form Feed, or Vertical Tab character, or when a character is received when the cursor is at the right margin and the auto wrap feature is selected.

Notes:

1. If autowrap is selected in SET-UP, it applies even in VT52 mode.

State Affected:

```
AUTO_PRINT_MODE: (AUTO_PRINT_OFF,AUTO_PRINT_ON);
```

Algorithm:

```
PROCEDURE ENTER_AUTO_PRINT_MODE;
AUTO_PRINT_MODE:= AUTO_PRINT_ON;
END;
```

Known Deviations: None

EXIT AUTO PRINT MODE

Purpose: Turn off Auto Print Mode.

Format: ESC

1/11 5/15

Description: Turns off Auto Print Mode (see Enter Auto Print Mode).

State Affected:

```
AUTO_PRINT_MODE: (AUTO_PRINT-OFF,AUTO_PRINT_ON);
```

Algorithm:

```
PROCEDURE EXIT_AUTO_PRINT_MODE;
AUTO_PRINT_MODE:= AUTO_PRINT_OFF;
END;
```

Known Deviations: None

ENTER PRINTER CONTROLLER MODE

Purpose: Turn on Printer Controller Mode.

Format: ESC W

1/11 5/7

Description: Turns on Printer Controller Mode. All characters received by the terminal are printed by the serial printer without being displayed on the screen. The characters received are printed without being displayed or altered in any way. The terminal does not insert or delete spaces, provide line delimiters or select the proper character set within the printer.

State Affected:

```
PRINTER_CONTROLLER_MODE:
  (PRINTER_CONTROLLER_OFF,PRINTER_CONTROLLER_ON);
```

Algorithm:

```
PROCEDURE ENTER_PRINTER_CONTROLLER_MODE;
PRINTER_CONTROLLER_MODE:= PRINTER_CONTROLLER_ON;
END;
```

Known Deviations: None

EXIT PRINTER CONTROLLER MODE

Purpose: Turn off printer controller.

Format: ESC X

1/11 5/8

Description: Turns off Printer Controller Mode (see Enter Printer Controller Mode).

State Affeeted:

```
PRINTER_CONTROLLER_MODE:
  (PRINTER_CONTROLLER_OFF,PRINTER_CONTROLLER_ON);
```

Algorithm:

```
PROCEDURE EXIT_PRINTER_CONTROLLER_MODE;
PRINTER_CONTROLLER_MODE:= PRINTER_CONTROLLER_OFF;
END;
```

Known Deviations: None

PRINT CURSOR LINE

Purpose: Print the line containing the Active Position.

Format:

	ESC	V
	1/11	5/6

Description: Causes the serial printer to print the line on the screen containing the cursor. The cursor position is not changed. The print cursor line operation ends when the line is printed.

Notes:

1. This operation can be selected from the keyboard of the VT52 and the VT102.

State Affected: None

Known Deviations: None

PRINT SCREEN

Purpose: Print the contents of the screen.

Format: ESC]

1/11 5/13

Description: Causes the serial printer to print the contents of the screen. The print screen operation ends when the screen is printed.

Notes:

1. This operation can be selected from the keyboard of the VT52 and the VT102.

State Affected: None

Known Deviations: None

A.9 CHANGE HISTORY

A.9.1 Revision 0.3 To Rev 1.0

1. Added the notes in the Abstract to the Purpose section in the Introduction.

2. Added note to Scope indicating that conforming software will not use VT52 Emulation Mode.

3. Added note to Scope indicating that all algorithms in this appendix are dependent on presetting of conditions before entering VT52 Emulation Mode.

4. Removed the reference to ASCII as an "extension" in the list of display features, and added the Line Drawing set as an extension.

5. Corrected the display size from 132 columns to 80 columns.

6. In note on Undefined Functions, changed "software" to "applications software".

7. In Enter and Exit VT52 Emulation Mode, changed descriptions and algorithms to allow entry from any conformance level, and to exit to same level entered from.

8. Added note to Enter VT52 Emulation Mode to indicate that the physical state of the terminal should not change.

9. Added note to Enter VT52 Emulation Mode to indicate that there is no action when already in VT52 Mode.

10. Added a deviation note to Enter Alternate Keypad Mode for the VT125.

11. Cleaned up the note on the effect of New Line Mode on the Enter key.

12. Corrected the description and the algorithm for Direct Cursor

Position to indicate that one legal coordinate will be operated
on.

13. Removed the note on the Active Position from the description
of Line Feed.

A.9.2 Rev AX10 To AX11

1. Changed description of EXIT VT52 MODE to indicate that it always returns to level 1 operation.

Section Index

VIDEO SYSTEMS REFERENCE MANUAL

Programmer's Guide

Document Identifier: A-DS-EL00070-0B-0 Rev. AX10, 15-May-1983

ABSTRACT: This appendix contains information of general interest to software engineers designing software interfacing with products designed in in conformance with the Video Systems Reference Manual.

APPLICABILITY: SARA Formal Cross-Component Standard. Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria".

STATUS: FOR REVIEW ONLY

+----------+
| This document has been placed in the SARA "Formal Cross-Component Standard" category. The material contained within this document is assumed to define mandatory standards unless it is clearly marked as (a) not mandatory or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements. The procedures for modifying or evolving this standard are contained within the contents of this document. |
+----------+

+----------+
| FOR INTERNAL USE ONLY |
+----------+

TITLE: VIDEO SYSTEMS - Programmer's Reference

DOCUMENT IDENTIFIER: A-DS-EL00070-0B-0 Rev. AX10, 15-April-1983

REVISION HISTORY: Original Draft 25-Dec-82
Revision 0.1 12-Jan-83
Revision AX01 28-Feb-83

FILES: User Documentation EL070SB.mem

Internal Documentation EL070SB.rno
EL070SB.rnt
EL070SB.rnx

Document Management Group: Terminal Interface Architecture
Responsible Department: Video Architecture Group
Responsible Individual: Peter Sichel

ACCEPTANCE: This document has been approved by the Manager of the Video Architecture Group based on a comprehensive review of its individual sections by the members of the SARA component groups working Terminal Interface Architecture issues. The list of individuals on the review and approval list are on file in Standards and Methods Control.

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, RANI::VIDARCH

Copies of this document can be ordered from:

Standards and Methods Control
APO1/F7, DTN 289-1414, JOKUR::SIMONETTI

CONTENTS

B.1 INTRODUCTION

This document is designed as a supporting reference to the Video Systems Reference Manual (Video SRM) supplied by the Terminals Interface Architecture (TIA) group. It covers only selected SRM topics, and is not intended as a comprehensive treatment of them.

The purpose of providing a Programmer's Guide to the SRM is that it is probably unreasonable to expect the average programmer to read through the entire document and understand it thoroughly. Programmers are going to use the SRM for what it is - a reference manual. They will consult it when they need to do something, but they are not expected to thoroughly understand the operation of terminal products (as opposed to terminal implementors, who presumably will read the SRM in detail before their implementation and become thoroughly acquainted with it). The Programmer's Guide, therefore, is intended to provide some general context of what programming a terminal is all about, and to accentuate important points that are documented elsewhere in the SRM.

B.1.1 SCOPE

The Programmer's Guide applies to all DEC products which are intended to be certified as conforming to the Video Systems Reference Manual. It also applies to software which intends to use DEC products in a conforming manner.

B.2 GENERAL PROGRAMMING GUIDELINES

The Video Systems Reference Manual contains performance requirements for conforming equipment, as well as for conforming software. A primary consideration for defining conformance requirements for software is to maximize its independence from the device.

B.2.1 LEVELS OF ABSTRACTION

An effective approach for terminals application programming is to partition the effort into distinct levels of abstraction. The approach is effective for one person writing one program, or several people writing several programs. The levels of abstraction are the application program, the routine library, and the operating system. The routine library can be subdivided into two parts: a logical layer and a back end layer that deals with the specific characteristics of the terminal being used.

B.2.1.1 Application Programs

The principal guideline for application programs is to avoid the use of escape sequences and control sequences directly. Applications should make calls to subroutines to perform the required functions, including those at a higher level, such as drawing a square or deleting a paragraph, or those higher than the individual functions outlined in the TIA specifications. A library of such routines should be developed and maintained by DEC for its customers.

B.2.1.2 Routine Libraries

Library routines can be effectively divided into two sections. One part of the routine can be totally independent of the data syntax and/or protocols being used on the device. The other part, the "back end" of the routines can take the features/idiosyncracies of the specific device into account. For example, a library routine could implement the Insert Character function, but the VT100 will not implement that in hardware. Therefore; the routine could perform several operations at the back end to perform the Insert Character function. That technique would allow the application program to be written as though the Insert Character were available on any device, thus allowing the customer to interface his software application to a wide variety of terminals and manufacturers.

DEC's VMS Run-Time Library keeps the call interface to the application independent of the device by utilizing several back ends, which allows it to interface to various types of terminals, and permits the customer to add foreign terminal support.

This guide is directed more to programming the back end processor for the routine library, than to programming the applications. Ideally speaking, only the library routines should receive the call interfaces. When it is necessary for the application to receive them, or where an application requires its own library, the guidelines outlined in this manual should be observed.

In general, DEC systems software allows whatever functionality is allowed by the hardware. There may be a need at times for an application program to perform functions not provided by the library, but that are within the capability of the terminal. Where such functionality is added to the library, the device dependent features should be at the device-dependent back end as much as possible.

B.3 COMMUNICATION CONTROLS

It is necessary to clarify the TIA interface for switching a G-set into the left half (GL) or the right half (GR). Such switching requires the use of modes, which are problematic to handle in software. The software works far more efficiently if the data it is operating on is independent of a mode context. The reason is that a string passed from one procedure to another encounters a procedure barrier, making the switch a cumbersome one. If the interpretation of that data is subject to a mode, the mode must be passed explicitly as another parameter or embedded in the string somehow. It is far simpler to pass the string and have its meaning unambiguously understood.

For the DEC Standard 169 the left half (GL) of the 8-bit code is assumed to be the ASCII graphic set, and the right half (GR) is assumed to be the DEC Supplemental (Multinational) graphics. That assumption should be adhered to by all conforming software unless some very explicit, well-documented additional mechanisms are used. Exceptions to the norm such as code extension controls, locking shifts, etc. exist for the exceptional case, or for use by a library routine, but not by the application program.

As covered in the previous section, library routines should perform all mode switching, leaving the applications as device-independent as possible. For instance, if a Multinational application wanted to use the DEC line-drawing set, it could simply request to draw a box from character cell position MN to IJ. The library routine would then perform the necessary switching, using the code extension, to access the line drawing set; when finished, it would switch back to DEC Supplemental. The application program would simply treat all the data that it deals with as ASCII on the left and DEC Supplemental on the right.

B.3.1 TEXT ATTRIBUTES

For text applications involving attributes such as underlining, bolding, etc., an effective technique is to store both the character code and the attributes associated with it as a single storage entity. This could include attributes such as character set, height, width, rendition, etc. An analogy to storing characters in this way is found in applications using integers. Where more complex, higher precision methods for accommodating numeric calculations are required, double-precision integers are stored with their unique attributes; they constitute a completely different data type than single-precision integers.

B.4 CONTROL CODE EXTENSION TECHNIQUES

B.4.1 DESIGNING CONTROL SEQUENCES

The design of control sequences is more complex than escape sequences, due to the presence of private parameters and intermediate characters. Recommended procedure is to make the syntax of the control codes as consistent as possible, and as independent as possible from the application program. All control code extensions should be used by the library routines rather than the applications as much as possible.

B.4.1.1 Syntax of Parameters

A parameter syntax option for control strings is to use leading zeros. Doing so can facilitate ease of programming; however, unless the leading zeroes are stripped off, they will be transmitted with the strings, slowing transmission time. A tradeoff decision, therefore, between ease of programming and transmission time is necessary.

The same tradeoff is necessary when using omitted parameters. Compacting the strings makes the most efficient use of the line. For ease of programming, however, the program could be set up to transmit a string of parameters, leaving spaces for omitted parameters by inserting semicolons - which would transmit on the line and slow transmission time. The optimum method for handling omitted parameters is to program using the straightforward approach (compacting strings) and put the parameter defaults at the back end of the library routine. The routine could compact strings and omit parameter values that were equal to the default values, along with trailing parameters, etc. In that way, efficiency considerations would be at a lower level of abstraction than the actual logic in the library routine. Such a separation would provide the benefits of both line efficiency and ease of programming.

B.4.1.2 Size of Parameters

Although the architecture does not limit sizes of parameter values, they are usually constrained by the implementation; not all value sizes will be recognized by all terminals. The SRM states that 255 decimal will be recognized by any implementation as a maximum value; values greater than that may not be recognized by some devices. The SRM also states that 16 parameters is the recommended maximum number for an implementation to allow in a string. Though some implementations may allow more than that, it is not safe to exceed that number; more than one string may be required.

B.4.2 TERMINAL INITIALIZATION

It is stated in the SRM that there is a definite technique which can be employed for initializing the communications line that should work under all states of the communications interchange. Examples of communications line initialization are: (1) the operating system broadcasting a message on a terminal while the application is running, (2) the application returning control to the operating system, and (3) the application programmer encountering a breakpoint during execution and entering the debugger.

Generally, when the communications line is initialized, the line is in an unknown state when the new program takes control. The program may be in the middle of a control sequence, device control string or escape sequence. It could be in an XOFF state, with the terminal not accepting data, or could have switched in new graphics (such as the line-drawing set) on top of the DEC supplemental set.

B.4.2.1 XON Resynchronization

A specific sequence of instructions is required to put the terminal into a known state. First, communication (terminal synchronization) should be restored with the XON control character. It will be accepted whether the terminal is in an XON or XOFF state. The control is ignored if the terminal is in an XON state. If it is in an XOFF state, the XON control will restore line synchronization and allow the terminal to accept data.

B.4.2.2 Control String Terminator

The second instruction required is a String Terminator control character encoded as a 7-bit Escape Fe sequence. If the terminal is in the middle of an escape sequence, control sequence, or device control string, the String Terminator character terminates it and causes subsequent characters associated with it to be ignored. The sequence is printed on the terminal, and the terminal restored to a state in which it can accept new control or graphic characters. If the terminal is in none of those states, the string terminator character is ignored.

B.4.2.2.1 Printer Controller Mode

Printer Controller Mode (MC) is a higher level management function than other modes. It is possible to log onto the system with the terminal in Printer Controller Mode, print out everything being executed by the applications and switch back to the operating system. As long as the terminal is in Printer Controller Mode, it will communicate to the printer, but not to the terminal. Therefore, if an application turns on Printer Controller Mode during execution, and does not turn it off, commands will not echo

on the terminal display when control is returned to the operating system. An explicit reset of Printer Controller Mode is therefore necessary to clear the communications line.

B.4.2.2.2 Selecting Conformance Levels

To put the internal state of the terminal into a known condition, the software must send the equivalent of a 'soft reset' which selects the terminal's conformance level. The Select Conformance Level control sequence, or DECSCL, puts the terminal into a known state; that state can be predicted according to values outlined in the SRM.

Two conformance levels exist, Level 1 and Level 2. Application software should select the appropriate conformance level for the type of operation it intends to perform. Level 1 is defined strictly to provide compatibility to existing programs. Level 2 is the preferred mode for character cell displays and for new software; new software using this synchronization technique should be able to accept and operate at Level 2.

B.4.2.3 Full Initialization Sequence

The following reset sequence should be used by all operating systems and applications to initialize the terminal when taking control: the Control String Terminator (ST), followed by Reset Printer Controller Mode (ESC [4 i), and concluded by a Select Conformance Level control (ESC [62 " p, for Level 2 operation). To insure that it works on all devices, all control codes should be transmitted as 7-bit ESC Fe sequences.

The entire initialization sequence is thus:

ESC	\	ESC	[	4	i	ESC	[	62		"	p
1/11	5/12	1/11	5/11	3/4	6/9	1/11	5/11	3/6	3/2	2/2	7/0

B.4.2.4 Device Control Strings

The bit combinations 08 through 13 (decimal) are provided for purposes of formatting the control string, but have no additional significance to its interpretation. The interpreter normally ignores the Backspace, Carriage Return, and Line Feed characters in a control string. Using an editor, however, it is possible to put those characters into a long control string in order to break and edit the string without effecting its interpretation. Backspace, Carriage Return, and Line Feed are recognized and interpreted in a control string within the context of the ReGIS T (Text) command, since it also functions as an editor.

In summary, control characters such as Backspace, Carriage Return, and Line Feed are permissible in control strings syntactically, but are ignored functionally, except when occurring in the ReGIS Text Command. In some implementations, certain control characters

will also not be permissible syntactically, since their semantics is defined by the control string internal format of the particular device.

It must be noted that control strings other than Device Control Strings (DCS), such as Application Program Commands (APC), Operating System Commands (OSC) and Privacy Messages (PM) are not currently implemented by DEC products and should not be used by component software.

B.4.2.5 Select 7-bit and 8-bit C1 Transmission Modes

The Select 7-bit and 8-bit C1 Transmission Mode control functions are described in the chapter "Code Extension Layer". Since these modes are set by the Select Conformance Level control, it should not be necessary for conforming software to use them. Setting them via the DECSCL control allows them to be changed from the default values; also, it is consistent with the goal of keeping application software independent from terminal modes.

The 7-bit C1 Transmission Mode causes the terminal to send C1 controls to the host coded as 2-character Escape Fe sequences instead of symbolized 8-bit controls. This mode permits existing software running at Level 1 conformance - in which C1 controls are coded only as 7-bit - to accept 2-character Escape Fe sequences without changes. New software, however, should be designed to accept either the 7-bit or 8-bit form, in order to give the software independence from the terminal's mode setting.

Note that the terminal will accept data in any format from the host, including 7-bit and 8-bit coding characters and 7-bit and 8-bit C1 controls, regardless of the mode, since the modes effect transmission of data from the terminal to the host only.

B.5 TERMINAL MANAGEMENT

B.5.1 DEVICE IDENTIFICATION

Conforming software should use none of the device identification functions; terminal identification should be performed by the operating system. When the terminal is logged on, the operating system should request the terminal's functional (and/or product specific) characteristics, and once they are recognized, identify them to conforming applications. Application software can obtain the device characteristics from the operating system.

B.5.2 DEVICE CONTROL

Controls that change the transmission modes from the host to the terminal should be performed by the operating system rather than by applications and library routines. Library routines should call the operating system to change those states. In that way, the operating system will know and control the state with no necessity to monitor the output stream. Different applications will be able to operate the terminal in a state always known by the operating system.

B.5.2.1 Primary Device Attributes

The chapter "Terminal Management" describes the Primary Device Attributes (DA1) control. The first terminal function to be performed by the operating system is the Primary Device Attributes request, in order to determine the functional characteristics of the terminal. It should not be used by conforming applications software. This control is designed to replace the existing Device Attributes control used by operating systems. It allows the device to identify itself to the operating system by functional characteristics rather than by specific product characteristics. It does so by identifying a specific class of operations. There are additionally levels of operation within each class; each higher level is a super set of the lower levels that precede it. At each level, there are a number of extensions which are identified in response to a request. The extensions describe the functional capabilities, which are essentially options at each level of the architecture; they may or may not be implemented in a given product, and/or may change from time to time.

The benefit of the Primary Device Attributes control is that because devices are identified to the operating system according to functional, rather than product, characteristics, new devices can be added to the system without changing existing software.

B.5.2.2 Secondary Device Attributes Control

The Secondary Device Attributes (DA2) control provides for device specific identification of terminals to the operating system. It causes the explicit product identification and revision level of the device to be returned. This control allows for backwards compatibility with previous devices and software that must know the product characteristics of a device that's being used. Conforming software should not use this control, but it is a good practice for operating systems to store the product specific information.

B.5.2.3 Identify Terminal Control

The Identify Terminal (DECID) control is provided for purposes of backwards compatibility with existing terminals. It causes the functional characteristics to be returned - the same response as the Primary Device Attributes - when the terminal is in Level 1 or Level 2 Character Cell mode of operation. It should not be used by conforming software, because it uses an encoded representation that is reserved for standardization by the ANSI and ISO standards. Should DEC implement these control standards in the future, any DECID sequences in use would be incompatible with the newly defined function.

B.5.2.4 Select Conformance Level Control

The Select Conformance Level (DECSCL) control allows the application software to set the device to a specific level of operation for interface compatibility. The purpose of the DECSCL control is to provide backward compatibility; it may be desirable for some implementations to operate the device at a lower level of conformance. The principal guideline in use of DECSCL is that the operating system should always operate the device at its highest level of conformance; it can be certain of this by selecting the highest conformance level (the level reported by the device) whenever the operating system takes control from an application. This technique helps to keep the software independent of terminal modes.

When a level change is called for by an application subroutine, operating systems with the capability of doing so should keep the terminal in the higher-level mode and perform a translation for the application. For example, software not designed to accept 8-bit characters would require the terminal to be in Level 1 mode, since Level 1 Character Cell Display does not transmit 8-bit characters. The operating system could leave the terminal at Level 2 Character Cell Display and filter the 8-bit characters, issuing a warning (bell, etc.) if one was received. Thus, the state of the device would neither fluctuate nor have any dependence on the software. Implementing the DECSCL function via the operating system is more difficult and requires more functions, but eliminates terminal user confusion about the state

of the terminal.

Another advantage of this approach becomes apparent when the user wishes to type information ahead to a subsequent application. For instance, if application A accepted only 7-bit characters and application B only 8-bit characters, the user could type to application A and, during its execution, type 8-bit characters ahead to application B. When application B executes, the operating system will begin to accept the 8-bit characters and pass them on. The application interprets the correct terminal mode at the point it actually calls the operating system for characters.

B.5.3 DEVICE STATUS AND TEST

The Test Control (DECTST) is meant strictly for hardware diagnostic purposes and should not be used for application or library software.

B.5.4 TEXT CURSOR ENABLE MODES

Text Cursor Enable Mode (DECTCEM) allows the cursor symbol to be turned on and off. This mode can be used to eliminate cursor flickering or bouncing while the display is reformatting by turning the cursor off, reformatting the display, restoring the cursor to its position, and turning it back on. To avoid user confusion, the cursor should only be turned off for very short periods of time, and turned back on when the reformatting operation is finished.

B.6 LOCAL FUNCTIONS

Certain keys on the keyboard have been reserved for local functions. They are used in products which implement functions not accessible remotely. Some local functions will not be available in all product implementations; they will need to be performed by host software. The sequences for the function keys are well defined; general applications and library software should not use them, and should ignore their codes if they are received. Ideally, the operating system should recognize the codes and perform the function locally, without passing them on.

DEC assigns and maintains a designated, registered sequence for each function key on the keyboard. Conforming implementations should become familiar with this registry and use the keys only as designated.

B.7 SYNCHRONIZING THE APPLICATION AND TERMINAL

B.7.1 AUTO REPEAT

DEC operating systems are designed such that an application, such as an editor, which uses the cursor keys to update a text image can get control frequently enough to insure that the user will not get too far ahead of the application, even when using Auto Repeat. The application program should assume responsibility for that synchronization, rather than attempting to do so by turning off Auto Repeat, locking the keyboard, etc.

B.7.2 TYPING AHEAD

Applications should allow the terminal user to make the decision whether to type ahead or not; the application program should not make it. Problems can develop when the application program detects an error in the instructions that have been typed ahead. If a sequence is determined to be erroneous, it may not be executed; subsequent instructions could then produce erroneous, possibly disastrous, results. Therefore, it is recommended that applications clear type ahead only when an error has been detected. The application should call the operator's library routine, which should call the operating system to clear type ahead. An application program should not clear type ahead when starting up; it should assume that the option to type ahead exists, and that the operator has decided to do so.

B.8 ReGIS GRAPHICS PROGRAMMING

B.8.1 SYNTAX

B.8.1.1 Elements of Syntax

The ReGIS syntax has five elements, the command keyletters and four argument types. The four types are:

1. Numbers - which have various semantic meaning.
2. Bracketed extents.
3. Parenthesized extents (options).
4. Quoted strings (used or text arguments or character indices).

B.8.1.2 ReGIS Flow of Execution

The entire flow of ReGIS execution is serial. The user begins with a selected element which sets the state so the following elements are executed as they are encountered. An important aspect of the execution is that ReGIS doesn't require that more arguments be received in order to take action on a given argument. Generally speaking, an action can take place from within the context of any of the argument types.

ReGIS actions can be visible or non-visible. An example of a visible action is the receipt of a position specifier while in a "V" (vector) state. The device would have enough information with that datum (and the stored state of the machine) to draw a vector in the appropriate line, color, pattern, etc.

Non-visible actions are those those which change state. For example, if the device were working in a "W" command and received an option to change color, an element of state would be changed in the machine that would lend its attributes to all subsequent applicable visible actions.

The instruction process has a well-defined hierarchical flow that is not too deep or extensive. The user works from the context of a specified command state, or at command level - outside the command state context. If, for instance, ReGIS is synchronized by some means (see next section), the system is at command level; subsequent arguments will be recognized but will cause no action to be taken, there being no command context to interpret them against. ReGIS arguments are self-delimiting; depending on the current command context, ReGIS requires proper character types to start and end instructions. Numbers are looked for in digit mode, quoted strings must always start and end with quotes, parenthesized extents start and end with parentheses, etc. Should the ReGIS interpreter be put in a non-command state at some point, arguments received after that will be parsed and recognized, but

no action will occur. It is possible to parse through a series of instructions that was sent to find out what has happened, however.

Since ReGIS allows a good deal of flexibility in including non-instructions with instructions, a good procedure to follow at points where state changes could cause problems or confusion is to include a statement within the instruction flow about it such as:

```
;"this is a comment. State has changed here."
```

B.8.1.3 Synchronization Procedures

It is possible to put ReGIS out of synchronization with erroneous syntax. For example, ReGIS option specifiers are nested at different levels, with each level using the same syntax as the entire instruction set. If a leading or trailing (left or right) parenthesis were omitted in any of the option sets, ReGIS would be out of synchronization. To write in the color red, for instance, the instruction:

```
W(I(R))
```

would be required. I (Intensity) is a parenthesized option and R (red) is the parenthesized extent containing the color specifier. If the second left-hand parenthesis had been omitted, resulting in the instruction:

```
W(IR)
```

being issued, R would not be interpreted as a subargument to the Intensity option, but as another option to the W command since it's at the same parsing level as the I. It happens that R has semantic meaning to the W command; it instructs to 'Write in Replace mode'. Therefore a well-structured ReGIS command was sent - and interpreted at the wrong level. If one of the right-hand parentheses had been omitted in the above command, resulting in the command:

```
W(I(R)
```

all instructions received after that command would be interpreted as options to the W command. Some of the instructions would be recognized by ReGIS in the W context, some would not, but none of the instructions would do what the user intended.

Quoted string commands also can easily put the system out of synchronization. If the Text command

```
;"this is a comment. State has changed here.
```

were sent with the end quote mistakenly omitted, all subsequent instructions would be interpreted literally and displayed on the screen as text. Upon encountering another double quote, (intended

to be the beginning of another text string), the system would be back at command level interpreting characters as ReGIS instructions.

It is important for the implementer to arrive at an instruction, or sequence of instructions, for synchronizing the system and setting it to a known state, preferably at the command level.

B.8.1.3.1 Parser Level Synchronization

B.8.1.3.1.1 The Semicolon

The semicolon acts to synchronize the system from within any ReGIS command context - except that of quoted strings (as indicated earlier, command syntax is interpreted as literal text characters within quoted strings). The semicolon overrides the nesting level of options, and provides a recovery mechanism for line errors, programming errors and other conditions that put the system out of synchronization.

The semicolon is the simplest, most straightforward method of synchronizing ReGIS. For instance, if a semicolon were to be sent before each command, it would return the interpreter to command level, irrespective of present command context, thus assuring the system is always synchronized.

B.8.1.3.2 Activation Synchronization

Activation synchronization is not a ReGIS feature as the semicolon is, but is a required capability for conforming ReGIS implementations. The ReGIS Extension states that the mechanism (escape sequence, etc.) used to activate ReGIS in the product must contain a means of forcing the ReGIS interpreter to the command level. This means of synchronization is at the discretion of the programmer, and is implementation dependent. In some implementations activation synchronization has meant that the state of the ReGIS interpreter is preserved across changes of mode; in other implementations it has not. For example, the VT125 implementation allows transition from ReGIS to text mode and back to ReGIS; ReGIS processing will pick up in the mode it was in before the transition. The GIGI implementation, on the other hand, does not retain the command context across a change of mode; the command must be reissued when returning to ReGIS.

The optimum goal for an implementation is to allow the user to exit and reenter modes without restating the existing command context; it is recognized, however, that this goal will be beyond the capabilities of some products. Also, in some implementations it is possible that other state settings, such as color specifiers, etc. will not remain stable across transitions of mode; an example of this is when hardware is shared among various protocols. It is important that the implementer be aware of all such changes of state and take them into account.

B.8.1.3.3 State Initialization

State initialization as a synchronization method is based on the Screen Erase function. It is based on the following two requirements as stated in the ReGIS Extension:

1. Devices which implement ReGIS extensions are required to implement default values for all such semantic features which effect the visible display.

2. For devices which have ReGIS extensions, all parameters and attributes - except possibly for screen attributes themselves - should take on their default values upon the execution of a Screen Erase function.

State initialization as a synchronization method requires that whenever a Screen Erase function is performed, all optional settings (extensions to the ReGIS base logical device) must reset to their default values. A list of the options and their recommended settings is as follows:

1. The line pattern setting should be returned to one.

2. (Possibly) reset the loadable character set selector to the default character set. There are two schools of thought on this, however; the other one says to keep the last character set that was loaded.

3. (Possibly) return the intensity selector to a default value upon a screen erase.

B.8.2 THE WRITING COMMAND

B.8.2.1 Writing Modes.

A predicate of ReGIS is the pattern register, which is a binary pattern of 1's and 0's used to select between the foreground intensity and background intensity (also called the secondary foreground intensity). The W(I) command is used to write to the the foreground specifier, and is indicated by the 1's in the pattern register.

The treatment of the 0's in the pattern register differentiate the writing modes. They are Overlay, Complement, Replace and Erase. A fifth mode, Negate, may be used to cancel/modify any of the other four.

B.8.2.1.1 Overlay mode - W(V)

Overlay mode is the base ReGIS device drawing mode. It is generally used as the default mode, since it is the most straightforward and simplest to program and execute. In Overlay mode, all 1's in the pattern register are executed into the display, while zeroes are not. In other words, the current register setting is executed into the display.

Overlay mode is analogous to a pen plotter that moves the pen along while lifting it up and down. The pen 'down' state - when it draws on paper - corresponds to the 1's state. The pen 'up' state - when it moves without drawing - corresponds to the 0's state.

B.8.2.1.2 Replace Mode - W(R)

Like the Overlay mode, Replace mode executes the 1's in the pattern register (the foreground intensity) into the display. For the 0's, however, it executes the background intensity (secondary foreground color). For instance, if the background specifier is red, and the foreground specifier is white, the result will be alternating red and white dashes, irrespective of the current background color.

After a Screen Erase, the behavior between Replace and Overlay modes is identical except at dashed line intersections, which disappear in Replace mode. This similarity holds true until a background color is set. If a red background were set, Overlay mode would cause a dotted white line over the background to be drawn, where Replace mode would cause a line with alternating white and red dashes to be drawn. Replace mode writing activity is implementation dependent.

B.8.2.1.3 Complement Mode - W(C)

Complement mode causes the 1's in the pattern register to complement and 0's to not modify the display. Complement mode is also implementation dependent. For example, in the VK100 (GIGI) Complement mode complements the display plane (which is just one plane deep), but does not modify the intensity attributes for that location. The VT125 display plane is two planes deep, and the intensity setting 'travels with' the drawing action, so to speak, and nothing is left over. After a Screen Erase, no locations have hidden attributes to be recovered by a Complement operation, so all of the bits at a given pixel location are complemented.

Possibly the best use for Complement mode in an implementation is for putting the display back to its original state; this action occurs if the same sequence of instructions is executed twice in Complement mode. The results of one execution of Complement mode are essentially undefined or product dependent, but two executions reverts the display to its original state; this is true of all products. In this way, a cursor could be drawn, because it (the cursor) could be executed, and then re-executed in Complement (so it goes away), producing a 'blinking effect' that the user could predict and have temporal control over. Another possible use for Complement mode is to highlight a field while in Shading block mode. Then with a re-execution of the Complement operation, the display would revert to its previous state - un-highlighted and otherwise un-modified.

Complement Mode should be implemented only where the programmer has complete control over the display medium. In more sophisticated systems where the programmer may not have complete, direct knowledge of how the screen refreshes, it should be viewed as a limited utility. A device's refresh method could produce unpredictable results with regard to Complemented locations (pixels). Particular caution is recommended in using it for systems that have layering between the users' perceived execution and the actual physical representation. It does not work well as a virtual tool, except for returning things to their original state as mentioned above. Even to accomplish that, intervening actions must be precluded from interfering.

B.8.2.1.4 Erase Mode - W(E)

Erase mode ignores the pattern register entirely. Its primary use is to clear areas and return them to the background state, particularly when used with shading or filling operations. A viewport system implementation, for example, could use it in a shaded area to clear a rectangle to the background state. Other uses of Erase Mode are:

1. To erase line patterns before executing other lines.

2. To erase individual constructs on a feature-by-feature basis, though this use can leave background 'holes' in certain cases, such as at line intersection points, etc.

3. To delete text, in some cases, though it is not designed for that use and is probably the least efficient way of doing it.

B.8.2.1.5 Negate Mode - W(N)

Negate Mode can work in conjunction with all the other writing modes. Basically speaking, it inverts the current state of the pattern register; the 1's become 0's and the 0's become 1's. Negate/Replace mode, for instance, could be used to produce text in reverse video, provided its setting in the character cell is normal. Negate used with Complement, Erase, and Overlay do not have any apparent utility at this point; it would be more efficient to invert the pattern register manually.

B.8.2.2 Write Command Extensions

B.8.2.2.1 Raster Extensions

The V,R,E,N and C options to the W command are implementation dependent raster extensions. They are the standard, compact writing modes which are transportable among all (and only) raster devices.

B.8.2.2.2 Open Extensions

The F and W options to the W command are bit plane (state machine) controls for extremely explicit descriptions of the writing modes.

These options are ReGIS open extensions, since they are transportable to a wide range of devices, but their cost/performance utility is highly implementation dependent.

B.8.3 SCREEN INSTRUCTION

B.8.3.1 Hard Copy Commands

The ReGIS Extension states that the S(H) command is used to retransmit the visible image on the display surface to a second device which may be a hard copy device. As an example, the syntax S(H[x,y][x,y]) is proper for the VT125. The H option to the S command is classified as a ReGIS open extension, since it isn't possible to predict the ability of various devices to re-dump. For example, a laser printer is a good device to accept ReGIS, even in raster format, but it isn't clear that it could retransmit to the raster image.

B.8.3.2 Address Space

One of the key portability constructs of ReGIS is the ability to impose an address space. If the Screen Addressing (S(A...)) construct at the beginning of the session is not used, the application will be programmed for that particular device and will not transport to other devices. A good approach to this would be to decouple physical and logical addressing at the default level. For example, although the actual physical measure of DEC's CT device is 960 x 240, the screen default was set at 767 x 479, equivalent to the VT125. In a VT125 emulation, therefore, the emulation and square pixel conversion has already been accounted for.

It is recommended that implementations always use the S(A) command, even if it means re-instituting the expected device defaults every time. It will then be certain that defaults are instituted in devices being transported to and the full screen used rather than having the viewing area shrink into a corner on a large screen, or overshooting on a small screen. It provides reasonable certainty that some sort of reasonable screen results will be obtained. The use of the S(A) command is covered very thoroughly in the chapter "ReGIS Graphics Extension".

B.8.4 MACROGRAPHS

The macrograph construct is derived from the macro instruction construct used in assembly language programming. It is included in ReGIS to provide local storage of graphic commands for frequent reuse throughout an application. The macro concept is actually at a higher level than ReGIS, and could be utilized above combinations of protocols. ReGIS and NAPLPS, for example, are text stream encoded protocols which, if the proper superior encoding existed, could be included in a 'super parser' which would accept both streams of text. Each could then be invoked from within individual parsers to pull out its stream; this is possible even though the NAPLPS macro structure is not compatible with that of ReGIS.

B.8.5 MACROGRAPH CONSTRUCTS

The three constructs of primary concern to the programmer are: the definition of an individual macrograph, the execution of an individual macrograph, and the clearing of the macrograph state.

Any combination of individual macrographs may be used and/or repeated in a given ReGIS command. ReGIS has a protocol-specific macro encoding built around a particular character, in this case the 'at' sign character (@). This specified character never goes into the ReGIS parser, regardless of context or current state of interpretation, because it indicates there is a macro level operation taking place above the parsing level.

The free-form structure of ReGIS allows the user to enter any combination of characters into macrographs; they can be entire commands, portions of them, or some mixture of each. When the macrograph invocation character is recognized at the pre-parser level, the macrograph data base is interrogated and the substitute character string read out. That character string is applied in the existing context of the ReGIS interpreter. For example, if ReGIS were at command level and encountered a macrograph the contents of which began with a "V", the ReGIS context would be changed to the Vector interpretation state and the remainder of the macrograph read within that context. Each character contained in the macrograph would be interpreted by the parser as though it had appeared individually along the communications line.

B.8.6 MACROGRAPH INVOCATION

B.8.6.1 Macrographs Invoking Others

Macrographs may invoke other macrographs; ReGIS recognition of macrographs is recursive. It should not be assumed that the entire contents of one macrograph will fall directly into the ReGIS state. One or more of the commands may be invoking other macrographs.

B.8.6.2 Macrograph Invoking Itself

A macrograph cannot invoke itself. Invocation of macrographs is entirely deterministic and non-conditional, since there are no conditionals in ReGIS. Therefore, it would not be possible to recognize an 'exit' condition in a macrograph that invoked itself.

For example, a path from macrograph D through other macrographs that further invoked macrograph D would repeat that same path predictably and exactly, resulting in an unbounded recursion requiring error recovery. In ReGIS devices, unbounded recursions are usually implemented so that the recursion stack overflows, creating an error condition which purges everything back to command level. It is entirely illegal to use any sort of recursive macrograph invocation, direct or indirect. Implementations may insert a flag that signals as each macrograph is accessed. For instance, it would signal when macrograph D were being accessed. If macrograph D invoked macrograph L, macrograph L would be flagged as accessed, if L invoked X, X would be so flagged. If X then invoked D, L or X, the flag would recognize an unbounded recursion state and terminate. Recovery mechanism and recovery recognition scheme is the responsibility of the implementation; the ReGIS Extension does not impose one.

B.8.7 MACROGRAPH STORAGE

ReGIS provides for exactly 26 separate macrographs, irrespective of implementation. While the ReGIS Extension calls for a minimum of 4000 characters macrograph storage, this may be an implementation dependent value due to device limitations. Should a user need to know the amount of available macrograph storage space, a ReGIS inquiry function exists which allows the user to interrogate the system. Otherwise, a 4000 character storage space should be assumed.

There are 'hooks' at certain points within ReGIS into the macrograph storage. The Report command, for example, can send back details of space, usage, or individual macrograph contents. So macrographs do enter the parser, even though they are superior to it conceptually.

While a macrograph can define the invocation of another macrograph, it cannot define its storage because macrograph definitions cannot be nested. The macrograph initiator and terminator are specific constructs. The macrograph initiator is:

'at' sign (@)
colon
letter (macrograph name)

The macrograph terminator is:

'at' sign (@)
semicolon.

The macrograph initiator immediately turns off all ReGIS and nearly all macrograph parsing. At that point, the only construct which will not be stored in a macrograph is the terminator. It is, therefore, impossible to nest macrograph definitions. The following sample instructions illustrate an attempt to nest definitions in macrographs A and B:

```
@:A  @:B(contents of B)  @;  (remaining contents of A)@;
```

These commands would not be interpreted as expected because the second instruction (@:B) would not be recognized at the macrograph storage level; it would be stored literally in the macrograph. Also, the first (@;) (third instruction) would be recognized as the termination of the entire macrograph extent. The last instruction, intended to invoke the remaining contents of macrograph A would be passed along to the parser, since all the macrograph storage constraints would have been satisfied; the final (@;) would simply act as a spurious terminator.

B.8.8 MACROGRAPH UTILITY

B.8.8.1 Specifications

Macrographs have been used effectively for storing combinations of specifications. Although ReGIS technically does not have the capability for named variable storage, macrographs can provide the capability to a limited degree. The following example macrograph (R) stores the HLS specification for the color red (hue 120, lightness 50 and saturation 100).

```
@:R H120 L50 S100 @;
```

This specification could be invoked within the context of a Write

mode instruction, as a suboption to the I command as follows:

```
W(I(@R))
```

or as a screen mapping command:

```
S(M 3 (@R))
```

Though the context varies, the same macrograph is used. The macrograph character stream is a literal stream; it could be invoked as two characters at the device, rather than the specification being retransmitted as several characters from the source.

B.8.8.2 Polygons

Macrographs can be used to define polygons. To accomplish this, the macrograph can be stored and executed twice, and the Write modes changed outside the macrograph. For example, Shading mode could be turned on, and the macrograph invoked; this would draw a shaded area in the shape of a polygon. Shading could then be turned off, the macrograph invoked again, the color changed and the border drawn around the polygon. Thus, the description of the polygon would be stored in the macrograph, and could be drawn shaded with a highlighted border.

B.8.8.3 Icons

Macrographs can be used effectively for drawing icons. The user may want to draw an icon that can be stored away for repetitive use, such as a Stop sign, for example, to be displayed at different places on the screen at different times. A line drawing of the picture could then be plotted out using only relative coordinate specifiers within vector imposition commands as the following example which draws a star:

```
@:I V[+20,-50][+20,+50][-50,-30][+60,][-50,+30] @;
```

From command level, the cursor could then be positioned at a desired point and the macrograph invoked to display the icon:

```
P[x,y] @I
```

Once the icon were positioned, other drawing operations could be performed on the picture, changing writing modes as necessary. If writing in Complement mode, the icon could be cleared out, if desired, by invoking the macrograph again in Complement mode at the same point.

This technique provides a cursor support capability. For example, if an interactive application required the tracking of an input device and the particular ReGIS device did not support Locator mode report, a cursor icon could be programmed to write, move, erase itself as necessary, or perhaps just display itself repeatedly - such as for graphing structures.

B.8.8.4 Coordinate Lists

Macrographs provide the capability of storing a coordinate list. Two examples show how data compression can be used in macrographs to do a variable presentation of a picture by storing away entire command lists. The following example shows a set of bracketed extents within a macrograph used in a Vector construct to draw a Begin and End block around the outside, creating a closed polygon:

```
@:C [ ][ ][ ]@; V@C V(B)@C(E) V(B)@C(E)
```

The same structure could then be invoked as a closed curve construct. Macrographs can be used in this way to demonstrate how the closed curve command works in ReGIS; a specified sequence of points are first connected with vectors and then with curves.

Another use for this type of curve construct is to invoke a different macrograph inside the command in order to apply markers at each point along the coordinate list extent. The following example has a coordinate list contained in macrograph C:

```
@:C @M[ ] @M[ ] @M[ ] @M[ ] @M[ ] @;
```

Between each specified coordinate, another macrograph (M) has been invoked which could be an icon such as a small square. If these instructions were executed as a polygon or an open vector set extent, they could send nested calls to a different macrograph to generate a graph, for instance, and draw the icon (small square) marker at each coordinate space to mark the position on the curve extent:

```
@:M P[+10,-10]V[+20][,+20][-20][,-20]P[-10,+10] @;
```

If it were necessary to streamline the execution and not display the icon, the definition of macrograph M could be nulled; it would be invoked as an empty extent between every coordinate point and execute directly, causing no extra overhead in the display.

B.8.8.5 Write Maps

The macrograph structure has been used as a surrogate Write map by DEC's Datatrieve system. Datatrieve allows the user to change the Write map settings in order to present a graph in different colors, changing the shades as desired. However, Datatrieve requires the default Write map to suppress the display while entering an expanded text menu. Datatrieve protects the default Write map settings from destruction by the user's Write map settings by loading the user Write map settings into a macrograph.

Datatrieve is then able to clear the user Write map away while performing its expanded text menu operations, invoking the macrograph to restore the user Write map settings again. The following shows a macrograph (Z) structure containing the user Write map settings.

```
@:Z S(Mn( )n( )) @;
```

The code that interrogates the user for color settings, etc. actually modifies the macrograph and invokes it. Datatrieve can then perform other operations, changing the Write map if necessary, restoring the user settings with the macrograph afterwards.

State features such as the Write map are not accessible to the Report command of ReGIS; in a system application such as Datatrieve, the system and the application both have access to that feature of the machine. Macrographs are used in this case to prevent the competing entities from writing to the same extent (without making private copies of that extent), so it can be restored later in an unambiguous fashion.

B.8.8.6 Text

Macrographs have limited application for standard cell text; most individual sections of text do not warrant the overhead of storing and recovering things by name. However, for situations where short and medium length words such as "the" and "first" are used frequently, macrographs can be used to refer to them by abbreviations. This would be keeping with coding efficiency principles that advocate using the least number of characters as possible.

Storing text in macrographs also presents the possibility of creating menus; it is a combination between display lists and local buffering of intelligence at the ReGIS device. Macrographs provide another tool for that type of system.

B.8.8.6.1 Quoted Strings

A caution in using macrographs for text is that they are not recognized or interpreted within quoted text strings. Therefore, it is not possible to use a Text command to invoke a macrograph contained within quoted strings, because the contents of quoted strings will be interpreted literally.

Conversely, it is possible to put all or part of a quoted string within a macrograph. However, the use of unmatched quotes in macrographs will produce erroneous results. The following sample instructions invoke macrograph X containing a Text command, quote character and the first part of a quoted string, followed by macrograph Y containing the remainder of the string and a closing quote character.

```
@:X T'(text) @;  @:Y (remainder of text)' @; @X@Y
```

The first instruction invokes macrograph X which sends the Text command to the parser. The next command is supposed to invoke macrograph Y, but the ReGIS parser has been instructed to execute quoted text (by the open quote in macrograph X) and has received no instruction to do otherwise. The result will be:

```
T, '(text) @Y
```

with no closing quote. This would will display as:

```
(text)@Y
```

and leave ReGIS out of synchronization. This result is obviously much different than what the user wanted or expected.

B.8.8.7 Display Segments

The ReGIS Extension states that macrographs are not intended for use as display segments. In most instances, the complexity required to support segmented displays is beyond the scope that can be offered by macrographs. As indicated earlier, just 26 macrographs are allowed, and a relatively small amount of storage (4000 character minimum) usually allotted. Most implementations will have storage in that range; a powerful implementation would likely have no more than twice that amount. A complex display file structure would be difficult, though not impossible, within those limitations. Pieces of pictures, each treated as a segment, could be stored away in macrographs and executed in turn. Proper Write modes could then be chosen outside the macrograph level, turned on and off non-destructively, opaqued with others, etc. Therefore, this application of macrographs does exist, but on a limited basis.

B.8.9 THE TEXT COMMAND

B.8.9.1 Sizing Text - the S Option

The best method for sizing text is the T(S) command with an integer argument. ReGIS will scale the character by the integer factor in horizontal and vertical extent, and set a default spacing into character escapement for character strings, until the S option changes again. For single-size characters, the instruction is T(S1). For double-size characters, the instruction is T(S2).

Another text sizing method is to use the S and H options with integer arguments. The H effects the height of the character only; it allows further changes to be made to the aspect ratio of the character extent and whatever changes required in text spacing to make that fit. To obtain standard-width characters at twice the height, for instance, the instruction would be T(S1, H2). The H option will work only after the S option, since only the height can be altered after the size has been set. The instruction T(H2, S3), for example, will produce characters three times as wide and three times as high because S sets both height and width, overriding any previous settings.

A default character cell size called a 'unit size' has been implemented in the newer versions. Basically, it amounts to the fact that the character definition as loaded into the program ROM, or down loaded in the Load Character extent, is treated as a logical display (that is, a selectable number of current coordinate units is used for each parameter of the character's height and width). The window for that character has the entire extent of the logical display for that character and can be viewported into a unit size on the screen.

The T(S) command fulfills most text sizing requirements, but doesn't provide for a way to know the actual size of the characters, or how many of them will fit across the display. For instance, if the unit size is unknown and a "standard size one" character is asked for, the actual size of the characters, plus how many will fit across a screen will be unknown. The T(S) command will give some relative size scaling, but the results will be trial and error without knowing the unit size.

B.8.9.2 The S Option Coupled With Other Options

B.8.9.2.1 Specifying Screen Area - The S, M and U Options

When coupled with the M option, the S option can be used to specify the amount of screen area the character will take up. When coupled with the U option, the use of the S option is more complicated, but also provides another means of predicting the amount of screen area the character will take up. The U option should be understood in order to figure out how to get transportability across various text resolution displays. When used with the S option, it becomes possible to size text without knowing the unit sizes, but the actual results will be implementation dependent. Most implementors program their devices at maximum resolution, making detailed text representation a necessity; the handling of detailed text in ReGIS is complex and highly device dependent.

B.8.9.2.2 Character Orientation - The S and D Option

The relationship between the size (S) and direction (D) options is a complicated one. The D option controls the orientation of the character cell. When used in conjunction with the S command in the same option set, it has the side effect of changing not only the cell orientation, but the intercharacter movement. If the S option is used with the D option, it couples the direction specifier to the character baseline. For instance, if the string,

```
T(D90 S1)"ABCD"
```

is sent, the characters will be oriented to each other as follows:

This technique can provide a way of labelling the Y axis of graphs, or can be used in other applications as well.

A D90 instruction without the (S) specifier simply rotates the characters 90 degrees clockwise (an angular orientation, not a pixel orientation) so the characters will display as follows:

The S and D options work for whatever character angles the implementation can support. The simplest and most portable character orientation is in 90 degree increments, though 45 degree increments will usually also work. ReGIS allows arbitrary character orientation, but product limitations usually prohibit it and round to the nearest 45 or 90 degree increment.

B.8.9.2.3 Text Bracketed Options - Space Setting

The T(S)[] command may be used to provide certain specific controls. Using a bracketed argument with the T command, for example, forces an intercharacter space setting that overrides any space defaults set by the character size. This can be used to change the perceived orientation of the lines of the character, the character path. An example of a character path altered in this manner can be found in Figure 8-6 of the ReGIS Extension.

B.8.9.3 Text Control Characters

ReGIS has three basic text control features: backspace, carriage return and line feed. Implementing them is straightforward in the base logical device, since text re-orientation (see above) is not possible in the base logical device. Carriage return returns to the start position of the text string being worked in. A line feed moves down one line vertically, without changing its horizontal position. Consecutive line feeds move along vertically, but the X position never changes because with no reorientation, the text runs horizontally from left to right. Since there is no vertical component to a Carriage Return, text following a Carriage Return goes to the left hand position; this is true with every text command and argument. Backspace moves back one position and overstrikes the character. It is possible with the backspace to go beyond the carriage return position.

In most implementations, Backspace, Carriage Return and Line Feed will work with rotated text. Carriage Return moves along the baseline orientation to either the position at the beginning of the T command, the beginning of the last quoted extent, or to the point at which the last Line Feed was encountered. Using CR and LF control characters, it should be possible to do a rotated block of text as follows:

To summarize, it may be stated that the control characters operate in a way that preserves the logical functions of Backspace, Carriage Return and Line Feed, irrespective of character orientation. These three controls also provide a shorthand method of putting the drawing point at a predictable location; alternative methods of accomplishing this are either to use the pixel vector specifiers or to leave the text command and send a relative position argument each time the drawing position changes.

Other text control characters, such as horizontal and vertical tab, are not covered in the ReGIS Extension, because they lose their meaning when the baseline is neither horizontal or vertical, or when character sizes and aspect ratios change.

B.8.9.3.1 Pixel Vector Specifiers

The text control characters described above provide a means of moving the drawing position in relation to the text characters. Another means of moving it is to use pixel vector specifiers as an argument within a text command. These may be included between quoted string arguments (though not within the quoted string). The way to write a superscript 12 to the ABCD text, for instance, is with the following command:

```
T"ABCD"2"12"6
```

which is the text, the pixel vector specifier to move the position up, the superscript characters, and the pixel vector specifier to move the position back to baseline. The pixel vector specifiers 2 and 6 move the drawing position up and down, respectively. Pixel vector specifiers used in this manner move the drawing location for the next character cell one-half character size. Pixel vector 2 would move the drawing position up one half the character height, while 22 would move it up a full character height. Text pixel vectors are measured differently from standard pixel vectors. Text pixel vector 0 is always in the direction of the text string, 2 is always up from the character baseline and 6 returns to the baseline. Standard pixel vector 0 is to the right, 2 is vertically up, 4 is to the left, and 6 is vertically down. Therefore, text can be superscripted/subscripted with the same pixel vectors whether or not the string is rotated. Pixel vector specifiers are covered in various places in the ReGIS Extension, since they are included in the base logical device as well as the raster and open extensions.

B.8.9.4 Scrolling and Wrap

ReGIS has no provision for scrolling or wrapping text, making it a very limited tool for general text purposes. Since there is no screen margin for text, the system keeps incrementing its X and Y pointers and goes beyond the margins and bottom of the screen. The user may be able to effect a "manual wrap" if the extents of the screen, the size of the character, the number of characters across, etc., are known. However, the system would still be without a general-purpose scrolling function that could satisfy text requirements.

The type of scrolling offered by ReGIS is graphics-oriented and highly device dependent. In the open extensions section of the ReGIS Extension, two scrolling methods, or models, of moving data in a graphics display are described. One model "shifts the data through the space" while the other "moves the viewpoint around within the data address space".

B.8.10 COLOR MAPPING

Color mapping is parameterized in ReGIS through the HLS reference model. The RGB model essentially duplicates the HLS model and is not fully parameterized in ReGIS. For instance, it is not possible to color a figure 20% red, 40% green, and 8% blue. The only ReGIS possibility with the RGB alphabet is to specify the distinct colors red, blue, yellow, cyan, magenta, black or white, which are all fully saturated colors at their maximum intensity. HLS provides the easier model with which to understand colors from a human point of view, since it is based on a perceptual model. It identifies a set of dimensions for color specifications that is based on what people perceive the three elements of color to be. The hue specifier, for instance, indicates the tint, which is the major determinant of the color. The hue specifier is based upon an angular specification that runs from 0 to 360, with blue at 0, red at 120 and green at 240. To change color to a hue between blue and red, the angular parameter between blue and red is adjusted.

The L parameter adjusts color lightness to specify how bright or dim the color is. The S parameter indicates how pastel the shade is - how much that color varies between a gray and a pure presentation at the specified hue and lightness. Color mapping requires a 3-dimensional specification; all three are needed to produce the precise representation of the color. The HLS model has well-defined end points and, while it is not linear along any of the axes, it can be interpreted as linear for the rough scale of models that we have.

B.8.10.1 Color and Gray Scale.

In theory, programs written for color will produce a corresponding visual effect on a non-color device that supports only gray scale. In practice, however, it will not transport perfectly because the ReGIS color model and gray scale is "grainy". Having just four, eight or sixteen individual colors visible at any given time or on a given gray scale limits the ability to shade a particular color to a particular shade of gray. A two-dimensional extent is being imposed into a one-dimensional extent, and some information loss is inevitable. If two colors that map very closely on the gray scale are used (such as bright blue and dim green in television) they could 'collide', or map to the same shade on the ReGIS gray scale. It is actually largely determined by the product's hardware. In most cases, if standard colors are used, or a widely varied set of colors, the chances of gray scales colliding with map from the colors is probably not too great. It is recommended to not use non-standard color gradations to present information. Standard, bright colors such as red, green etc. with the same whiteness saturation and only varying hues will produce discernible gray scales.

B.8.11 INTEGRATION OF GRAPHICS AND TEXT

ReGIS provides very limited control over registration between text and graphics, particularly across devices, or when the text is created with other than the ReGIS Text command. For example, an attempt to use VT100 text as captions for VT125 figures would be severely hampered by insufficient control over scrolling in the VT100. Also, column modes (widths) between the two devices would be difficult to synchronize, since it would not be known whether the user wanted the VT100 column mode set to 132 or 80 columns, or whether the device had an advance video option with only 14 lines in 132-column mode.

Another integrating situation that would encounter problems would be to attempt to integrate text that was created in the Text mode of the VT240 instead of with the ReGIS Text command.

The real utility of integrating editable text with graphics is that while composing, the user views the work outside of the context of the terminal being used; the graphics and text are viewed in a unified text environment. This requires the ability to address the application of various protocols to obtain predictable, integrated results, and is not currently feasible with the various text and graphics protocols used by DEC. To avoid incompatibilities with future products, application software should not depend on interactions between graphics and text. Some examples of interactions to avoid follow:

1. Using the graphics erasing command S(E) to erase text.

2. Using (non-ReGIS) text erasing commands to erase all or part of the graphics image.

3. Using graphics scrolling to scroll text.

4. Using text scrolling to scroll graphics.

5. Mixing text and graphics in the same screen area.

Applications which need to label graphs, charts, etc., or which need to perform advanced text functions such as proportional spacing, italics, colors, etc. should use the ReGIS T command exclusively, and not mix graphics text with text mode text.

```
A
B
C
D

ABCD

ABCDEFGHIJ
KLMNOPQRST
UVWXYZABDE
```

This page deliberately left blank.

Section Index

-L-

-M-

-P-

-R-

-S-

-T-

-X-

VIDEO SYSTEMS REFERENCE MANUAL

Product Reference

Document Identifier: A-DS-EL00070-0C-0 Rev. AX11, 18-Mar-1985

ABSTRACT: This appendix describes the products which have been certified as conforming to the architecture defined in the Video Systems Reference Manual, and identifies any areas in which those products deviate from the architectural specifications.

APPLICABILITY: SARA Formal Cross-Component Standard. Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria".

STATUS: FOR REVIEW ONLY

This document has been placed in the SARA "Formal Cross-Component Standard" category. The material contained within this document is assumed to define mandatory standards unless it is clearly marked as (a) not mandatory or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements. The procedures for modifying or evolving this standard are contained within the contents of this document.

FOR INTERNAL USE ONLY

TITLE: VIDEO SYSTEMS - Product Reference

DOCUMENT IDENTIFIER: A-DS-EL00070-0C-0 Rev. AX11, 18-Mar-1985

REVISION HISTORY: Original Draft 25-Dec-1982
Revision AX01 28-Feb-1983
Revision AX10 15-Apr-1983

FILES: User Documentation EL070SC.mem

Internal Documentation EL070SC.rno
EL070SC.rnt

Document Management Group: Terminal Interface Architecture
Responsible Department: Video Architecture Group
Responsible Individual: Peter Sichel

ACCEPTANCE: This document has been approved by the Manager of the Video Architure Group based on a comprehensive review of its individual sections by the members of the SARA component groups working Terminal Interface Architecture issues. The list of individuals on the review and approval list are on file in Standards and Methods Control.

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, RANI::VIDARCH

Copies of this document can be ordered from:

Standards and Methods Control
APO1/F7, DTN 289-1414, JOKUR::SIMONETTI

CONTENTS

C.1 Product/Architecture Matrix

The following table describes the products which are currently certified or planning to be certified as conforming to the Video Systems Reference manual in accordance with the conformance requirements described the chapter "Conformance Requirements". Products which are planned to be certified but have not yet been tested are indicated by an asterisk (*).

	VT220	VT240	PROFEXT	DECMATE EXT	RAINBOW EXT	VAXSTAI NT
LEVEL 1	X	X	X	X	X	
LEVEL 2	X	X				X
********** EXTENSIONS **********						
132 COLUMNS	X	X		X	X	
PRINTER PORT	X	X				
REGIS GRAPHICS		X				
SIXEL GRAPHICS		X				
DRCS	X	X				
UDK	X	X				
********** OPTIONS **********						
VT52	X	X				

C.2 Rainbow 100A Computing Terminal / 07-JAN-83

The Certification Waiver Panel met on 22-DEC-82 to review the results of RAINBOW Certification. Each deviation that was listed in the Certification Report of 20-DEC-82 was discussed and agreement was reached in every instance as to how the deviations would be resolved. Because of this, the opinion of the Waiver Panel is that RAINBOW has PASSED CERTIFICATION WITH WAIVERS as a Level 1 Device.

Many waivers were granted on the basis that the product would be changed to eliminate the deviations between it and the architecture (see section 1.0 below). In some cases, the changes are scheduled for the first or second planned ECO release, but in all cases, a commitment was made to fix the problems by the next generation release.

Because only part of DEC Standard 169 was implemented, and some problems exist with this and other Level 2 issues, a restriction is placed on the advertising of this product. Examples of the partial/incorrect implementation of DEC Standard 169 are:

- G2 and G3 character set designators parse incorrectly
- DEC Supplemental Character Set was unavailable.

These are Level 2 issues, however, and as such do not effect Level 1 Certification results. Advertising compatibility with either DEC Standard 169 or DEC Multinational is not to be allowed, but advertising the fact that a subset of DEC Multinational is available may be acceptable.

What follows is a list of the actual deviations placed in the categories as a result of the Waiver Panel meeting. The list is included here in order to document what changes were agreed upon in order to gain agreement that RAINBOW may be called a Level 1 product.

CATEGORY 1: CHANGE THE PRODUCT

For items in this category, it is agreed that the problem must be fixed, but that because plans are in place to do so by a specified time, a waiver is granted.

1) <BS> - Backspace

If the screen is set to 132 column mode and the cursor is positioned beyond column 127, the reception of the backspace control will position the cursor to the left margin.

2) <ESC> [2 h - Keyboard-action locked:

When Keyboard Action mode is set while a key is auto repeating, the WAIT indicator lights but the codes do not stop until the key is released. Then no new codes are generated. If KAM is reset while a key is down, no codes are generated until after the key is released. Further, when the keyboard is locked, the break key does not function.

3) Relative Cursor Positioning

When using relative cursor positioning (cursor down or cursor forward) a parameter greater than 255 will cause the cursor to disappear.

4) Auto repeating

If a control key is pressed before a non-control key, auto repeating is suppressed after the control key is released. A key pressed before the control key causes the the control code to be repeated after control is pressed.

5) Undefined Character Codes

RAINBOW displays undefined character codes as upside down question marks.

CATEGORY 2: CHANGE THE ARCHITECTURE

For items in this category, it is agreed that the product was implemented in a reasonable fashion and the architecture may need clarification.

1) SETUP

Entering SETUP resets keyboard action mode and turns off keyboard locked indicator. The architecture will be changed to state that entering and exiting SETUP should not change keyboard action mode unless the user has explicitly executed a function from within SETUP that unlocks the keyboard.

2) Error Bells

The following keys produced key clicks instead of error bells when pressed with control: E01, E09, E10, E11, E12, C10, C11, B00, B08, B09.

CATEGORY 3: CHANGE THE CERTIFICATION PROCESS

For items in this category, it is agreed that the certification process was in error in that it either failed to properly interpret the architecture or failed to properly test the product.

1) <ESC> V - Print-Cursor-Line:

No carriage-return and line feed are appended to the line when performing a print cursor line.

2) Dead Keys

Inappropriate error bells are sounded when the following keys are depressed: E1, E2, E3, E4, E5, E6, F6, F8, F9, F10, F14, F15, F16, F17, F18, F19, F20, Compose. These are specified as dead keys by the SRM.

CATEGORY 4: WAIVE BY AGREEMENT OF THE PANEL

For items in this category, it is agreed that no change is to be made at this time to the product, the architecture or the certification procedures. By granting this waiver, the Panel is expressing the opinion that it considers the issue insufficient cause for the product to fail certification.

1) <ESC> c - Reset to initial state:

RIS fails to accomplish a complete reset. Note: RIS is not a recommended sequence to execute.

2) <ESC> [? 7 h - Auto wrap on.

Auto wrap operates like the VT102 (not considering interdependent sequences) but the VT102 does not conform to the TIA Video SRM.

3) XON/XOFF

When the key combination Cntrl/S is sent, RAINBOW sends XOFF to the host. No other effect appears until the input buffer is tested while RAINBOW is in SET-UP or Hold Screen mode. At this time the buffer stores 235 characters before sending XOFF to the host. However, when RAINBOW exits SET-UP or Hold Screen mode, the screen updates using the buffered data, but RAINBOW does not send XON to the host. After the terminal is reset with Cntrl/SET-UP (or typed Cntrl/Q), the buffer holds 157 characters before sending XOFF and sends XON when the screen is updated.

4) SHIFT & LOCK

If control is pressed with both shift and lock set no code is generated when a code generating key is then pressed.

5) CONTROL

The following keys did not produce any codes when pressed with control: E00, E02, E03, E04, E05, E06, E07, E08, B10.

6) <ESC> [p K , <ESC> [p J - Erase in line / Erase in display.

The RAINBOW escape sequences for erase in line and erase in display only processes the first selective parameter if more than one is sent.

7) ABORT <ESC> SEQUENCE

When escape sequence parsing aborts, it does so in such a way that subsequent characters are printed rather than suppressed.

8) Shift Lock Mode

Shift Lock mode is unavailable.

9) COMPOSE KEY

The compose key beeped instead of producing 7-bit compose sequence codes.

10) PHYSICAL CONNECTORS

The female DB-25 connector on the back of the RAINBOW is not compatible with the VT102 printer port male DB-25 connector.

CATEGORY 5: RESTRICT

For items in this category, it is agreed that although no change to the product is involved at this time, the deviation is serious enough to warrant placing a restriction on what compatibility or other claims the product should be allowed to make.

There were no items in this category.

CATEGORY 6: NO RESOLUTION

For items in this category, the Waiver Panel was unable to agree on a resolution for the issue or on the proposed time frame for resolution, and will present it to upper management for a decision.

There were no items in this category.

C.3 DECmate II Word Processing Station / 20-JAN-83

The Certification Waiver Panel met on 10-JAN-83 to review the results of DECMATE II Certification. Each deviation that was listed in the Certification Report of 06-JAN-83 was discussed and agreement was reached in every instance as to how the deviations would be resolved. Because of this, the opinion of the Waiver Panel is that DECMATE II has PASSED CERTIFICATION WITH WAIVERS as a Level 1 Device.

Many waivers were granted on the basis that the product would be changed to eliminate the deviations between it and the architecture (see section 1.0 below). In some cases, the changes are scheduled for the first or second planned ECO release, but in all cases, a commitment was made to fix the problems by the next generation release.

Because only part of DEC Standard 169 was implemented, and some problems exist with this and other Level 2 issues, a restriction is placed on the advertising of this product. Examples of the partial/incorrect implementation of DEC Standard 169 are:

- COMPOSE is not functional.

- DEC Supplemental Character Set is not available in a way compatible with DEC Standard 169.

- Function key codes are transmitted from the Level 1 terminal emulation package.

It was agreed that because of these issues, advertising compatibility with either DEC Standard 169 or DEC Multinational is not to be allowed. It was further agreed that when DECMATE II implements Level 2 it will no longer be possible to access Level 2 features from Level 1. As a result, these issues will not impact Level 1 Certification results.

What follows is a list of the actual deviations placed in categories as a result of the Waiver Panel meeting. The list is included here in order to document what changes were agreed upon in order to gain agreement that DECMATE II may be called a Level 1 product.

CATEGORY 1: CHANGE THE PRODUCT

For items in this category, it is agreed that the problem must be fixed, but that because plans are in place to do so by a specified time, a waiver is granted.

1) <ESC> # 8 Screen Alignment

Sometimes when <ESC> # 8 is sent, the screen alignment grid pattern is not displayed until some additional character or escape sequence is sent. On other occasions, this works properly.

2) <ESC> # 3, <ESC> # 4, <ESC> # 6 Line Attributes

1. When a command is sent to set a line to Single-Height Double-Width characters, no change takes place until a character or escape is sent to the screen.

2. An attempt to set a line attribute in a line followed by a carriage return/line feed does not set the attribute for that line but sets it instead for the next line and then only if a character or escape is sent on that line.

3. The sequence <ESC>[H <ESC>#6 <CRLF> <ESC>#6 fails to set the proper line attribute for line one. Similarly for other line attributes.

4. When followed by a test string loses characters in columns 31 to 40 and displays the characters from a subsequent line.

5. When followed by a tab fails to display subsequent characters output to the line.

3) <ESC> [? 15 n - Request Printer Status

DECmate II does not respond to the request for printer status.

4) <ESC> [c - Device Status Report

DECMATE II responds with <ESC>[?1;2c indicating it is a VT100 rather than a a LEVEL 1 device. Since the DECMATE II emulates a VT102 rather than a VT100 it should respond as VT102 until software is prepared for the LEVEL 1 response.

5) <ESC> [2 h - Keyboard Action Mode

Keyboard action mode turns off the keyboard including local function keys. However, key clicks do not stop for 4 more keystrokes. When the keyboard is unlocked, the 4 codes are transmitted. Because the local function keys are locked, the only way to turn the keyboard back on is with another control sequence.

6) Parser Error Processing

The parser does not always process error conditions properly. For example, if the sequence <ESC>[[[[[is sent, the extraneous "["'s are not displayed. In some cases, it gets so confused that a <CAN> is required to bring it to life.

7) Default modes

The emulator starts up with the alternate keypad in application mode instead of numeric mode, and cursor keys in application mode instead of cursor mode.

8) <DEL> Processing

If a <DEL> is embedded in an ESCAPE sequence, the sequence is aborted. <DEL> should be ignored in this context.

9) Control S, Control Q processing

E02, D01 and C02 produce no codes when held with Control. This also relates to special processing associated with "\".

10) Function key processing.

The function keys other than delete do not autorepeat.

11) Keyboard response.

All keys click even for invalid combinations, and no error bell sounds for invalid combinations.

12) Legend Strip

The SET UP, BREAK, and blank key indications on the legend strip are incorrect.

CATEGORY 2: CHANGE THE ARCHITECTURE

For items in this category, it is agreed that the product was implemented in a reasonable fashion and the architecture may need clarification.

1) Answerback and disconnect by Control.

CONTROL/BREAK (answerback) and SHIFT/BREAK (disconnect) transmit break. Requirements in the architecture to be changed based on K. House memo to R. Sudama.

CATEGORY 3: CHANGE THE CERTIFICATION PROCESS

For items in this category, it is agreed that the certification process was in error in that it either failed to properly interpret the architecture or failed to properly test the product.

1) XON processing

When returning to the WPS menu, DECMATE II sends XON to the host.

2) SET-UP

SET-UP is not available from the terminal emulator.

CATEGORY 4: WAIVE BY AGREEMENT OF THE PANEL

For items in this category, it is agreed that no change is to be made at this time to the product, the architecture or the certification procedures. By granting this waiver, the Panel is expressing the opinion that it considers the issue insufficient cause for the product to fail certification.

1) "\" (Backslash)

"\" does not transmit a code on alternate pressings. That is to say if two backslashes are pressed only the second is transmitted.

This is related to the \R, mechanism for returning to WPS from CX mode. It is not clear why the terminal, having realized "\R" has not been been entered, cannot send both codes.

2) Transmission Rate

The transmission rate is not limited to 240 CPS. This may cause problems for some operating systems.

CATEGORY 5: RESTRICT

For items in this category, it is agreed that although no change to the product is involved at this time, the deviation is serious enough to warrant placing a restriction on what compatibility or other claims the product should be allowed to make.

There were no items in this category.

CATEGORY 6: NO RESOLUTION

For items in this category, the Waiver Panel was unable to agree on a resolution for the issue or on the proposed time frame for resolution, and will present it to upper management for a decision.

There were no items in this category.

C.4 Professional Computing Terminal / 20-APR-83

The Certification Waiver Panel met on 31-JAN-83 to review the results of PRO 350 Certification. Each deviation that was listed in the Certification Report of 24-JAN-83 was discussed and tentative agreement was reached in every instance as to how the deviations would be resolved.

In the weeks that followed, upper management was made aware of these decisions and has not objected to any of them. Some waivers were granted on the basis that the product would be changed to eliminate the deviations between it and the architecture. In all cases, a commitment was made to fix the problems by the next generation release. Because of this, the opinion of the Waiver Panel is that the PRO 350 has PASSED CERTIFICATION WITH WAIVERS as a Level 1 Device.

It is important to note that although the product has implemented many Level 2 features, specifically those that pertain to DEC Standard 169, we have not attempted to certify at Level 2. Since the Certification group has not tested DEC Standard 169 or DEC Multinational, the group is not yet prepared to make a statement about the product's compatibility with those standards.

What follows is a list of the actual deviations placed in categories as a result of the Waiver Panel meeting. The list is included here in order to document what changes were agreed upon in order to gain agreement that the PRO 350 may be called a Level 1 product.

CATEGORY 1: CHANGE THE PRODUCT

For items in this category, it is agreed that the problem must be fixed, but that because plans are in place to do so by a specified time, a waiver is granted.

1) <ESC> [? 8 h - Autorepeat

The function keys Backspace, Linefeed, and Tab do not autorepeat. The architecture states that all keys except RETURN should autorepeat.

2) <ESC> [c - Device Attribute Report

PRO 350 responds with <ESC>[?12;7;0;102c which is not the proper response for a LEVEL 1 device. Since software currently does not support the architectural identification scheme, it is agreed that certain device specific responses would be correct:

A) If the emulator is in PRO mode:

CSI?23;1;0;0;0c (if EBO installed)
CSI?23;0;0;0;0c (if no EBO)

B) If the emulator is in VT125 mode:

ESC[?12;7;0;102c (will always be ESC not CSI)

C) If the emulator is in VT102 mode:

ESC[?6c

D) If the emulator is in VT52 mode:

ESC\Z

3) Break Key

The Break key sends <BREAK> when pressed alone, or when pressed together with either shift or with control. Shift Break should be disconnect and Control Break should be answerback. Since answerback is not programmable in this product, control break should cause no code to be sent.

4) TAB

When auto wrap mode is set, the Horizontal Tab does not reset the LAST-COLUMN-FLAG.

5) <ESC> \ - String Terminator

ST is not handled properly with the control strings OSC (Operating System Command), DCS (Device Control String), APC (Application Program Command) and PM (Privacy Message). ST will only terminate the "<ESC>Pp" ReGIS control string, and it is inoperative for the others.

CATEGORY 2: CHANGE THE ARCHITECTURE

For items in this category, it is agreed that the product was implemented in a reasonable fashion and the architecture may need clarification.

1) <ESC> [? 8 h - Autorepeat

If auto repeat is turned off by a control function while a key is auto-repeating, two keystrokes worth of codes are transmitted. This is a timing issue, and the architecture will change to eliminate a specified requirement for the delay between receipt of autorepeat and when the function takes effect.

2) Answerback

Answerback message is not supported. The architecture will be changed to put Answerback into an appendix of documented "exceptions". This means that the function will no longer be required for conformance, and it will not be tested in future certifications.

3) <ESC> [? 7 h - Auto Wrap

When auto wrap mode is set, Erase in Display resets the LAST-COLUMN-FLAG, but according to the TIA specification it should not. The architecture was in error in this matter, and will be changed to require both Erase in Line and Erase in Display to reset the LAST-COLUMN-FLAG.

CATEGORY 3: CHANGE THE CERTIFICATION PROCESS

For items in this category, it is agreed that the certification process was in error in that it either failed to properly interpret the architecture or failed to properly test the product.

1) <ESC> [Pn @ - Insert Character

The Certification group reported that Insert Character was not implemented. After the problem was reported, the architecture was changed to no longer require this implementation in Level 1 but to require it for Level 2, and therefore the testing procedures will have to follow suit.

CATEGORY 4: WAIVE BY AGREEMENT OF THE PANEL

For items in this category, it is agreed that no change is to be made at this time to the product, the architecture or the certification procedures. By granting this waiver, the Panel is expressing the opinion that it considers the issue insufficient cause for the product to fail certification.

1) <ESC> [2 h - Keyboard Action Mode

Keyboard Action Mode is turned off whenever SET-UP is entered and exited.

2) Key Processing

The E1 - E5, F6 - F10, and F14 - F20 keys do not output codes but do click. They should not click.

3) <ESC> [5 m - Blink

Fields with the blink attribute do not always blink. This occurs when more than 100 fields are set to blink simultaneously.

4) <ESC> c - Reset

RIS does not control the setting of Auto Wrap mode.

CATEGORY 5: RESTRICT

For items in this category, it is agreed that although no change to the product is involved at this time, the deviation is serious enough to warrant placing a restriction on what compatibility or other claims the product should be allowed to make.

There were no items in this category.

CATEGORY 6: NO RESOLUTION

For items in this category, the Waiver Panel was unable to agree on a resolution for the issue or on the proposed time frame for resolution, and will present it to upper management for a decision.

There were no items in this category.

C.5 VT240/VT241 Terminal / 14-NOV-83

The Certification Waiver Panel met on 11-NOV-83 to review the results of VT240/VT241 Certification. Each deviation that was listed in the Certification Report of 10-NOV-83 was discussed and agreement was reached in every instance as to how the deviations would be resolved. Because of this, the opinion of the Waiver Panel is that VT240/VT241 has PASSED CERTIFICATION WITH WAIVERS as a Level 2 Device.

One waiver was granted on the basis that the product would be changed to eliminate the deviations between it and the architecture (see section 1.0 below). A commitment was made to fix the problem by the next generation release.

What follows is a list of the actual deviations placed in the categories as a result of the Waiver Panel meeting. The list is included here in order to document what changes were agreed upon in order to gain agreement that the VT240/VT241 may be called a Level 2 product.

CATEGORY 1: CHANGE THE PRODUCT

For items in this category, it is agreed that the problem must be fixed, but that because plans are in place to do so in a future release, a waiver is granted.

1) Insert Line / Delete Line

Attempting to INSERT or DELETE a line while the cursor is positioned below the bottom margin should be ignored by the terminal. The VT240 will crash if this is attempted. Due to the fact that this attempt would normally be ignored, software should avoid this condition.

CATEGORY 2: CHANGE THE ARCHITECTURE

For items in this category, it is agreed that the product was implemented in a reasonable fashion and the architecture may need clarification.

1) EXIT VT52 MODE leaves the terminal in LEVEL 1.

Sending the escape sequence to EXIT VT52 MODE will leave the terminal in LEVEL 1 mode even if the terminal had been set to LEVEL 2 mode prior to entering VT52 mode. The architecture will be changed to reflect this functionality.

CATEGORY 3: CHANGE THE CERTIFICATION PROCESS

For items in this category, it is agreed that the certification process was in error in that it either failed to properly interpret the architecture or failed to properly test the product.

There were no items in this category.

CATEGORY 4: WAIVE BY AGREEMENT OF THE PANEL

For items in this category, it is agreed that no change is to be made at this time to the product, the architecture or the certification procedures. By granting this waiver, the Panel is expressing the opinion that it considers the issue insufficient cause for the product to fail certification. As noted in the descriptions below the panel reccomends that certain of these be considered for fix in a future release of the product.

1) Pressing or Releasing SHIFT can cause multiple returns to be sent.

If the SHIFT key is pressed or released while the RETURN key is held down, a RETURN code will be transmitted by the terminal. This is a problem with the LK201 that cannot be solved by the terminal firmware.

2) ESC or CSI embedded in escape sequences is not always handled correctly.

The SRM calls for ESC and CSI to abort escape sequences in progress. If an ESC or CSI is sent to the terminal during the processing of a DEC PRIVATE control sequence, the sequence is not always aborted correctly. Attempts were made to fix this problem, but Certification testing was unable to verify that all instances of this problem had been corrected. Attempts will be made to both fix and test the problem in a future release.

3) LEVEL 2 parameters to SELECT GRAPHIC RENDITION (SGR) are valid in LEVEL 1.

The SRM specifies that SGR parameters 22,24,25,and 27 are LEVEL 2 functions. If LEVEL 2 SGR parameters are sent to the terminal while in LEVEL 1 mode, they will be acted upon. Due to the possibility of causing product compatiblity problems, the product will not be required to change the implementation in a future release. Software should not depend on these functions working in LEVEL 1, as other implementations will function as specified.

4) DECUDK allows keys to be downloaded in LEVEL 1.

The DECUDK control is specified to be a LEVEL 2 function. The terminal allows the function keys to be downloaded, but the keys are inactive until the terminal is placed into LEVEL 2 mode. Due to the possibility of causing compatiblity problems, the product will not be required to change the implementation in a future release. Software should not depend on these functions working in LEVEL 1, as other implementations will function as specified.

5) Device Status Request(DSR) for UDK Lock status works in LEVEL1.

The DSR for UDK Lock status is specified to be a LEVEL 2 function. The terminal repsonds to the request while the terminal is operating in LEVEL 1 mode. Due to the possibility of causing compatiblity problems, the product will not be required to change the implementation in a future release. Software should not depend on these functions working in LEVEL 1, as other implementations will function as specified.

6) DECTCEM works in LEVEL 1.

The DECTCEM control is specified to be a LEVEL 2 function. The terminal allows the cursor to enabled/disable while the terminal is operating in LEVEL 1. Due to the possibility of causing compatiblity problems, the product will not be required to change the implementation in a future release. Software should not depend on these functions working in LEVEL 1, as other implementations will function as specified.

7) Character 10/0 goes to the printer as SUB.

The SRM specifies the code 10/0 shall be sent to the printer as UNDERSCORE. The terminal currently sends the SUB code to the printer. Attempts will be made to fix the problem in a future release.

8) Undefined characters from GR are sent to the printer incorrectly.

The SRM specifies that undefined multinational codes be sent to the printer as UNDERSCORE. The terminal sends the character code to the printer instead of the underscore character. DEC printers that support DEC Multinational will correctly print the codes as 'REVERSE QUESTION MARK'. Attempts will be made to fix the problem in a future release.

9) IF DEC supplemental is in GL, SPACE is printed as 10/0.

Use of DEC supplemental in this way is prohibited by DEC Standard 169. The cosequences of the terminal behavior are not fully understood. Attempts will be made to fix the problem in a future release.

CATEGORY 5: RESTRICT

For items in this category, it is agreed that although no change to the product is involved at this time, the deviation is serious enough to warrant placing a restriction on what compatibility or other claims the product should be allowed to make.

There were no items in this category.

CATEGORY 6: NO RESOLUTION

For items in this category, the Waiver Panel was unable to agree on a resolution for the issue or on the proposed time frame for resolution, and will present it to upper management for a decision.

There were no items in this category.

C.6 VAXstation Workstation / 10-JUL-84

The Certification Waiver Panel met on 10-JUL-84 to review the results of VAXstation Certification. Each deviation that was listed in the Certification Report of 10-JUN-84 was discussed and agreement was reached in every instance as to how the deviations would be resolved. Because of this, the opinion of the Waiver Panel is that VAXstation has PASSED CERTIFICATION WITH WAIVERS as a Level 2 Device.

Ten waivers were granted on the basis that the product would be changed to eliminate the deviations between it and the architecture (see CATEGORY 1 below). A commitment was made to fix the problem by the next generation release.

What follows is a list of the actual deviations placed in the categories as a result of the Waiver Panel meeting. The list is included here in order to document what changes were agreed upon in order to gain agreement that the VAXstation may be called a Level 2 product.

CATEGORY 1: CHANGE THE PRODUCT

For items in this category, it is agreed that the problem must be fixed, but that because plans are in place to do so in a future release, a waiver is granted.

1. There is no implementation of the BLINK graphic rendition. (fix planned for update release)

2. <ESC>[?8h or <ESC>[?8l will not set or reset auto repeat Auto repeat is set via user profile, allowing it session wide. (fixed planned for , January 1985 release)

3. SUB does not have a representation in Level 1 mode. It correctly displays the backward question mark in Level 2 mode. (fix planned for update release)

4. If the terminal is set to Level 2, 8 bit controls (via S8C1T), and then Level 1 is selected (via DECSCL), the terminal is not reset to 7 bit controls. The Arrow keys and the Keypad Function keys still send 8-bit codes. (fix planned for update release)

5. With New Line Mode set, ENTER sends CR instead of CR LF and CTRL/M sends CR LF instead of CR. (fix planned for update release)

6. A Level 2 VT52 configuration is possible. This occurs if VT52 mode is selected on a 7-bit terminal (via terminal settings), and then the 8-bit multinational terminal setting is selected. Everything about the Level 2 VT52 is like a normal VT52, with the exception of the keyboard, which sends Level 2 codes for the Arrow keys and the Keypad Function keys. (fix planned for update release)

7. SHIFT/CAPS LOCK mode is not implemented. (fixed planned for v2.0, January 1985 release)

8. According to the SRM, ANSWERBACK should be a minimum of 20 characters for a Level 1 device and 30 characters for a Level 2 device. On the vaxstation, a maximum of 20 characters are allowed, the number depending on the size of the character. Therefore, most of the time, less than 20 characters must be used in answerback. (fix planned for update release)

9. KAM (keyboard action mode) sequences are ignored. The keyboard cannot be locked/unlocked with the ESC[2h/ESC[2l sequences. (fix planned for V2.0 January 1985 release)

CATEGORY 1: CHANGE THE PRODUCT (cont'd)

10. If a key is held down and is autorepeating (a, for example), and then CTRL is held down the autorepeating stops, and the appropriate control code (CTRL/A in this instance) may OR may not be sent. Release of the CTRL key results in continuation of autorepeat of the original character (a). (fix planned for update release)

CATEGORY 2: CHANGE THE ARCHITECTURE

For items in this category, it is agreed that the product was implemented in a reasonable fashion and the architecture may need clarification.

There were no items in this category.

CATEGORY 3: CHANGE THE CERTIFICATION PROCESS

For items in this category, it is agreed that the certification process was in error in that it either failed to properly interpret the architecture or failed to properly test the product.

There were no items in this category.

CATEGORY 4: WAIVE BY AGREEMENT OF THE PANEL

For items in this category, it is agreed that no change is to be made at this time to the product, the architecture or the certification procedures. By granting this waiver, the Panel is expressing the opinion that it considers the issue insufficient cause for the product to fail certification. As noted in the descriptions below the panel reccomends that certain of these be considered for fix in a future release of the product.

1. SCROLLING MODE can not be altered, therefore <ESC>[?4h or <ESC>[?4l are ignored. Smooth scroll is not supported on the Vaxstation.

2. Invalid control combinations and dead keys (F6-F10, F14, F17-F20, E1-E6 in Level 1) keyclick. They should not.

3. The TAB key does not autorepeat. It should.

CATEGORY 5: RESTRICT

For items in this category, it is agreed that although no change to the product is involved at this time, the deviation is serious enough to warrant placing a restriction on what compatibility or other claims the product should be allowed to make.

There were no items in this category.

CATEGORY 6: NO RESOLUTION

For items in this category, the Waiver Panel was unable to agree on a resolution for the issue or on the proposed time frame for resolution, and will present it to upper management for a decision.

There were no items in this category.

C.7 VT220 Video Terminal / 27-Oct-84

The Certification Waiver Panel met on 17-Jul-84 to review the results of VT220 Certification. Each deviation that was listed in the Certification Report of 2-JUL-84 was discussed and agreement was reached in every instance as to how the deviations would be resolved. Because of this, the opinion of the Waiver Panel is that VT220 has PASSED CERTIFICATION WITH WAIVERS as a Level 2 Device.

Five waivers were granted on the basis that the product would be changed to eliminate the deviations between it and the architecture (see page C-2). A commitment was made to fix the problem should there be a next generation release.

What follows is a list of the actual deviations placed in the categories as a result of the Waiver Panel meeting. The list is included here in order to document what changes were agreed upon in order to gain agreement that the VT220 may be called a Level 2 product.

CATEGORY 1: CHANGE THE PRODUCT

For items in this category, it is agreed that the problem must be fixed, but that because plans are in place to do so in a future release, a waiver is granted.

1. CTRL-M transmits CR LF rather than CR when new line mode is set.

2. Erase in Display from beginning of display to active position does not reset the line to single width when sent from the last column of a double width line.

3. One compose sequence for the \ (backslash) character, while using the UK keyboard in NRC mode, is missing. This sequence is COMPOSE / / . The other two sequences for backslash (COMPOSE / < and COMPOSE < /) are present.

4. If a character other than a valid character is included within a DECUDK sequence, the key definition involved may be ignored, depending upon the position of this character within the definition. If the character is within the "Kyn" portion of the definition (before the "/"), that key's definition is ignored. If the character appears after the "/", the bad character is ignored and the definition is assigned correctly. The SRM states that the result of including any invalid characters is undefined, but recommends that these characters be ignored for future compatibility.

5. When the terminal is set to eight bit printer port, the VT220 translates ESC Fe sequences to 8 bit CSI's rather than passing the 7 bit ESC['s. The terminal should pass the 7 bit ESC[.

CATEGORY 2: CHANGE THE ARCHITECTURE

For items in this category, it is agreed that the product was implemented in a reasonable fashion and the architecture may need clarification.

2. Top row function keys (F6-F20) are inoperative when pressed with the CTRL key and Editing keys (Find, Insert Here, etc.) are inoperative when pressed with either the SHIFT or CTRL keys.

The architecture currently states that these keys should not be affected by CTRL and/or SHIFT. It will be changed to reflect that the keys should be inoperative when pressed with CTRL and/or SHIFT.

3. Select Graphic Rendition allows Level 2 parameters (22, 24, 25, 27) in Level 1 mode.

The architecture will be changed to include a statement that Level 2 parameters (22, 24, 25, 27) may be recognized in level 1 mode, but that conforming software shall not rely on this feature.

4. According to the SRM, "Only the U.S. keyboard provides alternative forms for keying the C0 controls...". The VT220 provides the alternative forms on all of the foreign keyboards, where applicable .

A statement will be added to the architecture to clarify that these alternate forms are provided on non-US keyboards whenever the keys involved in these control combinations are present.

5. There are 2 additional compose sequences which are not included in the SRM, but are implemented by the terminal. They are: Section Symbol - Compose S! Compose !S, and Backslash - Compose /< and Compose </.

These control sequences will be added to the architecture.

6. In VT100 mode, multinational, with British keyboard, typing the British pound key generates and displays "#". This key should be dead in level 1 mode since it represents an 8 bit DEC Supplemental character.

The architecture will indicate that there is an exception to this rule for the British keyboard.

CATEGORY 3: CHANGE THE CERTIFICATION PROCESS

For items in this category, it is agreed that the certification process was in error in that it either failed to properly interpret the architecture or failed to properly test the product.

There were no items in this category.

CATEGORY 4: WAIVE BY AGREEMENT OF THE PANEL

For items in this category, it is agreed that no change is to be made at this time to the product, the architecture or the certification procedures. By granting this waiver, the Panel is expressing the opinion that it considers the issue insufficient cause for the product to fail certification. As noted in the descriptions below the panel reccomends that certain of these be considered for fix in a future release of the product.

1. Pressing or Releasing SHIFT can cause multiple returns to be sent.

   If the SHIFT key is pressed or released while the RETURN key is held down, a RETURN code will be transmitted by the terminal. This is a problem with the LK201 that cannot be solved by the terminal firmware.

2. Under some circumstances more than one BELL ring occurs when there should only be one.

   Again, this is a problem with the LK201 that cannot be solved by the terminal firmware.

3. Text Cursor Enable Mode (TCEM) is allowed in Level 1 mode.

   The DECTCEM control is specified to be a LEVEL 2 function. The terminal allows the cursor to enabled/disable while the terminal is operating in LEVEL 1. Due to the possibility of causing compatiblity problems, the product will not be required to change the implementation in a future release. Software should not depend on these functions working in LEVEL 1, as other implementations will function as specified.

CATEGORY 5: RESTRICT

For items in this category, it is agreed that although no change to the product is involved at this time, the deviation is serious enough to warrant placing a restriction on what compatibility or other claims the product should be allowed to make.

There were no items in this category.

CATEGORY 6: NO RESOLUTION

For items in this category, the Waiver Panel was unable to agree on a resolution for the issue or on the proposed time frame for resolution, and will present it to upper management for a decision.

There were no items in this category.

VIDEO SYSTEMS REFERENCE MANUAL

Documented Exceptions

Document Identifier: A-DS-EL00070-0D-0 Rev. AX11, 18-Mar-1985

ABSTRACT: This appendix contains the specification of terminal features which are considered exceptions to the architecture. As exceptions, they are not required of conforming hardware implementations, and will not be used by conforming software. However, when implemented, they should be implemented according to this standard in order to insure consistency of behavior across products.

APPLICABILITY: SARA Formal Cross-Component Standard. Mandatory for Engineers designing hardware for terminal products and Software Engineers designing programs using terminal interfaces. Additional requirements are defined in the section on "Concepts and Conformance Criteria".

STATUS: FOR REVIEW ONLY

This document has been placed in the SARA "Formal Cross-Component Standard" category. The material contained within this document is assumed to define mandatory standards unless it is clearly marked as (a) not mandatory or (b) guidelines. Material which is marked as "not mandatory" is considered to be of potential benefit to the corporation and should be followed unless there are good reasons for non-compliance. "Guidelines" define approaches and techniques which are considered to be good practice, but should not be considered as requirements. The procedures for modifying or evolving this standard are contained within the contents of this document.

FOR INTERNAL USE ONLY

TITLE: VIDEO SYSTEMS - Documented Exceptions

DOCUMENT IDENTIFIER: A-DS-EL00070-0D-0 Rev. AX11, 18-Mar-1985

REVISION HISTORY: Revision AX10, 15-May-83

FILES: User Documentation EL070SD.mem

Internal Documentation EL070SD.rno
EL070SD.rnt
EL070SD.rnx

Document Management Group: Terminal Interface Architecture

Responsible Department: Video Architecture Group
Responsible Individual: Peter Sichel

ACCEPTANCE: This document has been approved by the Manager of the Video Architecture Group based on a comprehensive review of its individual sections by the members of the SARA component groups working Terminal Interface Architecture issues. The list of individuals on the review and approval list are on file in Standards and Methods Control.

Direct requests for further information to Peter Sichel, PKO3-1/10C, DTN 223-5162, RANI::VIDARCH

Copies of this document can be ordered from:

Standards and Methods Control
APO1/F7, DTN 289-1414, JOKUR::SIMONETTI

CONTENTS

D.1 Introduction

The specifications contained in this appendix comprise a registry of exceptions to the architectural structure, which are intended to serve as guidelines to hardware implementors. These features are not required for conformance, and they will not be tested as a part of the certification process. It is recommended, however, that when implemented they be implemented in their entirety, and as closely as possible to the descriptions provided herein.

Software is warned against the use of any of these functions, since they are not required and are not likely to be implemented in all products.

D.2 Reset to Initial State

RESET TO INITIAL STATE RIS

Purpose: Reset the terminal to power-up condition.

Format:

ESC	c
1/11	6/3

Description: When this control is received all state in the terminal is set to their default values.

Notes:

1. Conforming software will not use the RIS control, but should use the Soft Terminal Reset (DECSTR) in Level 2 operation to put the terminal into a known state.

2. The default values of a certain state in some implementations may be set and stored under local control by the terminal user (e.g., Set-Up and non-volatile RAM or a disk) as well as remotely. In such implementations RIS will set the storable state to the values selected by the terminal user. Therefore software cannot rely on the values of this state after an RIS is executed. The values given in the algorithm below apply only to those implementations in which the state can not be set and stored locally by the terminal user. The state defined by the architecture to which this indeterminate condition may apply is limited to the following:

```
CONFORMANCE_LEVEL: (LEVEL_1,LEVEL_2);
COLUMN_MODE: (EIGHTY,ONE_THIRTY_TWO);
SCROLLING_MODE: (JUMP,SLOW);
SCREEN_MODE: (NORMAL_SCREEN,REVERSE_SCREEN);
AUTO_WRAP_MODE: (WRAP_OFF,WRAP_ON);
TEXT_CURSOR_ENABLE_MODE: (TEXT_CURSOR_OFF,TEXT_CURSOR_ON);
INSERT_REPLACEMENT_MODE: (REPLACE,INSERT);
NEW_LINE_MODE: (NEW_LINE_OFF,NEW_LINE_ON);
HORIZONTAL_TAB_STOPS: ARRAY [MAX_NUM_COLUMNS] OF BOOLEAN;
AUTO_REPEAT_MODE: (REPEAT_ON,REPEAT_OFF);
CURSOR_KEY_MODE: (CURSOR,CK_APPLICATION);
KEYPAD_MODE: (NUMERIC,KP_APPLICATION);
CAPS_SHIFT_LOCK_MODE: (CAPS_LOCK,SHIFT_LOCK);
AUTO_PRINT_MODE: (AUTO_PRINT_OFF,AUTO_PRINT_ON);
PRINT_FORM_FEED_MODE: (PRINT_FF_OFF,PRINT_FF_ON);
PRINT_EXTENT_MODE: (PRINT_DISPLAY,PRINT_SCROLLING_REGION);
GRAPHICS_CURSOR_ENABLE_MODE:
  (GRAPHICS_CURSOR_OFF,GRAPHICS_CURSOR_ON);
```

3. The DRCS and UDK definitions are also cleared in Level 2 operation.

State Affected:

```
CONFORMANCE_LEVEL: (LEVEL_1,LEVEL_2);
IN_USE_TABLE: IN_USE_TABLE_TYPE;
DESIGNATED_GRAPHIC_SETS:
  ARRAY [G0..G3] OF GRAPHIC_CHARACTER_SET_TYPE;
IN_CONTROL_STRING: BOOLEAN;
C1_TRANSMISSION_MODE: (SEVEN_BIT,EIGHT_BIT);
SINGLE_SHIFT: (NONE,SS2,SS3);

TOP_MARGIN: LINE_TYPE;
BOTTOM_MARGIN: LINE_TYPE;
CURRENT_RENDITION: CHARACTER_RENDITION_TYPE;
LINE_RENDITION: ARRAY [LINE_TYPE] OF LINE_RENDITION_TYPE;
COLUMN_MODE: (EIGHTY,ONE_THIRTY_TWO);
SCROLLING_MODE: (JUMP,SLOW);
SCREEN_MODE: (NORMAL_SCREEN,REVERSE_SCREEN);
ORIGIN_MODE: (ABSOLUTE,DISPLACED);
AUTO_WRAP_MODE: (WRAP_OFF,WRAP_ON);
TEXT_CURSOR_ENABLE_MODE:
  (TEXT_CURSOR_OFF,TEXT_CURSOR_ON);
INSERT_REPLACEMENT_MODE: (REPLACE,INSERT);
NEW_LINE_MODE: (NEW_LINE_OFF,NEW_LINE_ON);
DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
CURSOR_SAVE_BUFFER: SAVE_BUFFER_TYPE;
HORIZONTAL_TAB_STOPS: ARRAY [COLUMN_TYPE] OF BOOLEAN;
LAST_COLUMN_FLAG: BOOLEAN;

KEYBOARD_ACTION_MODE: (UNLOCKED,LOCKED);
AUTO_REPEAT_MODE: (REPEAT_ON,REPEAT_OFF);
CURSOR_KEY_MODE: (CURSOR,CK_APPLICATION);
KEYPAD_MODE: (NUMERIC,KP_APPLICATION);
CAPS_SHIFT_LOCK_MODE: (CAPS_LOCK,SHIFT_LOCK);
COMPOSE_MODE: (COMPOSE_OFF,COMPOSE_ON);

AUTO_PRINT_MODE: (AUTO_PRINT_OFF,AUTO_PRINT_ON);
PRINT_FORM_FEED_MODE: (PRINT_FF_OFF,PRINT_FF_ON);
PRINT_EXTENT_MODE:
  (PRINT_SCROLLING_REGION,PRINT_DISPLAY);

GRAPHICS_CURSOR_ENABLE_MODE:
  (GRAPHICS_CURSOR_OFF,GRAPHICS_CURSOR_ON);
REGIS_GRAPHICS_MODE: (REGIS_OFF,REGIS_ON);
SIXEL_GRAPHICS_MODE: (SIXEL_OFF,SIXEL_ON);
```

Algorithm:

```
PROCEDURE RESET_TO_INITIAL_STATE;
VAR       Y: LINE_TYPE;
          X: COLUMN_TYPE;
BEGIN
CASE SERVICE_CLASS OF                          (* see Note 1 *)
  61: CONFORMANCE_LEVEL:= LEVEL_1;
  62: CONFORMANCE_LEVEL:= LEVEL_2;
  END;
IN_USE_TABLE.C0:= ASCII_C;
IN_USE_TABLE.GL:= ASCII_G;
IN_USE_TABLE.INVOKED_GL:= G0;
IN_USE_TABLE.C1:= SUPPLEMENTAL_C;
IN_USE_TABLE.INVOKED_GR:= G2;
CASE CONFORMANCE_LEVEL OF
  LEVEL_1:
    BEGIN
    IN_USE_TABLE.GR:= ASCII_G;
    C1_TRANSMISSION_MODE:= SEVEN_BIT;
    END;
  LEVEL_2:
    BEGIN
    IN_USE_TABLE.GR:= SUPPLEMENTAL_G;
    C1_TRANSMISSION_MODE:= EIGHT_BIT;
    END;
  END;
SINGLE_SHIFT:= NONE;
IN_CONTROL_STRING:= FALSE;
TOP_MARGIN:= 1;
BOTTOM_MARGIN:= MAX_NUM_LINES;
DESIGNATED_GRAPHIC_SETS[G0]:= ASCII_G;
DESIGNATED_GRAPHIC_SETS[G1]:= ASCII_G;
CASE CONFORMANCE_LEVEL OF
  LEVEL_1:
    BEGIN
    DESIGNATED_GRAPHIC_SETS[G2]:= ASCII_G;
    DESIGNATED_GRAPHIC_SETS[G3]:= ASCII_G;
    END;
  LEVEL_2:
    BEGIN
    DESIGNATED_GRAPHIC_SETS[G2]:= SUPPLEMENTAL_G;
    DESIGNATED_GRAPHIC_SETS[G3]:= SUPPLEMENTAL_G;
    END;
  END;
CURRENT_RENDITION[BOLD]:= FALSE;
CURRENT_RENDITION[UNDERSCORE]:= FALSE;
CURRENT_RENDITION[BLINK]:= FALSE;
CURRENT_RENDITION[REVERSE]:= FALSE;
COLUMN_MODE:= EIGHTY;                          (* see Note 1 *)
SCROLLING_MODE:= SLOW;                         (* see Note 1 *)
SCREEN_MODE:= NORMAL_SCREEN;                   (* see Note 1 *)
ORIGIN_MODE:= ABSOLUTE;
```

```
AUTO_WRAP_MODE:= WRAP_OFF;                              (* see Note 1 *)
TEXT_CURSOR_ENABLE_MODE:= TEXT_CURSOR_ON;               (* see Note 1 *)
INSERT_REPLACEMENT_MODE:= REPLACE;                      (* see Note 1 *)
NEW_LINE_MODE:= NEW_LINE_OFF;                           (* see Note 1 *)
FOR Y:= 1 TO MAX_NUM_LINES DO
  BEGIN
  FOR X:= 1 TO MAX_NUM_COLUMNS DO
    DISPLAY[Y,X].CODE:= EMPTY_CHARACTER;
  LINE_RENDITION[Y]:= SINGLE_WIDTH;
  END;
ACTIVE_POSITION.LINE:= 1;
ACTIVE_POSITION.COLUMN:= 1;
HORIZONTAL_TAB_STOPS[1]:= FALSE;              (* see Note 1 *)
FOR X:= 2 TO MAX_NUM_COLUMNS DO
  IF X MOD 8 = 1 THEN HORIZONTAL_TAB_STOPS[X]:= TRUE
  ELSE HORIZONTAL_TAB_STOPS[X]:= FALSE;                 (* see Note 1 *)
LAST_COLUMN_FLAG:= FALSE;
SAVE_CURSOR;
KEYBOARD_ACTION_MODE:= UNLOCKED;
AUTO_REPEAT_MODE:= REPEAT_ON;                           (* see Note 1 *)
CURSOR_KEY_MODE:= CURSOR;                               (* see Note 1 *)
KEYPAD_MODE:= NUMERIC;                                  (* see Note 1 *)
CAPS_SHIFT_LOCK_MODE:= CAPS_LOCK;                       (* see Note 1 *)
COMPOSE_MODE:= COMPOSE_OFF;
AUTO_PRINT_MODE:= AUTO_PRINT_OFF;                       (* see Note 1 *)
PRINT_FORM_FEED_MODE:= PRINT_FF_OFF;                    (* see Note 1 *)
PRINT_EXTENT_MODE:= PRINT_DISPLAY;                      (* see Note 1 *)
GRAPHICS_CURSOR_ENABLE_MODE:= GRAPHICS_CURSOR_ON;       (* see Note 1 *)
REGIS_GRAPHICS_MODE:= REGIS_OFF;
SIXEL_GRAPHICS_MODE:= SIXEL_OFF;
DEVICE_STATUS:= READY;
END;
```

Known Deviations:

Execution of the RIS control on some older terminals may cause the terminal self-test to be performed, which takes several seconds and flashes test patterns on the display. It may also cause a modem disconnect to occur. Therefore conforming software is recommended not to use this control.

D.3 Answerback

D.3.1 Description

The Answerback feature provides the terminal with the capability of transmitting a stored message (string of control and graphic codes) to the host port, either on a prompt received from the host port, or by request from the keyboard. Answerback is intended to provide a communications function, and should never be used as a "programmable function key".

Devices which implement the Answerback capability should provide a user definable message of not less than 20 characters for Level 1 devices and not less than 30 characters for Level 2 devices. The Answerback message buffer should contain any combination of control and graphic character codes (7-bit codes only for Level 1 devices, and 8-bit codes for Level 2 devices). The Answerback message will be transmitted by the terminal on receipt from the host of an ENQ (0/5) control code, typing of the Here Is key at the terminal keyboard, or 500ms after a full-duplex modem connection is established with auto-answerback enabled.

(Note: When a Here Is key is not provided on the keyboard, the Control Key in combination with the Break Key should be used as a fallback to provide this function.)

The ability to erase the Answerback message buffer or enter new characters into the buffer is a Setup function. The buffer is empty by factory default. The buffer is a FIFO, and any unfilled positions remain empty. Characters are transmitted from the buffer exactly as if they were typed at the keyboard, and are subject to the same timing and rate limiting restrictions.

```
MAX_POSITION = 20 (* or more for Level 1 *);
               30 (* or more for Level 2 *);

ANSWERBACK_MESSAGE_BUFFER:
  ARRAY [1..MAX_POSITION] OF CODE_TYPE;

ANSWERBACK_MESSAGE_SIZE: 1..MAX_POSITION;
```

D.3.2 Control Function

ANSWERBACK ENQ

Purpose: Cause the terminal to transmit a stored string of coded characters.

Format: ENQ

0/5

Description: Receipt of the ENQ control code causes the terminal to transmit the coded character string stored in the Answerback message buffer. Characters are transmitted up to the first empty character position in the buffer.

1. The number and value of character codes in the buffer is a Setup function, determined by the User.

State Affected: None

Algorithm:

```
PROCEDURE ANSWERBACK;
VAR       N: 1..MAX_POSITION;
BEGIN
FOR N:= 1 TO ANSWERBACK_MESSAGE_SIZE DO
 WRITE (HOST_PORT,ANSWERBACK_MESSAGE_BUFFER[N]);
END;
```

D.4 Device Test and Status

TEST DECTST

Purpose: Invoke the device self-tests.

Format:	CSI	Ps	y	default Ps: 0
	9/11	Ps	7/9	

Description: The DECTST control is provided as a means of invoking device specific diagnostic tests, and subsequently determining the device status by means of the Device Status Report (DSR) control. The sequence consists of a parameter string indicating a series of product specific tests to be performed. The architecture does not specify the nature or duration of any of the tests performed. The minimal implementation of this control is to to execute the Reset to Initial State (RIS) control, and to return a "Device Ready" indicator by default.

Notes:

1. Since the minimal implementation does not require the execution of any tests, and since the nature of the tests performed is not defined, software should not depend on the effects of using this control. In all probability execution of this control in any product will have an adverse effect on the state of the device, and after execution of this control the state of the device should be considered to be UNDEFINED. NO RECOVERY IS GUARANTEED WHEN THIS CONTROL IS EXECUTED.

2. Product literature should be consulted for a description of any test sequences which might be performed on a specific product.

State Affected: UNDEFINED

Algorithm:

```
PROCEDURE TEST;
BEGIN
RESET_TO_INITIAL_STATE;
(*
  Execute product specific tests as indicated
  by the parameter value.
*)
END;
```

D.5 Send/Receive Mode

SET/RESET SEND/RECEIVE MODE SRM

Purpose: Cause all data transmitted at the terminal interface to be echoed back into the terminal.

Set Format:	CSI	12		h
	9/11	3/1	3/2	6/8
Reset Format:	CSI	12		l
	9/11	3/1	3/2	6/12

Description: The SRM control is used to affect the state of Send/Receive Mode in the terminal. Send/Receive Mode provides a means of looping all control and graphic data transmitted on the host port back into the receiver. All data received by the terminal when this mode is set (Echo_On) will be interpreted as if transmitted directly from the host. The transmitted data will not be affected by the setting of this mode.

```
SEND_RECEIVE_MODE_TYPE = (ECHO_OFF,ECHO_ON);

SEND_RECEIVE_MODE: SEND_RECEIVE_MODE_TYPE;
```

Algorithm:

```
PROCEDURE SET_SEND_RECEIVE_MODE;
BEGIN
SEND_RECEIVE_MODE:= ECHO_ON;
END;

PROCEDURE RESET_SEND_RECEIVE_MODE;
BEGIN
SEND_RECEIVE_MODE:= ECHO_OFF;
END;
```

D.6 Auto Wrap Mode

D.6.1 Description

Auto Wrap Mode provides the ability to select wrapping of lines when the right line boundary is reached while entering characters into the display [set state = wrap_on]. This state applies only to entering data, and does not affect movement of the cursor using any of the cursor positioning functions or control codes. When auto-wrapping is not enabled [reset state = wrap_off], attempts to enter characters beyond the right line boundary are handled by overwriting the character at the last column in the line.

```
AUTO_WRAP_MODE: (WRAP_OFF,WRAP_ON);
```

A special condition occurs when writing a Space or graphic character into the last column position of a line while Auto Wrap Mode is set. Normally the Active Position is advanced immediately after a character is entered into the display. The only exception to this is when a character is entered into the last column position of a line, in which case the Active Position cannot be advanced. When Auto Wrap Mode is set, however, writing a character into the last position in a line causes a special flag (referred to as the Last Column Flag) to be set. This flag is checked each time a character is entered into the display, and if Auto Wrap Mode is set, will cause the Active Position to advance to the first column of the next line PRIOR to entering that character. The flag is then reset.

The Last Column Flag should also be reset by executing any control which potentially changes the Active Position from the end-of-line position. These controls include the following:

```
SET_TOP_AND_BOTTOM_MARGINS
SINGLE_WIDTH_LINE
DOUBLE_WIDTH_LINE
DOUBLE_HEIGHT_LINE_TOP
DOUBLE_HEIGHT_LINE_BOTTOM
SET_MODE (COLUMN_MODE, ORIGIN_MODE)
RESET_MODE (COLUMN_MODE, ORIGIN_MODE, AUTO_WRAP_MODE)
INSERT_OR_REPLACE_CHARACTER
CURSOR_UP
CURSOR_DOWN
CURSOR_FORWARD
CURSOR_BACK
CURSOR_POSITION
HORIZONTAL_VERTICAL_POSITION
BACK_SPACE
```

HORIZONTAL_TAB
LINE_FEED
VERTICAL_TAB
FORM_FEED
CARRIAGE_RETURN
SUBSTITUTE
INDEX
REVERSE_INDEX
NEXT_LINE
ERASE_CHARACTER
DELETE_CHARACTER
INSERT_CHARACTER
ERASE_IN_LINE
SELECTIVE_ERASE_IN_LINE
DELETE_LINE
INSERT_LINE
ERASE_IN_DISPLAY
SELECTIVE_ERASE_IN_DISPLAY

Furthermore, the state of the Last Column Flag should be saved when a Save Cursor operation is performed, and restored when a Restore Cursor operation is performed.

It should be noted that existing products vary widely in their handling of the end-of-line condition in regards to resetting the Last Column Flag.

D.6.2 Control Function

SET/RESET AUTO WRAP MODE DECAWM

Purpose: Change the state of Auto Wrap Mode between Wrap Off (reset) and Wrap On (set).

Set Format:	CSI	?	7	h
	9/11	3/15	3/7	6/8
Reset Format:	CSI	?	7	l
	9/11	3/15	3/7	6/12

Description: The DECAWM controls are used to change the state of Auto Wrap Mode between Wrap Off (reset state) and Wrap On (set state). (For a complete description of Auto Wrap Mode, see the section "State Descriptions")

State Affected:

```
AUTO_WRAP_MODE: (WRAP_OFF,WRAP_ON);
```

Algorithm:

```
PROCEDURE SET_AUTO_WRAP_MODE;
BEGIN
AUTO_WRAP_MODE:= WRAP_ON;
END;


PROCEDURE RESET_AUTO_WRAP_MODE;
BEGIN
AUTO_WRAP_MODE:= WRAP_OFF;
END;
```

D.7 Control Representation Mode

A device may provide a mode of operation in which all received control and graphic codes are displayed visually, and their normal execution is ignored or redefined. This mode is provided for test purposes and is not to be used by conforming software. Although Control Representation Mode is defined in the ANSI standard X3.64 for use with the Set Mode and Reset Mode control functions, it is recommended that within Digital it be implemented only as a locally selected function, and not be accessible from the host.

CONTROL_REPRESENTATION_MODE: (DISPLAY_OFF,DISPLAY_ON);

In the set state (Display_On) all control functions (except the Reset Mode control sequence for Control Representation Mode) are displayed as graphic characters. Thus all character codes (128 in 7-bits and 256 in 8-bits) will be displayed with unique representations. No control functions will be executed, with the following exceptions:

- LF - Display the graphic character for LF, and then execute a new line (CR LF).
- FF - Display the graphic character for FF, and then execute a new line (CRLF).
- VT - Display the graphic character for VT, and then execute a new line (CRLF).
- XOFF - Display the graphic character for XOFF, and suspend transmission from the terminal to the host (its normal function).
- XON - Display the graphic character for XON, and resume transmission from the terminal to the host (its normal function).
- RM - Reset Mode, if it contains the parameter for CRM (which is 3), will cause CRM to be reset, and control characters will resume their normal actions. If RM does not contain the parameter for CRM, its characters are displayed as graphic characters.

In the reset state (Display_Off) the terminal resumes its normal operation.

Notes:

1. When set, this control function causes NUL and DEL to be displayed, so these characters may not be used as fill characters
when CRM is set.

2. Control functions already received and processed by the terminal are not affected by setting Control Representation Mode,
and will not be displayed. Control functions displayed while Control Representation Mode is set will not be processed
again after the mode is reset.

3. While Control Representation Mode is set G2 and GR will be
designated and invoked to be the DEC Supplemental set, regardless
of the existing state of designation or invocation. The C1
characters, and any GR (DEC Supplemental) characters that are not
assigned (reserved for future standardization), will be displayed
as "hex pairs".

D.8 Screen Alignment

SCREEN ALIGNMENT DECALN

Purpose: Fill the entire display with a test pattern for alignment purposes.

Format:

	ESC	#	8
	1/11	2/3	3/8

Description: The DECALN control is used to fill the physical display with a screen alignment pattern which is implementation defined.

Notes:

1. This control is provided strictly as a maintenance aid and is not intended for use by any applications software.

State Affected:

```
DISPLAY: ARRAY [LINE_TYPE,COLUMN_TYPE] OF CHARACTER_TYPE;
LINE_RENDITION: ARRAY [LINE_TYPE] OF LINE_RENDITION_TYPE;
ACTIVE_POSITION: CHARACTER_POSITION_TYPE;
TOP_MARGIN: LINE_TYPE;
BOTTOM_MARGIN: LINE_TYPE;
CURRENT_RENDITION: CHARACTER_RENDITION_TYPE;
CURRENT_TABLE:= CHARACTER_TABLE_TYPE;
ORIGIN_MODE: ORIGIN_MODE_TYPE;
LAST_COLUMN_FLAG: BOOLEAN;
```

Algorithm:

```
PROCEDURE SCREEN_ALIGNMENT;
VAR       Y: LINE_TYPE;
          X: COLUMN_TYPE;
BEGIN
ORIGIN_MODE := ABSOLUTE;
CURRENT_RENDITION[BOLD]:= FALSE;
CURRENT_RENDITION[UNDERSCORE]:= FALSE;
CURRENT_RENDITION[BLINK]:= FALSE;
CURRENT_RENDITION[REVERSE]:= FALSE;
FOR Y:= 1 TO MAX_NUM_LINES DO
  BEGIN
  LINE_RENDITION[Y]:= SINGLE_WIDTH;
  FOR X:= 1 TO MAX_NUM_COLUMNS DO
```

```
    DISPLAY[Y,X].CODE:= EMPTY_CHARACTER;
  END;
ACTIVE_POSITION.LINE:= 1;
ACTIVE_POSITION.COLUMN:= 1;
TOP_MARGIN:= 1;
BOTTOM_MARGIN:= MAX_NUM_LINES;
LAST_COLUMN_FLAG:= FALSE;
END;
```

D.9 Local Function Keys

The use of Local Function Keys in terminals is described in the section "Keyboard Processing". Normally these keys never transmit characters at the coding interface. However, in certain implementations they may be used by internal software, such as in a Personal Computer or Workstation. In these cases, the following codes are assigned for their use.

Control sequences designated for these keys are intended only for use as an internal interface, and should be ignored by conforming application software.

The use of Shift and/or Control Keys affect the interpretation of the Local Function Keys.

Note: In cases where the Compose function is not handled locally by the terminal, it may be necessary to treat the Compose Key as a Local Function Key so that the Compose function can be implemented at a higher level in the system. In this case, the Compose Key should transmit the code sequence CSI 10 ~ (9/11 3/1 3/0 7/14). This sequence is reserved for this use, and should not be re-assigned to any other key. Conforming software will ignore this code sequence.

In the following table, the first entry indicates the code sequence transmitted when C1 Transmission Mode is in the reset (SEVEN_BIT) state. The second entry indicates the code sequence transmitted when C1 Transmission Mode is in the set (EIGHT_BIT) state.

Since the Local Function Keys are not usually transmitted (but are usually interpreted locally by the terminal), the coding in the following table is independent of the conformance level of operation.

Codes Possibly Used by Local Function Keys

Key	Legend	All Levels	
G99	Hold Screen (normal)	1/11 5/11 3/1 3/1 7/14	(ESC [11 ~)
		9/11 3/1 3/1 7/14	(CSI 11 ~)
	Hold Screen (with Shift)	1/11 5/11 3/15 3/1 3/1 7/14	(ESC [? 11 ~)
		9/11 3/15 3/1 3/1 7/14	(CSI ? 11 ~)
	Hold Screen (with Control)	1/11 5/11 3/14 3/1 3/1 7/14	(ESC [> 11 ~)
		9/11 3/14 3/1 3/1 7/14	(CSI > 11 ~)
G00	Print Screen (normal)	1/11 5/11 3/1 3/2 7/14	(ESC [12 ~)
		9/11 3/1 3/2 7/14	(CSI 12 ~)
	Print Screen (with Shift)	1/11 5/11 3/15 3/1 3/2 7/14	(ESC [? 12 ~)
		9/11 3/15 3/1 3/2 7/14	(CSI ? 12 ~)
	Print Screen (with Control)	1/11 5/11 3/14 3/1 3/2 7/14	(ESC [> 12 ~)
		9/11 3/14 3/1 3/2 7/14	(CSI > 12 ~)
G01	Set-Up (normal)	1/11 5/11 3/1 3/3 7/14	(ESC [13 ~)
		9/11 3/1 3/3 7/14	(CSI 13 ~)
	Set-Up (with Shift)	1/11 5/11 3/15 3/1 3/3 7/14	(ESC [? 13 ~)
		9/11 3/15 3/1 3/3 7/14	(CSI ? 13 ~)
	Set-Up (with Control)	1/11 5/11 3/14 3/1 3/3 7/14	(ESC [> 13 ~)
		9/11 3/14 3/1 3/3 7/14	(CSI > 13 ~)
G02	F4 (Data/Talk) (normal)	1/11 5/11 3/1 3/4 7/14	(ESC [14 ~)
		9/11 3/1 3/4 7/14	(CSI 14 ~)
	F4 (Data/Talk) (with Shift)	1/11 5/11 3/15 3/1 3/4 7/14	(ESC [? 14 ~)
		9/11 3/15 3/1 3/4 7/14	(CSI ? 14 ~)
	F4 (Data/Talk) (with Control)	1/11 5/11 3/14 3/1 3/4 7/14	(ESC [> 14 ~)
		9/11 3/14 3/1 3/4 7/14	(CSI > 14 ~)
G03	Break (normal)	1/11 5/11 3/1 3/5 7/14	(ESC [15 ~)
		9/11 3/1 3/5 7/14	(CSI 15 ~)
	Break (with Shift)	1/11 5/11 3/15 3/1 3/5 7/14	(ESC [? 15 ~)
		9/11 3/15 3/1 3/5 7/14	(CSI ? 15 ~)
	Break (with Control)	1/11 5/11 3/14 3/1 3/5 7/14	(ESC [> 15 ~)
		9/11 3/14 3/1 3/5 7/14	(CSI > 15 ~)

D.10 Change History

D.10.1 Rev AX10 to AX11

1. Renamed screen modes NORMAL_SCREEN and REVERSE_SCREEN in code
   for better consistency.

2. Renamed character rendition from REVERSE_VIDEO to REVERSE throughout
   code for better consistency.

3. Removed paragraph describing auto-wrap as a user preference feature.
   Auto-wrap is no longer defined as a user preference feature.

4. Modified algorithm for DECALN to include effect on ORIGIN_MODE
   and CURRENT_RENDITION.

5. Removed statement that DECALN resests terminal.

Section Index

VIDEO SYSTEMS REFERENCE MANUAL -

MASTER INDEX

| Document Identifier: A-GL-EL00070-IN-0 Rev C, 14-Apr-1989

ABSTRACT: This appendix is the master index for the Video Systems Reference Manual.

APPLICABILITY: Mandatory for engineers designing hardware for terminal products and software engineers designing programs using terminal interfaces. Additional requirements are defined in DEC STD 070-1 Video Systems Reference Manual - Concepts and Conformance Criteria.

| STATUS: APPROVED 14-Apr-1989; type $ VTX SMC for expiration date.

TITLE: VIDEO SYSTEMS REFERENCE MANUAL - MASTER INDEX

DOCUMENT IDENTIFIER: A-GL-EL00070-IN Rev C, 14-Apr-1989

REVISION HISTORY: Revision A 24-Nov-1987
Revision B 29-Apr-1988 ECO #CT001
Revision C 14-Apr-1989 ECO #CTS02

Document Management Category: Terminal Interface Architecture (STI)
Responsible Department: DSG Terminals Architecture
Responsible Person: Peter Sichel
SMC Writer: Patricia Winner

APPROVAL: This document, prepared by the Desktop Systems Group, is approved for use throughout Digital.

Peter A Sichel
Peter Sichel, Desktop Systems Group

Direct requests for further information to Peter Sichel, DSG1-2/C7, DTN 235-8374, HANNAH::TERMARCH

Copies of this document can be ordered from Standards and Methods Control, $ VTX SMC, CTS1-2/D4, DTN: 287-3724, or JOKUR::SMC.

Please supply your name, badge number, cost center, mailstop, and ENET node when ordering.

MASTER INDEX

-A-

-B-

-E-

-F-

-G-

-S-

-T-

-W-

-X-

+----------+
| READER COMMENTS |
| |
| Your comments and suggestions will help Standards and Methods Control improve their services and documents. |
+----------+

Did you request this document? _____ If so, did it arrive within a satisfactory period of time? _____ Please comment.

What are your impressions of this document? Consider format, organization, completeness, readability, and illustrations.

- -- -- -- -- -- -- -- -- -- -- FOLD ON THIS LINE -- -- -- -- -- -- -- --

Did you find technical or clerical errors in this document? If so, please specify the page number(s) and the error(s).

Are the instructions for the update package clear? _____
Do you have other suggestions for improving this document?

The following information is optional:

Name _____ Mailstop _____

Department _____ Node _____

Send your comments to JOKUR::PROJECTS, or fold, staple, and send this page through interoffice mail to:

+----------+
| READERS' COMMENTS |
| STANDARDS AND METHODS CONTROL |
| CTS1-2/D4 |
+----------+