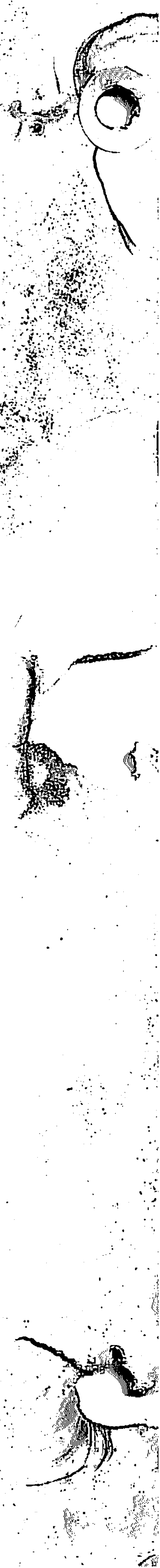
GC 1000

OPERATING INSTRUCTIONS

DOC. 85-06077-02

norpak



NORPAK

Copyright c 1982 by

Norpak Ltd.
10 Hearst Way,
Kanata, Ontario
Canada, K2L 2P4
(613) 592-4164

Norpak Product Number

85-06077-01
REV A 09-Jun-82
RELEASE A

Your comments on, corrections to, and constructive criticisms of this manual are welcomed. Please direct them to Technical Writing at the address given above.

CONTENTS

		PAGE
1	OPERATION	5
	INTRODUCTION	5
	MONITOR UNIT FEATURES	5
	Front Panel	5
	Rear Panel	6
	KEYBOARD FEATURES	8
	Text Keys	8
	Cursor Control Keypad	11
	Numeric Keypad	11
	Function Keys	11
	FLOPPY DISK HANDLING	13
	Write Protect Feature	13
	Disk Storage	13
	POWER-UP PROCEDURE	14
	POWER-DOWN PROCEDURE	14
	VT52 TERMINAL EMULATION	15
	VT52 Emulator	15
	SIMPLE LINK DATA TRANSFER PROTOCOL	16
	Introduction	16
	LOCAL Terminal	16
	Limitations	16
	Operation	17
	Using the Protocol	18
	GRAPHIC TABLET SUBPROGRAM	20
	OS-9 PASCAL INSTALLATION GUIDE	21
	Introduction	21
	Installation Instructions	22
	Multi-User Environment	26
	Single User Environment	26
2	GRAPHIC ROUTINE LIBRARY	27
	Application Programs	27
	Compilation	28
	Graphic Library:	
	Miscellaneous Subroutines	29
	INITIALIZE	29
	TERMINATE	29
	DOMAIN	29
	SET_POINT	29
	CURSOR	30
	SET_CURSOR	30
	RESET_SCREEN	30
	CLEAR	30
	Graphic Primitive Subroutines	31
	POINT	31
	LINE_REL	31
	RECTANGLE	31
	POLYGON	32
	ARC	32
	CIRCLE	32

Attribute Subroutines 33
SEL_COLOR 33
SET_COLOR 33
FILL 33
LINE_STYLE 33
FILL_TYPE 34
HIGH_LIGHT 34
LINE_WIDTH 34
Text Subroutines 35
TEXT 35
CHAR_SIZE 35
CHAR_SPACE 35
CHAR_LINE 35
CHAR_ROTATION 36
CHAR_PATH 36

APPENDIX A: SAMPLE APPLICATION PROGRAMS A-1
APPENDIX B: REMOTE/LOCAL PROTOCOL B-1
APPENDIX C: VIDEOTEX PRESENTATION LEVEL PROTOCOL C-1

ILLUSTRATIONS

Figure 1: Front View 7
Figure 2: Rear View 7
Figure 3: Keyboard and Keypads 12
Figure 4: Function Keys and Hexadecimal Codes 12

1. OPERATION

INTRODUCTION

Chapter 1 describes the monitor unit features and associated controls. The keyboard (Figure 3) is described and the purpose of each key is explained. There is a brief discussion of floppy disks for those unfamiliar with disk handling. Power-up and power-down procedures are provided. Instructions are given for running the VT52 Terminal Emulation program. A protocol for data transfer between a GC 1000 and another processor is outlined and the Graphics Tablet Subprogram is described. Finally, for users who have selected the OS-9 Pascal compiler, installation instructions are provided.

Chapter 2 outlines the subroutines in the Graphic Routine Library, and gives instructions for linking the library to BASIC and Pascal application programs.

MONITOR UNIT FEATURES

Front Panel

The front panel (Figure 1) contains five red LEDs and a Master Reset switch. From left to right, the LEDs are:

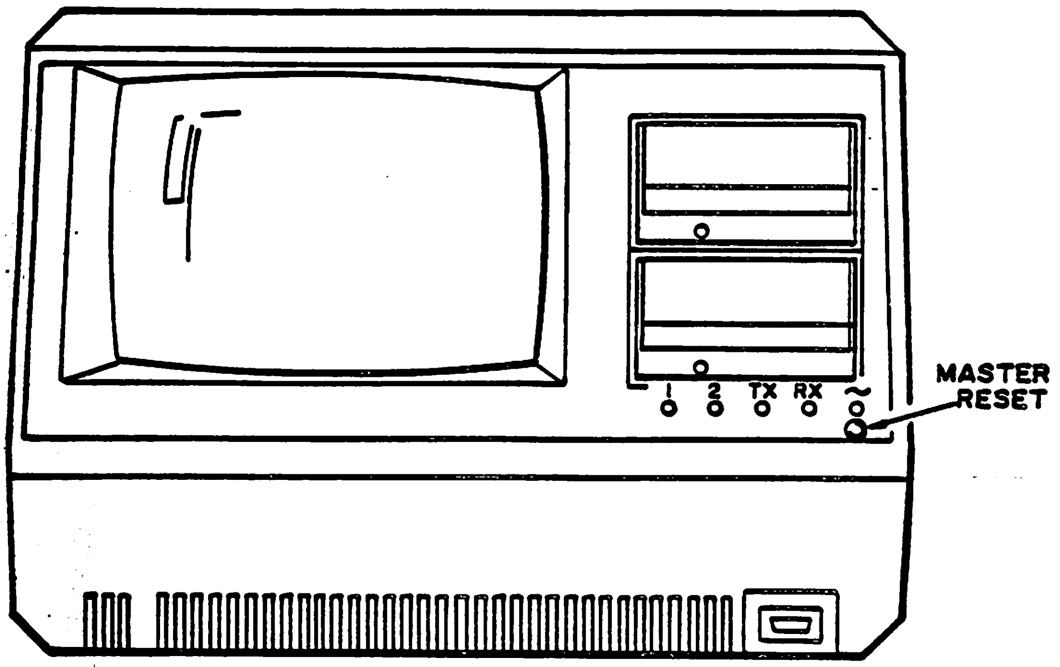
1	- indicates a memory fault
2	- indicates when OS-9 is operational by lighting briefly and then extinguishing
TX	- indicates when system is transmitting data
RX	- indicates when system is receiving data
~	- indicates that prime power is applied to the unit

MASTER RESET - push-switch that causes system to re-boot

Rear Panel

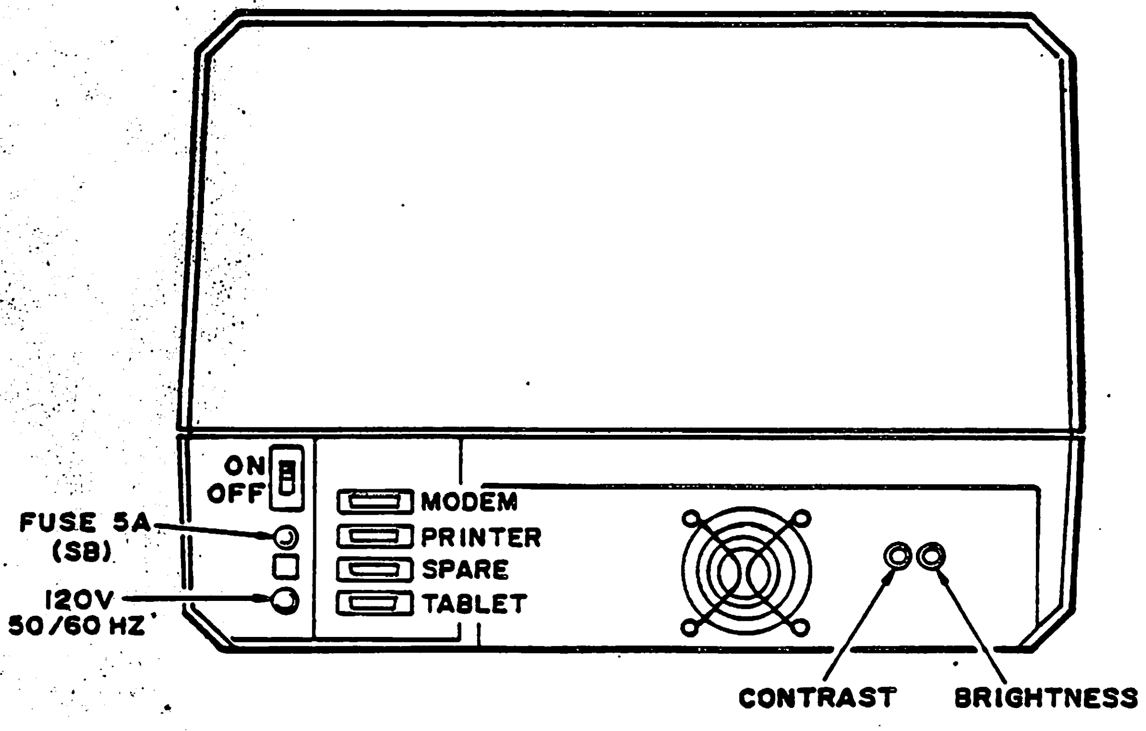
The rear panel (Figure 2) contains the following items:

ON/OFF	- power switch
FUSE 5A SB	- slow-blow fuse
MODEM	- connector for serial communications cable
PRINTER	- connector for optional printer cable
TABLET	- connector for optional tablet cable
CONTRAST	- monitor control; display contrast between black and white increases as the knob is rotated clockwise
BRIGHTNESS	- monitor control; display brightness increases as the knob is rotated clockwise.



FRONT VIEW

FIGURE 1



REAR VIEW

FIGURE 2

KEYBOARD FEATURES

Text Keys

The arrangement of keys on the GC 1000 keyboard is similar to that of an ordinary typewriter. In conjunction with the standard QWERTY keyboard, there are additional keys having specific functions, described below.

CTRL Key

The CTRL (control) key is used in conjunction with other keys on the keyboard to generate control codes. Control codes can be used as defined in the OS-9 operating system when running OS-9 programs, or can be defined in the programmer's own software.

LOCK Key

The LOCK key functions as a shift-lock. When the LOCK key is pressed, the keyboard remains shifted until SHIFT is pressed. The LED on the LOCK key lights when it is pressed.

SHIFT Key

Alphabetic keys: Pressing one or both SHIFT keys causes an upper-case code to be transmitted. When neither SHIFT key is pressed, a lower-case code is transmitted.

Non-alphabetic keys: Pressing one or both SHIFT keys causes the code for the character at the top of the key to be transmitted. When neither SHIFT key is pressed, the code for the lower character on the key is transmitted.

The SHIFT key shifts all keys having shifted functions (ie the cursor keypad, the main keyboard, the top row of keys).

Keys having no upper case function are not affected when the SHIFT key is pressed.

ESC Key

The ESC (escape) key returns the hexadecimal code 1B and is used by some programs as a termination character.

RETURN Key

The RETURN key acts as an end of line character and returns the hexadecimal code OD. It generally acts as a carriage return line feed, but depends on the software program for its function.

LINE FEED Key

The LINE FEED key moves the cursor down to the next line, and returns the hexadecimal code OA.

RUBOUT Key

When the RUBOUT key is pressed, the last character entered is deleted (erased from the screen). It returns the hexadecimal code 7F.

REPT Key

When the REPT (repeat) key is pressed, any other key pressed is repeated at the rate of 10 chars/sec.

ALPHA Key

The ALPHA key toggles the system between alphanumeric and graphics mode of operation. The default state is alphanumeric.

The following function keys are currently used only by the VT52 Emulator:

SETUP
RESET
ECHO
LINE

SETUP Key

Pressing the SETUP key allows the user to change the following:

- modem receive and transmit baud rates
- printer baud rates
- parity
- number of stop bits

To set the RX baud rate for the modem port, press R on the keyboard. Each time R is pressed, a baud rate appears. Stop pressing R when the required baud rate appears. Available RX baud rates for the modem port are 75, 150, 300, 600, 1200, 2400, 4800 and 9600.

To alter the TX baud rate for the modem port press T until the required baud rate appears. Available baud rates are 75, 150, 300 or same as RX (ie TX=RX).

To alter the baud rate for the printer port press P until the required baud rate appears. Available baud rates for the printer are 75, 150, 300, 600, 1200, 2400, 4800 and 9600.

To set the parity for the modem port, press A until the required parity appears. Parity can be odd, even or NO.

The number of stop bits for the modem port can be set at either 1 or 2 by pressing S.

ECHO Key

Pressing the ECHO key causes the character entered to appear on the screen. The default state is no echo (or echo off).

LINE Key

The LINE key toggles between on-line and local operations. When the GC 1000 is on-line, keyboard data is transmitted over the modem port to a remote computer. When the GC 1000 is in local mode, characters entered on the keyboard are only displayed on the screen.

Cursor Control Keypad

The VT52 Emulator uses the following cursor control keys:

- up arrow (↑)
- right arrow (->)
- left arrow (<-)
- down arrow (↓)
- home (*)

Pressing a specific direction key once moves the cursor one character in X or Y, depending on the direction indicated on the key.

Pressing the * (home) key causes the cursor to return to the top left corner of the screen.

RESET Key

When the RESET key is pressed, control is returned to the OS-9 operating system.

Numeric Keypad

The mathematical keys on the numeric keypad transmit special codes for the numbers 0 to 9 (hexadecimal D0 to D9). The ENTER key returns the hexadecimal code 8D). In general, the system does not distinguish between numbers entered on the keypad and those entered on the keyboard. However, software can be written to test the high-order bit on the code returned to determine whether the keypad or keyboard has been used.

Function Keys

Function keys include the top row of keys, the numeric keys and the cursor controls (a total of 45 keys). As well, the cursor control keys and the middle 15 of the top row of keys can be shifted. The function keys, their location and their hexadecimal codes are shown in Figure 4.

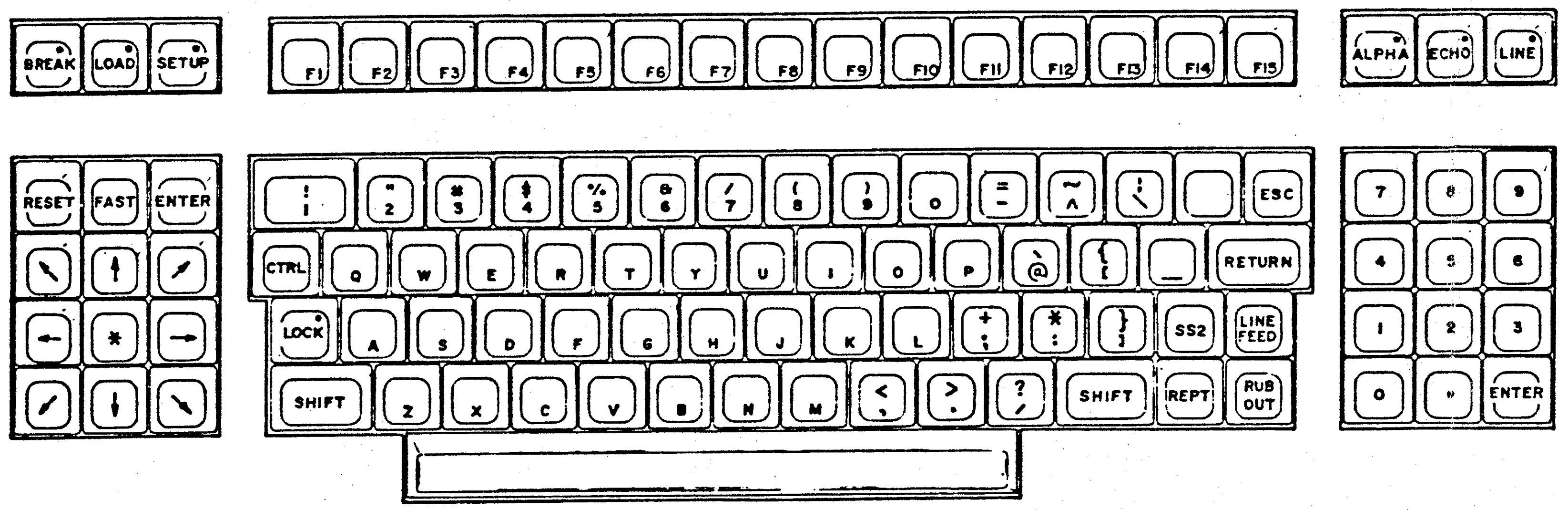


FIGURE 3 - GC/1000 KEYBOARD AND KEYPADS

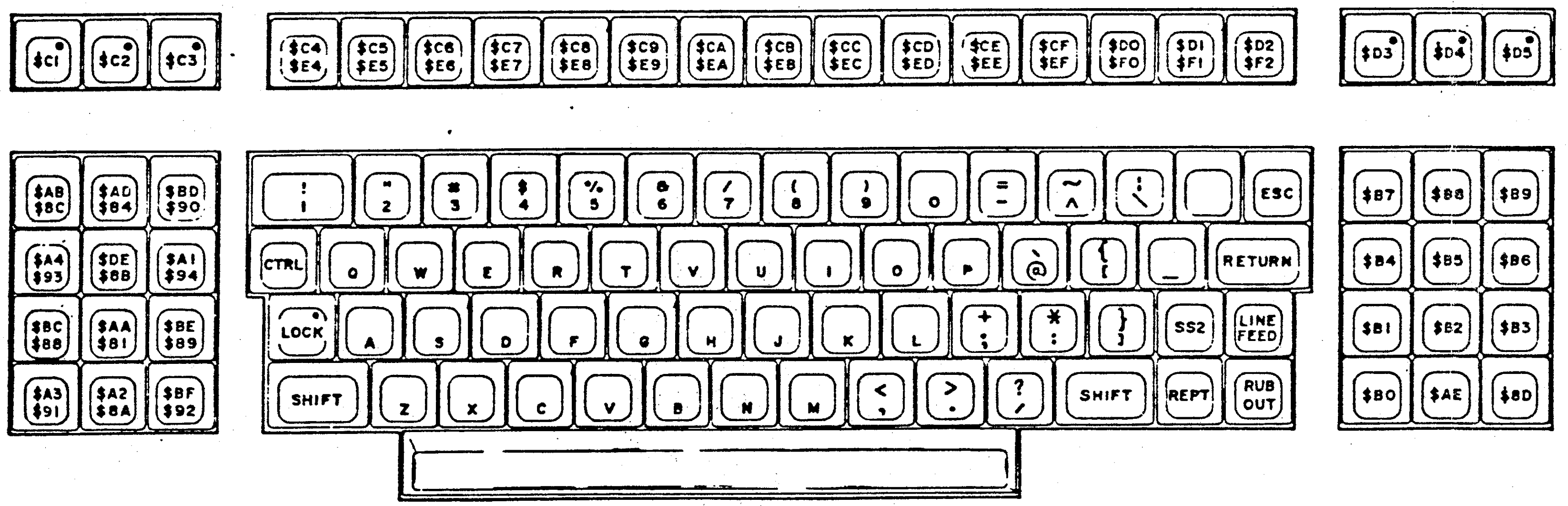


FIGURE 4 - GC/1000 FUNCTION KEYS AND HEXADECIMAL CODES

FLOPPY DISK HANDLING

The drive door is opened by pulling outward on its bottom edge. The floppy disk is then slipped into the slot with the label upwards and towards you. The edge of the floppy disk with the oval cutout in its cover must enter the drive first. The edge with the label must enter the drive last.

To insert the floppy disk, push gently until it is entirely in the drive. Close the drive door by pushing it down again.

To remove the floppy disk, open the drive door and carefully pull the floppy disk from the drive.

WARNING: Do not remove a floppy disk when the LED on the drive is illuminated. The result can be damage to the disk and its information.

Write Protect Feature

The notch on the side of the floppy disk allows it to be write protected with a write protect tab. A photo-electric sensor detects the presence of the tab on the notch and inhibits writing. When there is no write protect tab on the notch, the sensor detects the notch and allows the disk to be written to. The write protect tabs are supplied with the floppy disks.

WRITE PROTECT TAB	ACTION
Present	Read Only
Absent	Read and Write

Storage

Floppy disks must be carefully stored away from dust and extremes of temperature. They should be stored vertically in their paper pockets.

WARNING: Placing a floppy disk on a graphic tablet destroys all the information on the disk. Keep floppy disks away from all magnetic fields.

POWER-UP PROCEDURE

The power switch is located at the right rear corner of the unit. Ensure that the power cable is plugged into a standard 3-pronged 115 V outlet and that the modem (or null-modem) is connected if you intend to communicate with other systems.

- Load the OS-9 system disk in drive 0 (the upper drive)

- Press the power rocker switch to ON

- The RUN LED illuminates

- LEDs 1 and 2 will briefly illuminate and then extinguish while the system performs a memory integrity test.

- The disk drive 0 LED and LED 2 then illuminate while the system is read from disk.

NOTE: Listen for the clicks indicating that OS-9 is loading. If no clicks are heard, ensure the system disk is in the drive, as the drive LED is on whether or not a disk has been inserted.

The following display lines appear on the monitor:

Display:	Norpak Graphics Workstation (month, day, year, time)
Display:	SHELL
Prompt:	OS9:

POWER-DOWN PROCEDURE

- Remove and store the disks

- Press the power switch to OFF

VT52 TERMINAL EMULATION

VT52 Emulator Program

Before attempting to run the VT52 Emulator, ensure that the top serial interface on the back of the monitor is connected to a modem (or null-modem), which is then connected to the remote computer communications channel (host). The VT52 Emulator resides on the OS-9 system disk.

In response to the OS-9 prompt,

Type: VT52 (CR)

The host should reply with a prompt. If there is no prompt, on the keyboard,

Press: (SETUP)

The current baud rates, parity and number of stop bits appear across the bottom of the screen. Ensure that they are correct. If they are not, change them by following the instructions for the SETUP key in the Keyboard Features section under Text Keys.

Once they are correct,

Press: (SETUP)
Press: (RETURN)

The host responds with a prompt. The GC 1000 system is now in VT52 mode.

To return control to the OS-9 operating system at any time, on the keyboard

Press: (RESET)

The following VT52 features are not included in the GC 1000 VT52 Emulator software package:

- hold screen
- alternate keypad
- hardware tabs

SIMPLE LINK DATA TRANSFER PROTOCOL

INTRODUCTION

The Norpak Simple Link Data Transfer Protocol is a communications package enabling the transfer of ASCII, PDI and binary files between a GC 1000 processor and another processor (eg. a PDP-11). The package is based on two programs: LOCAL, running in the local processor and REMOTE, running in the remote processor. The local processor initiates the transfer process, allowing the user to emulate a terminal to the remote processor. When LOCAL is used in conjunction with REMOTE, the user can initiate any number of file transfers. The LOCAL and REMOTE routines control the transfers so that errors are minimized.

Refer to Appendix B for a more detailed discussion of this protocol.

NOTE: 'Local' refers to the processor to which the operator's terminal is attached and 'remote' refers to the processor at the other end of a communications link.

LOCAL Terminal

The LOCAL routine is ignorant of the actual processor and operating system it is communicating with. LOCAL scans the data coming from the remote end to detect a data transfer initiated by the REMOTE routine. Using the same operating system in both processors can result in confusing operator prompts.

NOTE: When LOCAL is operating under RSX-11M it may drop characters from a remote processor emulating a terminal, because RSX-11M LOCAL uses an AST routine to intercept data from the remote end. The communications protocol portion of the LOCAL routine is designed not to lose data because data from the remote end is always solicited.

LIMITATIONS

The operator must be responsible for some error recovery and detection. The communications system provides error detection information. The highest level of error recovery is an operator-initiated re-transmission of the file.

OPERATION

Although this protocol can be initiated by either end of the link, the local processor should always be a slave to the remote processor. The operator runs a program on the remote processor which controls the transfer of files to or from the local processor. This program requests filenames from the operator and determines the direction of the transfer.

A file transfer is initiated when a filename packet is successfully received by the local processor. This is followed by a variable number of data packets containing the file information (transmitted by either the local or remote processor, depending on the direction of transfer). Each data packet contains 63 bytes of data, except the last packet. The receiving processor detects end-of-file (EOF) when it receives a data packet with less than 63 bytes of data (including a packet with 0 data bytes if necessary). If a SEND request is received for a file which does not exist, the processor detecting the error terminates the transfer with an <EOT> <CR>.

Each time a file transfer begins, the name of the file is displayed on the local user terminal. Keyboard input is inhibited at this point, but the operator can request the current status of the transfer by pressing any key and/or can abort the transfer by pressing (Ctrl) C. Status includes the number of blocks successfully transmitted or received and the number of retransmissions due to errors (for the current file transmission). At the end of each transfer this status information is automatically displayed on the user terminal.

Using the Protocol

GC 1000 as a remote processor:

1. Connect an RS-232 cable from the GC 1000 modem port (top serial port) to the modem port of the other processor.

2. On the GC 1000, invoke the OS-9 Shell command interpreter using the following command (in response to the OS-9 prompt):

   Type: SHELL </m >/m >>/m

   The above command initiates an OS-9 process enabling the other processor to appear to be a terminal to the GC 1000. The GC 1000 can now run tasks using the REMOTE program.

GC 1000 as a local processor:

1. Connect an RS-232 cable from the GC 1000 modem port (top serial port) to a remote host.

2. Insert the VT52 Emulator disk in drive 1 and run the program according to the instructions given in the section on the VT52 Emulator.

   The GC 1000 now appears to be a terminal to the remote processor. From the GC 1000, ensure that the files to be transferred from the remote processor are in the data directory. Use the CHD command to change the data directory if necessary.

File Transfer - Local GC 1000 to remote processor:

1. In response to the OS-9 prompt,

   Type: RMO filename, ... filename
   (80 characters maximum)

   where each filename can have one or two optional qualifiers, as follows:

/A	ASCII file
/P	PDI file (default)
/B	binary file

   and

/S	send file to remote (default)
/R	receive file from remote

   The specified qualifier remains in effect until a new one is used.

2. Press any key on the keyboard to obtain the current transfer statistics while the transfer is in progress.

   To abort the transfer, press (Ctrl) C.

3. When file transfer is complete, the remote processor issues another RMO prompt. The user can then enter another list of filenames. To exit from RMO,

   Press: (Ctrl) C
   Press: (RETURN)

   GC 1000 is still a terminal to the remote processor. To exit from the Shell program,

   Press: (ESC)

   To exit from the VT52 Terminal Emulation program,

   Press: (RESET)

GRAPHIC TABLET SUBPROGRAM

The Graphic Tablet Subprogram, when used in conjunction with the Graphic Routine Library, allows the user to write an application program which accepts input from the tablet, and thereby draw the shapes involved in the creation of computer graphics.

The user must create a data area on the user disk using a data module. The module start address must be stored at memory address DDFC, DDFD (hex). The Tablet Subprogram then updates the current stylus position and status at the following offsets into the user data module:

```
1B, 1C (hex):     stylus position X (0-255)
1D, 1E (hex):     stylus position Y (0-255)
1F     (hex):     stylus status:
                  FF (hex) = no data read
                  0        = stylus up
                  1        = stylus down
```

OS-9 PASCAL INSTALLATION GUIDE

Introduction

If the user has selected model GCS-SYS-161 of the GC 1000 system, the following Pascal Installation Guide is required to run the accompanying Pascal software.

This section assumes a familiarity with running programs under the OS-9 operating system and an understanding of the use of data and execution directories. If necessary, familiarize yourself with this information before proceeding.

The following files are supplied on disk:

FILE NAME	USE
Pascal	OS-9 Pascal compiler program.
Pascal_Compiler	OS-9 Pascal compiler pcode file.
PascalErrs	Text file containing error messages.
PascalN	Nonswapping pcode interpreter.
PascalS	Virtual code swapping pcode interpreter.
PascalT_PRUN	Native code translator pcode file.
PascalT_PRUNX	Native code translator pcode file using machine code external routines for enhanced performance.
PascalT_MODL	Native code external routine module for use with PascalT_PRUNX file.
PascalE	External routine mapping program.
PascalDefs	Text file containing assembly language equates required to assemble native code programs.
Support	Full support package module.
Support1	Support package module without routines to support SIN, COS, LN, EXP, ATAN, and SQRT calls.

Support2	Support package module without the routines named in Support1 above, and also without routines to support real number addition, subtraction, and division, and calls to the standard functions and procedures: AFRAC, AINT, FILESIZE, CNVTREAL, SEEKEOF, write of a real number and read of a real number.

Several sample source programs are also included. Their file names have a "_Source" suffix.

Before attempting to use any of the above files, read the OS-9 Pascal User's Manual thoroughly. After reading the manual, follow the product installation instructions given below.

Installation Instructions

The programs supplied with the OS-9 Pascal package are briefly described below, and the names of any files required in either the execution or data directories when the program is run are given. A Pascal User's Manual is also supplied with the Pascal package.

PASCAL

Pascal is the actual OS-9 Pascal compiler program. The file Pascal_Compiler must be in the current execution directory when Pascal is run. However, if the file PascalErrs is not in the current execution directory also, the text corresponding to most error messages does not appear. The following files should all be located on the same disk within the same directory:

- Pascal
- PascalErrs
- Pascal_Compiler

When the Pascal_Compiler file is contained within a single segment on the disk, the compile time is significantly decreased. A utility program (VolStats) is supplied in source form and can be used to determine the number of segments contained in any file. This program is described in Chapter 13 of the Pascal User's Manual. Chapter 2 of the same manual describes how to run the Pascal program.

PASCALN

PascalN is the pcode interpreter program and requires:

- one of the appropriate support package modules loaded into memory before PascalN is executed

or

- the appropriate support package in the current execution directory with the name Support.

If the file PascalErrs is not also in the current execution directory, the text corresponding to most error messages does not appear. The files PascalN, Support, and PascalErrs should all be located on the same disk within the same directory. The various support package modules are discussed below. Chapter 3 of the Pascal User's Manual describes how to run the PascalN program.

PASCALS

PascalS is the virtual code swapping pcode interpreter program. Other than its code swapping function it is identical with PascalN described above. Chapter 3 of the user manual describes how to run the PascalS program.

PASCALT_PRUN

PascalT_PRUN is the optimizing native code translator pcode file and is executed using PascalS. PascalT_PRUN need not be located on a particular disk or in a particular directory to be run properly. However, the installation requirements for PascalS must be met; in particular the file PascalErrs must be in the current execution directory. PascalT_PRUN requires:

- the support package module Support or Support1 loaded into memory before PascalT_PRUN is executed

or

- one of the two support package modules existing in the current execution directory in the file Support when Pascal_PRUN begins execution.

As PascalT_PRUN is run with the virtual code swapping interpreter, it must remain online throughout its execution. Do not remove or change the disk containing the PascalT_PRUN file while the program is executing. The use of the native code translator is described in Chapter 5 of the Pascal User's Manual.

PASCALT_PRUNX

PascalT_PRUNX is essentially the same file as PascalT_PRUN except that some of the critical routines have been translated into native code to improve the execution speed of the translation process. Since the optimizing translator can perform large optimizations, some programs take a long time to run. Therefore this enhanced version is valuable on those systems with enough memory to use PascalT_PRUNX. The module containing the native code routines is contained in the file PascalT_MODL and must be either pre-loaded into memory or be in the current execution directory before PascalT_PRUNX can be run. Otherwise, the same considerations apply as for running PascalT_PRUN. This enhanced version can run several times faster than the unenhanced version. Since it requires an additional native code module in memory during its execution, it requires more memory to run than PascalT_PRUN. If the system does not have much random access memory (RAM), there may be improved performance running PascalT_PRUN.

PASCALDEFS

PascalDefs contains the assembly language EQUates required to assemble native code programs produced by the PascalT program. The PascalDefs file can reside on any drive in any directory. PascalT puts a "USE Pascal Defs" statement at the beginning of its output code. This statement must be modified to refer to the correct location of PascalDefs when the native code is assembled.

PASCALE

PascalE is the utility program which provides pcode files containing EXTERNAL routine references. These references have the path names and internal information required to properly link to the modules containing the native code versions of any external routines. PascalE requires:

- Support or Support1 loaded into memory before PascalE executes

or

- one of the two support package modules existing in the current execution directory in the file Support when PascalE begins execution.

The file PascalErrs should also be in the current execution directory. The use of PascalE is described in Chapter 5 of the Pascal User's Manual.

NOTE: Support2 does not contain the routines required to run PascalE.

SUPPORT, SUPPORT1, SUPPORT2

These files contain the runtime support routines to run any OS-9 Pascal program. The file Support requires about 9K bytes of memory and contains the full complement of support routines.

Support1 requires about 7K bytes of memory but does not support the following functions: SIN, COS, ATAN, EXP.LN and SQRT.

Support2 requires about 5K bytes of memory and does not support the following routines: calls to AFRAC, AINT, CNVTREAL, FILESIZE, SEEKEOF; routines to support real number addition, subtraction and division, reading and writing real numbers; all routines not supported by Support1.

Multi-User Environment

When any OS-9 Pascal program begins execution (except for the Pascal compiler) it first ensures that one of the support packages is in memory. If not, the file Support is loaded from the current execution directory. In a multi-user environment, if one user runs a Pascal program and causes Support2 to be loaded into memory, then other users running Pascal programs must use Support2 until it is unlinked from memory. A Pascal program assumes that the support package in memory is the required version. To prevent confusion, a rule can be enforced that the only support file used be the full function package Support. This way no one will find that a required support routine is missing when a program is executed.

Single User Environment

In a single user environment there are two ways of using the three support files effectively. The best way is to pre-load the required support file (using the OS-9 LOAD command), before program execution, and to remove that support package using the OS-9 UNLINK command on program completion. Alternatively, if one of the support packages is used more frequently than the other two, that file can be renamed Support and the other two files can also be appropriately renamed. If no support package is found in memory when an OS-9 program begins execution, the file Support in the current execution directory is loaded. If the three support packages have been renamed so that the original Support1 file is now called Support, the default support package load will then get the Support1 package. In this case the renamed file originally called Support must be manually loaded if the full support package is required.

NOTE: The OS-9 Pascal Compiler program (Pascal) does not use the support packages. The support packages need not be online or pre-loaded when compiling a source program.

2. GRAPHIC ROUTINE LIBRARY

INTRODUCTION

The GC 1000 graphic routine library (GRAPH.LIB on the master directory of the system disk) consists of subroutines which allow the application programmer to draw pictures on the display. The subroutines in the graphics library are called from programs written by the user, and follow the conventions used in the Presentation Level Protocol, Videotex Standards, described in Appendix C. This section outlines all the subroutines.

Depending upon the subroutine called, the library will interpret the arguments passed to it and construct an appropriate Picture Description Instruction (PDI) string which is then sent to the display processor. PDIs are also discussed in Appendix C.

Application Programs

The GC 1000 graphic library interfaces with both Pascal and BASIC application programs. The following explanation deals with how to run an application program written in Pascal.

In the Pascal application program, the "PROCEDURE" lines near the beginning (lines 14 to 38) are included to inform the Pascal compiler that these routines are external to the application program. In this example, all library subroutines are declared; however, only those used need be mentioned.

The "TYPE" section (lines 4 and 5) defines arrays for polygon vertices and ASCII character strings. These two lines may be omitted if neither subroutine is referenced.

A sample BASIC application program also appears in Appendix A.

Compilation

Pascal:

The compiling of the application is identical to a normal Pascal compile.

```
i.e. PASCAL < APPLICATION.PROGRAM #16K
```

Since the application program calls subroutines in the graphic routine library, the compiled application program must be linked to it. To link the program to the graphic routine library, issue the following command line:

```
PASCALE </dO/GRAPH.LINK :PCODEF #16K
```

where GRAPH.LINK is the file containing all the information required by the linker.

To run the application program, issue the following command line:

```
PASCALN PCODEF #16K
```

NOTE: The graphic library writes all its PDI information to OS-9 output path 1.

BASIC:

The graphic library can also be accessed by a BASIC program. To invoke the BASIC system consult the OS-9 BASIC manual.

When running a BASIC program, all routine calls are identical to those in Pascal. As in a Pascal routine, INITIALIZE must be the first routine called. The BASIC09 system loads INITIALIZE from the execution directory. When INITIALIZE is loaded, all other routines required are loaded also.

GRAPHIC LIBRARY:

MISCELLANEOUS SUBROUTINES

INITIALIZE;

This procedure links or loads a data module, named DF, used for necessary global data. At the same time all values are set to their defaults.

This must be the first routine called by the application program.

TERMINATE;

This procedure unlinks the data module, named DF, in preparation for the termination of the application program, and should be the last called by the application program.

DOMAIN (RES: integer);

This procedure defines the resolution (RES) of the graphics terminal. The value of RES can be in the range of 1 to 5, with a default value of 3.

The following table equates the value of RES to the screen resolution.

RES	resolution	
1	7	
2	31	
3	255	(default)
4	2047	
5	16383	

Since the RES value is required in the conversion of co-ordinate data to PDI form, it is essential that the resolution required matches the co-ordinate system used.

SET_POINT (X,Y: integer);

This procedure defines the start of the next draw command. SET_POINT is no longer in effect after completion of the draw.

The co-ordinates X,Y are interpreted as absolute screen co-ordinates.

CURSOR (ON_OFF, STYLE: integer);

This procedure allows the independent use of the cursor. The cursor will appear at a constant position on the screen, and can only be moved when the SET_CURSOR procedure is called.

The cursor may be displayed in one of two styles (0 to 3). The following table shows the relationship between the parameter for style and the type of cursor:

0 - underline cursor (default)
1 - block cursor (not implemented)
2 - cross-hair cursor
3 - custom cursor (not implemented).

SET_CURSOR (X, Y: integer);

This procedure sets the cursor to a desired position on the screen. The cursor will not move from this position until another SET_CURSOR procedure call has been made.

The co-ordinates X, Y are interpreted as absolute screen co-ordinates.

The default cursor position is at X=0, Y=0.

RESET_SCREEN;

This procedure erases the graphics screen (i.e. set to black), sets all graphic attributes to their default, sets the cursor position to X=0, Y=0, sets the current drawing position to X=0, Y=0, and turns the cursor off.

CLEAR;

This procedure erases the graphics screen (i.e. set to black).

GRAPHIC PRIMITIVES SUBROUTINES

Current Draw Position

The Current Draw Position is the starting point of the current draw command. This starting point is dependent upon the following conditions, given in order of importance:

1. if a currently outstanding SET_POINT is evident, it is taken to be the Current Draw Position of the Current Draw Command.
2. if a SET_POINT is not in effect and the cursor is on, the position of the cursor is taken to be the Current Draw Position for the Current Draw Command. The position of the cursor is not altered by the draw.
3. if 1 and 2 are not true, the Current Draw Position is taken to be the position left at after the last draw, or the last position of the cursor if this is the first draw command issued since the cursor had been turned off.

POINT;

This procedure displays a point (1 pixel) at the Current Draw Position.

Color is the only attribute affecting this subroutine.

LINE_REL (DX,DY: integer);

This procedure draws a line a relative distance (DX,DY) starting from the Current Draw Position.

The attributes affecting this procedure are: color, line-style, and line-width.

The Current Draw Position for the next draw issued is set to the end of the line if SET_POINT is not issued and the cursor is off.

RECTANGLE (DX,DY: integer);

This procedure draws a rectangle starting at the Current Draw Position, with a relative width (DX) and a relative height (DY).

The attributes affecting this procedure are: color, line-style, fill, fill-type, high-light and line-width.

If SET_POINT is not issued before the next draw and the cursor is on, the Current Draw Position will be the opposite corner along the X axis.

POLYGON (N: integer, VAR VERT: ARRAY [0..N*2] OF INTEGER)

This procedure draws a polygon from the Current Draw Position, with "N" relative vertices. The array VERT must be arranged so that element 1 is DX1, element 2 is DY1, element 3 is DX2, element 4 is DY2, etc.

The attributes affecting this procedure are: color, fill, line-style, fill-type, high-light, and line-width.

If SET_POINT is not issued before the next draw and the cursor is off, the Current Draw Position will be the starting point of the polygon.

ARC (DX1,DY1,DX2,DY2: integer)

This procedure draws an arc starting at the Current Draw Position, passing through the relative points (DX1,DY1) and finishing at the relative point (DX2,DY2).

The attributes affecting this procedure are: color, fill, line-style, fill-type, high-light, and line-width.

If no SET_POINT is issued before the next draw and the cursor is off, the Current Draw Position will be the finishing point (DX2,DY2) of the arc.

CIRCLE (DX,DY: integer);

This procedure draws a circle with a relative diameter of (DX,DY), starting at the Current Draw Position.

The attributes affecting this procedure are: color, fill, line-style, fill-type, high-light and line-width.

If SET_POINT is not issued before the next draw and the cursor is off, the Current Draw Position will be the starting point on the circle.

ATTRIBUTE SUBROUTINES

SEL_COLOR (VALUE: integer);

This procedure selects a position in the color look-up tables (LUT) and uses the color values found at this position as the current drawing color.

The required LUT entry may be referenced by setting VALUE between 0 and 15.

SET_COLOR (RED, GREEN, BLUE: integer);

This procedure re-defines the color value at the position in the LUT last specified by SEL_COLOR.

The values for Red, Green and Blue can be in the range of 0 to 255, where 0 is no color and 255 is full intensity of a given color.

FILL(ON_OFF: integer);

This procedure sets all future draws of rectangles, arcs, circles and polygons to appear filled in the current fill-type.

The value ON_OFF is 0 or 1, where 0 = off (default) and 1 = on.

LINE_STYLE (VALUE: integer);

This procedure sets the LINE_STYLE for all future draws of line, arc, rectangle, circle and polygon.

The argument VALUE may be in the range of 0 to 3 and refers to the following:

0 - Solid (default)
1 - Dotted
2 - Dashed
3 - Dot-Dashed.

FILL_TYPE (VALUE: integer);

This procedure sets the future fill-type to a specific pattern for all future draws of rectangles, arcs, polygons, and circles. However, an object will not be filled with this pattern unless the FILL procedure has been called with a value of 1 (fill on).

The argument VALUE can be in the range of 0-7 and refers to the following:

0 - Solid Fill (default)
1 - Verical Hatching
2 - Horizontal Hatching
3 - Cross Hatching
4 - System Dependent (coarse vertical cross-hatching)
5 - System Dependent (diagonal hatching [45 deg.])
6 - System Dependent (diagonal hatching [135 deg.])
7 - System Dependent (close vertical cross-hatching)

HIGH_LIGHT (ON_OFF: integer);

This procedure sets all future filled objects (rectangle, arc and polygon) to be highlighted by a solid border drawn around the object.

The argument ON_OFF can have a value 0 or 1, where 0 = off (default) and 1 = on (highlight).

LINE_WIDTH (DX,DY: integer);

This procedure defines the line width for future draws of point, line, arc, circle and polygon.

The arguments DX,DY are the width of the future draw.

TEXT SUBROUTINES

TEXT (N: integer, VAR CHARS: ARRAY [O..N] OF CHAR);

This procedure displays a string of ASCII characters, starting at the Current Draw Position.

The attributes affecting text output are: CHAR_SIZE, CHAR_SPACE, CHAR_LINE, CHAR_ROTATION, CHAR_PATH, and SEL_COLOR.

The displayable ASCII characters are in the range of 20 (hex) to 7F (hex).

If SET_POINT is not issued before the next draw and the cursor is off, the Current Draw Position is the end of the text string.

CHAR_SIZE (DX, DY: integer)

This procedure defines the size of the character field width (relative DX) and height (relative DY) for all future text output.

CHAR_SPACE (VALUE: integer);

This procedure sets the spacing between all future text characters.

The argument VALUE refers to the following:

0 - 1 (default)
1 - 1.25
2 - 1.50
3 - proportional

CHAR_LINE (VALUE: integer);

This procedure defines the spacing between text lines for all future text output.

The argument VALUE refers to the following:

0 - 1 (default)
1 - 1.25
2 - 1.50
3 - 2.0

CHAR_ROTATION (VALUE: integer);

This procedure sets the rotation of all future text output.

The argument VALUE refers to the following:

0 -	0 degrees (default)
1 -	90 degrees
2 -	180 degrees
3 -	270 degrees

CHAR_PATH (VALUE: integer);

This procedure sets the character path for all future text output.

The argument VALUE refers to the following:

0 -	Right (default)
1 -	Left
2 -	Up
3 -	Down

APPENDIX A

SAMPLE APPLICATION PROGRAMS

```
PROGRAM LIBRARYTEST;

TYPE
  POLYARY=ARRAY[1..512] OF INTEGER;
  TEXTARY=ARRAY[1..512] OF CHAR;


VAR
  CHAROUT: TEXTARY;


(* The following list of external procedures are found *)
(* in the Graphics Library.                              *)

PROCEDURE DOMAIN(RES: INTEGER); EXTERNAL;
PROCEDURE SET_POINT(X, Y: INTEGER); EXTERNAL;
PROCEDURE CURSOR(ON_OFF, STYLE: INTEGER); EXTERNAL;
PROCEDURE SET_CURSOR (X, Y: INTEGER); EXTERNAL;
PROCEDURE POINT; EXTERNAL;
PROCEDURE LINE_REL(DX, DY: INTEGER); EXTERNAL;
PROCEDURE RECTANGLE(DX, DY: INTEGER); EXTERNAL;
PROCEDURE POLYGON(N: INTEGER; VAR DXY: POLYARY); EXTERNAL;
PROCEDURE ARC(DX1, DY1, DX2, DY2: INTEGER); EXTERNAL;
PROCEDURE CIRCLE(DX, DY: INTEGER); EXTERNAL;
PROCEDURE SEL_COLOR(VALUE: INTEGER); EXTERNAL;
PROCEDURE SET_COLOR(RED, GREEN, BLUE: INTEGER); EXTERNAL;
PROCEUDRE FILL(ON_OFF: INTEGER); EXTERNAL;
PROCEDURE LINE_STYLE(VALUE: INTEGER); EXTERNAL;
PROCEDURE FILL_TYPE(VALUE: INTEGER); EXTERNAL;
PROCEDURE HIGH_LIGHT(ON_OFF: INTEGER); EXTERNAL;
PROCEDURE LINE_WIDTH(DX, DY: INTEGER); EXTERNAL
PROCEDURE RESET_SCREEN; EXTERNAL;
PROCEDURE CLEAR; EXTERNAL;
PROCEDURE TEXT(N: INTEGER; VAR CHARS: TEXTARY); EXTERNAL;
PROCEDURE CHAR_SIZE(DX, DY: INTEGER); EXTERNAL
PROCEDURE CHAR_SPACE(VALUE: INTEGER); EXTERNAL;
PROCEDURE CHAR_LINE(VALUE: INTEGER); EXTERNAL;
PROCEDURE CHAR_ROTATION(VALUE: INTEGER); EXTERNAL;
PROCEDURE CHAR_PATH(VALUE: INTEGER); EXTERNAL;


PROCEDURE PAUSE;


VAR
  CHAR_IN: CHAR;


BEGIN
  WRITE(output, 'Hit a character and return to continue: ');
  PROMPT(output);
  READLN(input, CHAR_IN);
END; (* PROCEDURE PAUSE *)
```

```
PROCEDURE TEST_POINT;


BEGIN
  SEL_COLOR(1);
  SET_COLOR(255,0,0);
  SET_POINT (50,50)
  POINT;


  SEL_COLOR(2);
  SET_COLOR(0,255,0);
  SET_POINT(70,130);
  POINT;


  SEL_COLOR(3);
  SET_COLOR(0,0,255);
  SET_POINT(190,110);
  POINT;


  SEL_COLOR(4);
  SET_COLOR(255,255,255);
  SET_POINT(140,70);
  POINT;


END;  (* PROCEDURE TEST_POINT *)


PROCEDURE TEST_LINE;


BEGIN
  SEL_COLOR(1);
  SET_COLOR(255,0,0);
  LINE_REL(90,70);


  SEL_COLOR(2);
  SET_COLOR(0,255,0);
  LINE_REL(-50,60);
```

```
    SET_POINT(120,80);
    SEL_COLOR(3);
    SET_COLOR(255,0,0);
    LINE_STYLE(0);
    LINE_REL(60,40);
    SEL_COLOR(4);
    SET_COLOR(0,255,0);
    LINE_STYLE(1);
    LINE_REL(40,-50)
    SEL_COLOR(5);
    SET_COLOR(0,0,255);
    LINE_STYLE(2);
    LINE_REL(-90,-40);
    SEL_COLOR(6);
    SET_COLOR(255,255,255);
    LINE_STYLE(3);
    LINE_REL(-10,50);

    SET_POINT(100,100);
    LINE_STYLE(0);
    LINE_WIDTH(10,10);
    SEL_COLOR(7);
    SET_COLOR(0,255,0);
    LINE_REL(50,50);

END; (* PROCEDURE TEST_LINE *)

PROCEDURE TEST_RECTANGLE;


BEGIN
    SEL_COLOR(1);
    SET_COLOR(255,0,0);
    RECTNAGLE(50,50);


    SEL_COLOR(2);
    SET_COLOR(0,255,0);
    LINE_STYLE(2);
    SET_POINT(20,100);
    RECTANGLE(50,50);


    SEL_COLOR(3);
    SET_COLOR(0,0,255);
    LINE_STYLE(0);
    FILL(1);
    SET_POINT(100,60);
    RECTANGLE(70,40);
```

```
   SEL_COLOR(4);
   SET_COLOR(255,255,255);
   FILL_TYPE(2);
   HIGH_LIGHT(1);
   SET_POINT(180,0);
   RECTANGLE(100,50);

END;  (* PROCEDURE TEST_RECTANGLE *)

PROCEDURE TEST_ARC;

BEGIN
   SEL_COLOR(1);
   SET_COLOR(255,0,0);
   ARC(50,50,50,-50);

   SEL_COLOR(2);
   SET_COLOR(0,255,0);
   LINE_STYLE(1);
   SET_POINT(0,100);
   ARC(50,-50,50,50);

   SEL_COLOR(3)
   SET_COLOR(0,0,255);
   LINE_STYLE(0);
   FILL(1);
   SET_POINT(100,100);
   ARC(50,-50,50,50);

   SEL_COLOR(4);
   SET_COLOR(255,255,255);
   FILL_TYPE(1);
   HIGH_LIGHT(1);
   SET_POINT(100,0);
   ARC(50,50,50,-50);

END;  (* PROCEDURE TEST_ARC *)
```

```
PROCEDURE TEST_CIRCLE;

BEGIN
  SEL_COLOR(1);
  SET_COLOR(255, 0, 0);
  CIRCLE(50, 50);

  SEL_COLOR(2);
  SET_COLOR(0, 255, 0);
  LINE_STYLE(3);
  SET_POINT(0, 100);
  CIRCLE(100, 0);


  SEL_COLOR(3);
  SET_COLOR(0, 0, 255);
  LINE_STYLE(0);
  FILL(1);
  SET_POINT(100, 100);
  CIRCLE(100, 0);


  SEL_COLOR(4);
  SET_COLOR(255, 255, 255);
  FILL_TYPE(3);
  HIGH_LIGHT(1);
  SET_POINT(50, 50);
  CIRCLE(50, 0);

END;  (* PROCEDURE TEST_CIRCLE *)

PROCEDURE TEST_POLYGON;

VAR
  VERT: POLYARY;
```

```
 BEGIN
   BERT[1]:=50;
   VERT[2]:=50;
   VERT[3]:=-50;
   VERT[4]:=50;
   VERT[5]:=50;
   VERT[5]:=50;
   VERT[6]:=50;
   VERT[7]:=50;
   VERT[8]:=-50;
   VERT[9]:=50;
   VERT[10]:=50;
   VERT[11]:=50;
   VERT[12]:=-50;
   VERT[13]:=-50;
   VERT[14]:=-50;
   VERT[15]:=50;
   VERT[16]:=-50;


   SEL_COLOR(1);
   SET_COLOR(255,0,0);
   FILL(1);
   FILL_TYPE(0);
   POLYGON(8,VERT);


END;  (* PROCEDURE TEST_POLYGON *)


BEGIN (* MAINLINE *)


  CHAROUT[1 FOR 10]:='POINT TEST';          (* POINT TEST *)
  RESET_SCREEN;
  TEST_POINT;
  SEL_COLOR(5);
  SET_COLOR(255,0,0);
  SET_POINT(70,170);
  CHAR_SIZE(10,10);
  TEXT(10,CHAROUT);


  PAUSE;
```

```
CHAROUT[1 FOR 9]:='LINE TEST';          (* LINE TEST *)
RESET_SCREEN;
TEST_LINE;
SEL_COLOR(8);
SET_COLOR(0,255,0);
SET_POINT(70,170);
CHAR_SIZE(10,10);
TEXT(9,CHAROUT);


PAUSE;

CHAROUT[1 FOR 14]:='RECTANGLE TEST';  (* RECTANGLE TEST *)
RESET_SCREEN;
TEST_RECTANGLE;
SEL_COLOR(5);
SET_COLOR(0,0,255);
SET_POINT(70,170);
CHAR_SIZE(10,10);
TEXT(14,CHAROUT);

PAUSE;


CHAROUT[1 FOR 8]:='ARC TEST'; (* ARC TEST *)
RESET_SCREEN;
TEST_ARC;
SEL_COLOR(5);
SET_COLOR(255,255,255);
SET_POINT(70,170);
CHAR_SIZE(10,10);
TEXT(8,CHAROUT);


PAUSE;


CHAROUT[1 FOR 11]:='CIRCLE TEST';       (* CIRCLE TEST *)
RESET_SCREEN;
TEST_CIRCLE;
SEL_COLOR(5);
SET_COLOR(255,255,0);
SET_POINT(70,170);
CHAR_SIZE(10,10);
TEXT(11,CHAROUT);


PAUSE;
```

```
   CHAROUT[1 FOR 12]:='POLYGON TEST';      (* POLYGON TEST *)
   RESET_SCREEN;
   TEST_POLYGON;
   SEL_COLOR(2);
   SET_COLOR(255,0,255);
   SET_POINT(70,170);
   CHAR_SIZE(10,10);
   TEXT(12,CHAROUT);


END.  (* MAINLINE *)



DOMAIN,/D1/Library,33,0,2,20
SET_POIN,/D1/Library,33,3,0,20
CURSOR,/D1/Library,33,6,5,20
SET_CURS,/D1/Library,33,9,0,20
POINT,/D1/Library,33,12,13,20
LINE_REL,/D1/Library,33,15,13,20
RECTANGL,/D1/Library,33,18,13,20
POLYGON,/D1/Library,33,21,13,20
ARC,/D1/Library,33,24,13,20
CIRCLE,/D1/Library,33,27,13,20
SEL_COLO,/D1/Library,33,30,2,20
SET_COLO,/D1/Library,33,33,7,20
FILL,/D1/Library,33,36,0,20
LINE_STY,/D1/Library,33,39,2,20
FILL_TYP,/D1/Library,33,42,2,20
HIGH_LIG,/D1./Library,33,45,2,20
LINE_WID,/D1/Library,33,48,0,20
RESET_SC,/D1/Library,33,51,13,20
CLEAR,/D1/Library,33,54,0,20
TEXT,/D1.Library,33,57,3,20
CHAR_SIZ,/d1/Library,33,60,13,20
CHAR_SPA,/d1/Library,33,63,0,20
CHAR_LIN,/d1/Library,33,66,0,20
CHAR_ROT,/d1/Library,33,69,0,20
CHAR_PAT,/d1/Library,33,72,0,20
```

The following is a sample BASIC program using the graphics library. The program draws a series of concentric rectangles which become progressively smaller as they approach the centre of the screen.

```
PROCEDURE BASIC_TEST
DIM I: INTEGER

PRINT CHR$(27); CHR$(82); CHR$(27); CHR$(83);
RUN INITIALIZE
RUN RESET_SCREEN

RUN SEL_COLOR(1)
RUN SET_COLOR(255, 0, 0)
RUN LINE_STYLE(0)
FOR I=0 TO 100
  RUN SET_POINT(127-I, 105-I)
  RUN RECTANGLE(I+I, I+I)
NEXT I

RUN SEL_COLOR(2)
RUN SET_COLOR(0, 255, 0)
RUN LINE_STYLE(1)
FOR I=0 TO 90
  RUN SET_POINT(127-I, 105-I)
  RUN RECTANGLE(I+I, I+I)
NEXT I

RUN SEL_COLOR(3)
RUN SET_COLOR(0, 0, 255)
RUN LINE_STYLE(2)
FOR I=0 TO 80
  RUN SET_POINT(127-I, 105-I)
  RUN RECTANGLE(I+I, I+I)
NEXT I

RUN SEL_COLOR(4)
RUN SET_COLOR(255, 255, 255)
RUN LINE_STYLE(3)
FOR I=0 TO 70
  RUN SET_POINT(127-I, 105-I)
  RUN RECTANGLE(I+I, I+I)
NEXT I

RUN SEL_COLOR(5)
RUN SET_COLOR(0, 0, 0)
RUN LINE_STYLE(1)
FOR I=0 TO 60
  RUN SET_POINT(127-I, 105-I)
  RUN RECTANGLE(I+I, I+I)
NEXT I

END
```

APPENDIX B

REMOTE/LOCAL PROTOCOL

INTRODUCTION

The following protocol assumes that only text files (ASCII files with only <CR>, <LF>, <FF> and/or <TAB> control characters) and PDI files are to be transmitted over the link. Standard ASCII control characters are used. Parity is removed from the data and all control characters plus <DEL> are translated into two-character sequences: <DLE> followed by the control character XORed with character @. All transfers operate through standard terminal drivers; therefore all control sequences require a <CR> terminator.

PROTOCOL

At startup, the local processor appears as a normal terminal to the remote processor. Each character entered by the operator is transmitted, with no error checking, to the remote end which can echo them as required. The local processor continuously scans the data received from the remote end until it recognizes a SOH character. This character signals that a data transfer is in progress and passes control to the data transfer software. Data is transmitted between processors in packets of up to 63 data bytes as follows:

```
<SOH><byte count><control byte> .. data .. <lrc1><lrc2><CR>
```

```
where   byte count is a binary number between 0 and 63
                    defining the number of data bytes
                    in this packet. If the count is in
                    the C0 set it is translated to the
                    G0 set by ORing it with @.

        control byte is a byte encoded as follows:

                     Bits     Meaning

                     0-4 = sequence number (0-31) or
                           file type depending on the
                           contents of the data type
                           field
                     5-6 = data type field
                            1 = File data
                            2 = SEND filename
                            3 = ACCEPT filename
```

lrc bytes are a longitudinal redundancy check including all bytes after the SOH but excluding the lrc. The lrc is 12 bits long and is split into 2 consecutive bytes. If either byte is in the CO set (octal 0 to octal 37) it is translated into the GO set (octal 40 to octal 177) by ORing it with @.

The data included in a packet is either a filename or file data, depending on the contents of the control byte. All packets are variable length, with the amount of data in a packet determined by the associated byte count.

After assembling a data packet, the receiving processor determines whether an error has occurred. If the packet is received successfully, an 'ack' is transmitted to the other processor (thus indicating another data transfer). If an error is detected in the packet, a 'nak' is transmitted to the other processor (thus initiating a re-transmission of the packet). An 'ack' is actually <ENQ> <CR> and a 'nak' is <DC4> <CR>.

Detectable error conditions are:

LRC error	(longitudinal sum)
Sequence error	

LRC errors force a re-transmission of the packet. Sequence errors are uncorrectable and therefore result in an abort of the current file transfer (with a message to the operator).

Sequence numbers are used to tag each packet in a file transfer. The sequence number is reset to 0 when a filename packet is received successfully and is incremented each time a packet has been transmitted successfully (modulo 32). If there is a sequence error, the receiving processor transmits an <EOT> <CR> to force the transmitting processor to abort the transfer.

The sequence number bits in the control byte of a filename packet define the type of file being transferred. Currently there are two file types:

0 - PDI
1 - ASCII

This information is used by the receiving processor as required. It is used by the RSX-11M transfer routines to create ASCII files with variable length records.

FLOW CONTROL

During data transfers, no flow control is necessary because the transmitter never sends data until the previous packet is 'acked' by the receiver. However, the same is not true when the local processor is acting as a terminal to the remote end. Therefore, the local processor can use CD1 and DC3 to achieve flow control in a standard format recognizable by operating systems such as RSX-11M and RT-11.

APPENDIX C

VIDEOTEX PRESENTATION LEVEL PROTOCOL

Videotex

The alpha-geometric Telidon system uses text characters and simple geometric shapes called primitives to define the image, so that the picture is built up in finely detailed areas rather than line by line. The geometric shapes and text characters are described by Picture Description Instructions (PDIs) which comply to an industry videotex protocol. Both the PDIs and the protocol are described in the following two sections.

Picture Description Instructions

Picture Description Instructions are concise codes used to define graphic primitives (shapes) in Telidon pictures. The codes are formed from the 7-bit, 128-character ASCII set and comply to OSI communications protocol. The user does not need any special knowledge of the protocol or the code structure. However, if further details are required they are given in the Videotex Standard referred to in the following section.

The PDIs define graphical and textual information in a concise alpha-geometric code set. The code set consists of the primitive identifier, its attributes, and numeric location data. The primitive is the graphic shape, the attribute are its color, whether it is to be drawn in outline or filled, and the text size and its rotation; the numeric location data defines the screen co-ordinates on which the primitive is to be located.

Videotex Protocol

To promote compatibility in videotex information systems the Bell System has adopted a presentation level protocol. The protocol is defined in a document that is concerned with the definition of the "formats, rules, and procedures adopted for the encoding of text, graphics, and display control information for videotex applications". The document is highly technical and will not be of interest to most users.

Should you be interested in the technical details of videotex protocol, further details can be obtained from either of the following sources.

The AT&T document title is:

Presentation Level Protocol
Videotex Standard
Bell System, May 1981,

and further information can be obtained through:

Manager, Information Management Planning
and Development
American Telephone and Telegraph Company
5 Wood Hollow Road
Parsippany NJ 07054

The DOC document title is:

Telidon Videotex Presentation Level Protocol:
Augmented Picture Description Instructions
CRC Technical Note 709

and further information can be obtained through:

Department of Communications
Information Services
300 Slater St.
Ottawa, Ontario K1A 0C8