TEK SYSTEM MANUAL

Part No. 070-5307-00
Product Group 65

6130 System

USER'S GUIDE

First Printing OCT 1984
Revised APR 1985

Tektronix®
COMMITTED TO EXCELLENCE

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the users at their own expense will be required to take whatever measures may be required to correct the interference.

Specifications subject to change.

Contents

Figures

Examples

Tables

Safety Summary

Symbols on Equipment

 ATTENTION — refer to manual.

Terms

In This Manual

CAUTION statements identify conditions or practices that can result in damage to equipment or other property.

Marked on Equipment

CAUTION indicates a personal injury hazard not immediately accessible as one reads the marking, or a hazard to property including the equipment itself.

Use the Proper Power Cord

Use only the power cord and connector specified for your product.

Use only a power cord that is in good condition.

Refer cord and connector changes to qualified service personnel.

Power Source and Ground

The 6100 and 6200 Series workstations are designed with a protective ground connection in the Tektronix-supplied power cord. A protective ground connection by way of the grounding connector in the power cord is essential for safe operation. To avoid electrical shock, plug the power cord into a properly wired outlet.

This product is designed to operate from a power source that does not apply more than 250 volts rms between the supply conductors or between either supply conductor and ground.

Use Care When Accessing Back Panel

When you access the back panel (to change boards, attach connectors, check line voltage or configuration switch settings, or whatever) FOLLOW ALL DIRECTIONS CAREFULLY. Always shutdown and unplug the system at the point and in the manner that the instructions describe.

Do Not Remove Covers or Panels

To avoid personal injury, do not remove the workstation's covers or panels, unless instructed to do so by the manual. Do not operate the workstation without the cover and panels properly installed.

1

Introduction

The 6130 Intelligent Graphics Workstation is a complete computer system in a desk-top package. The 6130 workstation is designed with computing resources to support:

- A mechanical engineer doing drafting, analysis, and solids modeling.
- An electronic engineer doing simulation and layout.
- A software engineer doing program development, execution, and maintenance.

All of these tasks require a lot of computing power, which may be at a premium in a multipurpose mainframe computer being shared by many users. If your task must wait for computing resources, you must wait for results. The 6130 Workstation lets you dedicate your computer resources to your tasks, to speed the turnaround of results.

While you may want to dedicate your computing resources to your tasks, you may also want to share some resources with larger groups. The workstation provides Local Area Network support that lets you share files and peripherals, and exchange electronic mail with other workstations and computers. The workstation also provides Network File System (NFS) support, allowing transparent access to files and commands on other NFS machines.

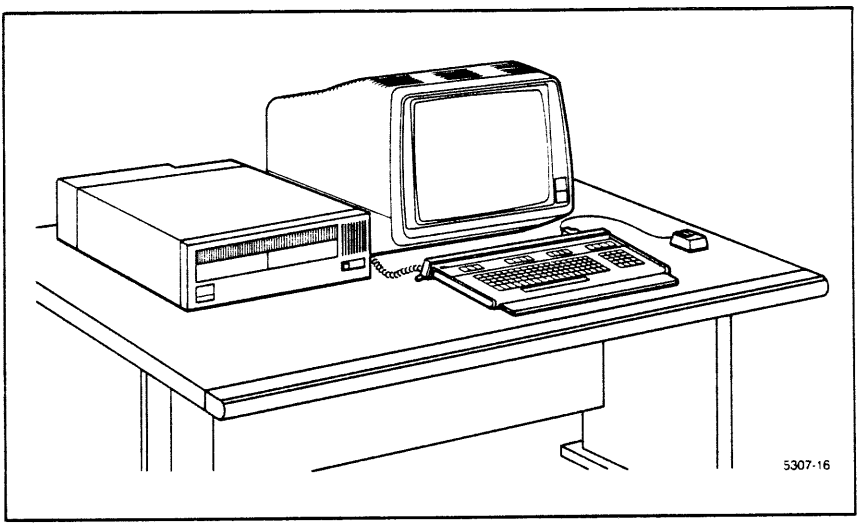

Figure 1-1. A Typical 6130 Workstation Configuration

WHAT IS THE 6130 WORKSTATION?

The 6130 workstation consists of a base configuration, workstation options, and workstation enhancements. The *base configuration* contains the standard components of the workstation. *Workstation options* can be substituted for components in the base configuration. *Workstation enhancements* can be added to expand the capability of your workstation.

Base Configuration

The base configuration of the workstation consists of a system enclosure, computer board, hard disk drive, diskette drive, and the UTek operating system.

System Enclosure

The workstation's system enclosure includes:

- Hard disk drives
- A six-slot card cage for mounting circuit boards
- A power supply capable of operating with an input voltage of 115 or 230 volts
- Interface connectors

Computer Board

The computer board provides the computing resources and input/output interfaces for the workstation. The computer board contains:

- A microprocessor
- Floating point arithmetic hardware
- 1 megabyte of random-access memory (RAM)
- A time-of-day clock with a battery
- Input/output ports (communications)

The input/output ports of the workstation provide high-level interfaces for I/O devices. The following I/O ports are located on the computer board:

- Two RS-232-C serial ports that connect terminals, printers, and modems to your workstation.
- One Local Area Network (LAN) port that lets you connect your workstation to an Ethernet LAN with other workstations and computers.
- One General Purpose Interface Bus (GPIB) port, which conforms to the IEEE 488-1980 standard, that connects programmable instruments to your workstation.

Diskette Drive

The terms *diskette* and *flexible disk* are interchangeable. The diskette drive, which mounts in the front of the system enclosure, is used to transfer files to and from the workstation and to back up and archive files. The diskette provides 360 kbytes of formatted storage.

Hard Disk Drive

The internally-mounted 5.25 inch hard disk drive stores the UTek file system. The hard disk[1] drive, which is based on Winchester technology, has 40 megabytes of formatted storage.

UTek

The workstation's UTek operating system is a multiuser, multiprocessing operating system based on UNIX.

UTek is based on University of California at Berkeley's Version 4.2 UNIX and contains features from AT&T's System V UNIX, Sun's Network File System (NFS), and utilities developed by Tektronix. For a complete list of the features of UTek, see the *UTek Command Reference* manual.

Workstation Options

The workstation can be configured with the following options:

Option 14	A 40 megabyte hard disk drive that replaces the standard hard disk drive of the workstation's base configuration.
Option 15	An 80 megabyte hard disk drive that replaces the standard hard disk drive of the workstation's base configuration.
Options A1-A4	International power cords.

Workstation Enhancements

Enhancements you can add to your workstation include a floor stand for the workstation, additional interfaces for I/O devices, memory expansion, mass storage peripherals, printers, copiers, and plotters. These enhancements are described in Section 3.

1. Because the hard disk is based on Winchester technology it is often referred to as the Winchester hard disk drive, or just the Winchester drive.

ABOUT THIS GUIDE

This manual describes the standard and optional components of the 6130 workstation and directs you to more detailed discussions in other manuals.

You should complete the learning sessions of the *6130 Learning Guide* before reading this manual. If you are not familiar with UNIX-based operating systems, you may wish to read *Introducing the UNIX System* (Chapters 1-4, 11, and 13 are recommended) before reading this manual.

This manual contains the following sections plus a Glossary and Index:

Section 1	(this section) Introduces the 6130 workstation.
Section 2	Describes the internally-mounted peripherals and interfaces that are standard on the workstation.
Section 3	Describes enhancements you can add to your workstation.
Section 4	Describes how to start up your workstation and how to shut it down.
Section 5	Gives examples of how you can create special files to tailor the Bourne Shell, C-Shell, MH mail system, vi text editor, and the **finger** command to meet your needs.
Section 6	Describes how to use the local area network, including the Network File System (NFS).
Section 7	Describes the programming languages and tools that are available for the workstation.
Section 8	Describes the workstation's implementation of the General Purpose Interface Bus (GPIB).
Section 9	Describes how to write a BASIC language program to control instruments over the General Purpose Interface Bus (GPIB).
Appendix A	Introduces General Purpose Interface Bus (GPIB) concepts.
Appendix B	Introduces the Tektronix Codes and Formats standard for GPIB instruments.
Appendix C	Contains the ASCII-GPIB code chart.

Notation Conventions

The notation conventions listed below are used throughout this manual.

<RETURN>	Special keys are shown as all capital letters, surrounded by angle brackets.
<CTRL-X>	Control characters are shown using the same notation as for special keys. Control characters are created by holding down the key labeled *CONTROL* (or *CTRL* on some keyboards) while typing the indicated key (in this example, x).
file	Filenames, directory names, pathnames, and text for which you substitute your own information when entering a command are in *italics*.
cd	Command names and text you enter exactly as it appears are in **boldface**.
grep(1)	A command followed by a parenthesized number, *command(n)*, is a reference to more information on that command in the *UTek Command Reference —Section n*.

RELATED DOCUMENTS

The following books are available from Tektronix, Inc. to help you use your workstation. Some of these documents came packaged with your workstation. To get copies of these books, contact your Tektronix Field Office.

Workstation User Manuals

- *6130 Learning Guide*
- *6130 System Administration*

UTek Manuals

- *Introducing the UNIX System*
  McGilton and Morgan
- *UTek Command Reference*
  Volumes 1 and 2
- *UTek Tools*
  Volumes 1 and 2
- *Network File System Reference Manual*
- *6130/4132 UTek Exceptions & Extensions*

Programming Language Books

- *The C Programming Language*
  Kernighan and Ritchie
- *FORTRAN 77 Reference*
- *Pascal User Manual and Report*
  Jensen and Worth
- *Tektronix BASIC Keyword Dictionary*
  Volumes 1 and 2
- *Tektronix BASIC Quick Reference*
- *Tektronix BASIC Users Guide*

Installation and Service Manuals

- *6130 System Installation*
- *6100 Series Service*
- *6130 System Diagnostics*

Peripherals and Interfaces

This section describes the hard disk drive, diskette drive, and the input/output ports that are provided in the base configuration of the workstation. Figure 2-1 shows the locations of the hard disk drive and diskette drive in the system enclosure.

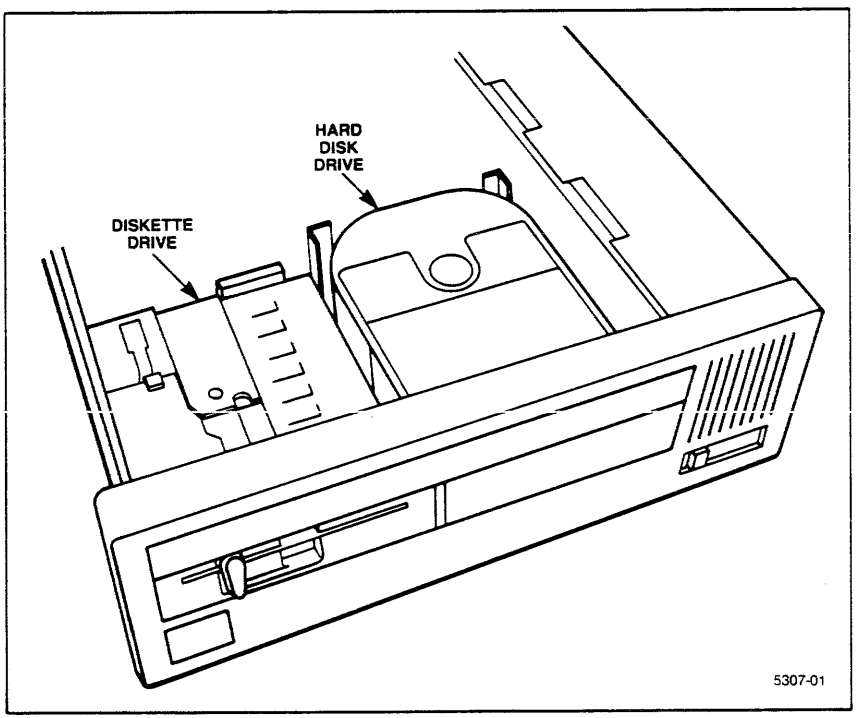

Figure 2-1. Hard Disk and Diskette Drive Locations

HARD DISK DRIVE

The UTek file system is stored on the Winchester hard disk. To save storage space on the hard disk, you can archive unneeded files to another storage medium such as diskette or tape. Archiving files to diskette with the **cpio** command is described later in this section.

To back up the hard disk on another storage medium and to rebuild the UTek operating system on the hard disk, see the *6130 System Administration* manual. For more information on the device driver that interfaces the hard disk drive to UTek, see *dwa(4)* in the *UTek Command Reference* manual.

DISKETTE DRIVE

The diskette drive is used to back up the file system, store files, transfer files to and from other workstations, and to rebuild the hard disk. To back up the UTek file system and rebuild the Winchester disk from diskettes, see the *6130 System Administration* manual.

The diskette drive mounts in the front of the system enclosure. The drive uses 5.25 inch, double-sided diskettes with a formatted storage capacity of 360 kilobytes. The diskettes are formatted at 48 tracks per inch (tpi) and are compatible with the IBM PC.

For more information on the device driver that connects the diskette drive to UTek, see *dfa(4)* in the *UTek Command Reference* manual.

Diskette Care

To protect your diskettes, you should take the following precautions:

- Return the diskette to its storage envelope when you remove it from the drive.
- Do not bend or fold the diskette.
- Store diskettes in their box.
- Do not store diskettes on top of a terminal or on top of the workstation cabinet. The heat generated by these devices can warp a diskette.
- Keep your diskettes away from all magnetic materials and devices. Strong magnetic fields destroy data on the diskette.
- Replace the diskette's storage envelope when it becomes worn, cracked, or distorted.
- Apply the diskette identification labels in the correct location on the diskette. Never apply a new identification label on top of an old identification label.
- Do not write on the plastic diskette jacket with a pencil or ball point pen. Always use a felt tip pen.
- Do not use erasers on your diskettes.
- Keep cigarette ashes away from your diskette. The heat and contamination can damage a diskette.
- Keep your diskettes away from heat and sunlight.

How to Insert a Diskette

To load a diskette into the drive:

1. Release the latch on the front panel of the drive by turning it parallel to the slot (Figure 2-2).
2. Insert the diskette face up, with its label towards the drive latch (Figure 2-3).
3. Turn the latch on the drive to allow the head to engage (Figure 2-4).

To prevent head damage, a mechanical interlock prevents you from closing the latch when no diskette is in the drive.

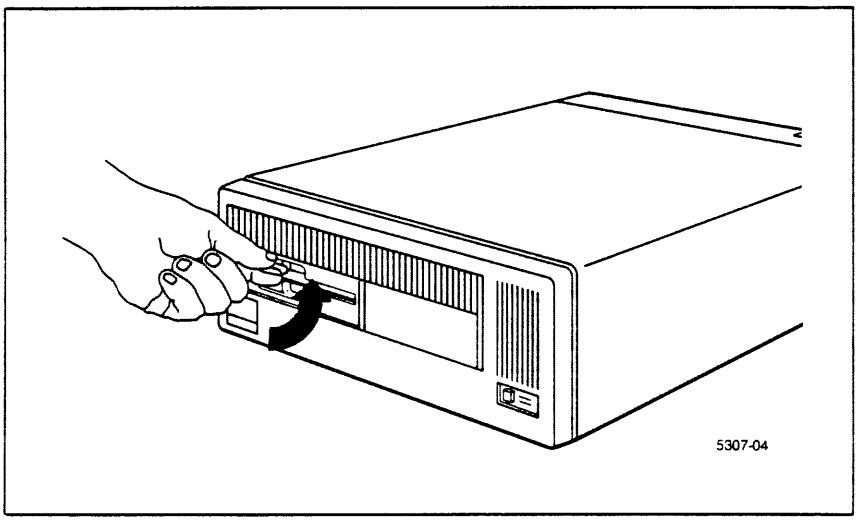

Figure 2-2. Opening the Latch on the Diskette Drive

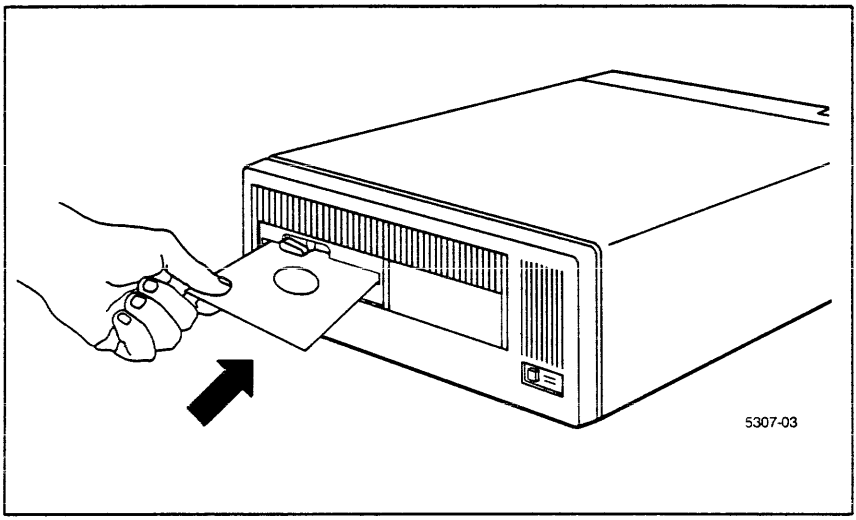

Figure 2-3. Inserting a Diskette.

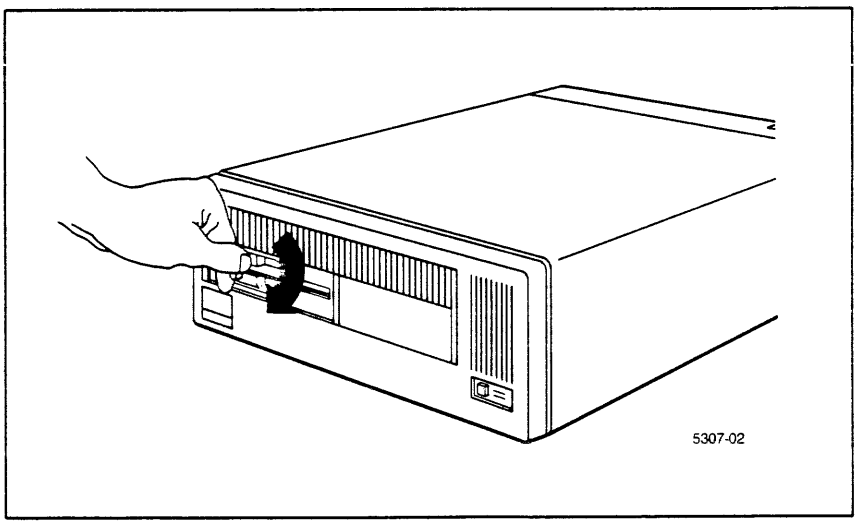

Figure 2-4. Closing the Latch on the Diskette Drive.

CHECKOUT OF THE 61KP04

The standard diagnostics package for the 6130 contains routines to verify the operation of the 61KP04. See *6130 Diagnostics* for additional information.

READING AND WRITING FILES

The following sequence of commands writes a file to tape, reads it back, and compares the file read back with the original. Log on as a normal (non-root) user.

```
assign tc
cp /vmunix foobar
tar cvbf 256 /dev/rtc foobar
rm foobar
tar xvbf 256 /dev/rtc
cmp /vmunix foobar
rm foobar
deassign tc
```

In the above procedure, a copy of the UTek kernel is made in the user's directory. The copy is written to tape, the copy is deleted, the file is read back from tape and then it is compared with the original. If the file read back in is the same, cmp will exit with no messages.

Possible Problems

If the individual components of the tape subsystem check out individually, there are only a few things that can go wrong at the system level. Most problems are likely to be configuration problems or cabling problems. You may get some of the following messages.

No such file or directory

or

Permission denied

This message indicates that tar could not open the special file specified on the command line. Either you mistyped the name, */dev/rtc*, on the tar command line or you created the special file with a different name. Make sure you use the same name on the tar command as you used when you created the special file.

Similar errors could be encountered when doing the **assign** command. If the ownership or permissions of the special file are not correct, you may not be able to write to it.

tar: /dev/rtc: No such device or address.

Nothing happened —You see this message if the tape driver is asked to open a tape special file using a non existent SCSI interface. Make sure the slot number you used with the *MAKEDEV* command agrees with the location of the 61KP04 SCSI interface.

<UTek> KP04 slot *s* drive *u*: abort .....

or

tm*su*: no tape

The device driver has found the SCSI interface but cannot talk to the tape controller or tape drive. Check the cables from the SCSI port to the external hard disk, the SCSI address of the tape controller, and make sure the external hard disk unit is turned on. If everything checks out and you still get the error, try turning the external hard disk unit off and back on.

tar: tape write error: No such device driver or operation.

Most errors from the tape controller result in this message. Check cables from the controller to the tape drive, unit number of the tape drive, and the tape drive power cable. Make sure your tape cartridge is not write-protected.

STREAMING TAPE CONSIDERATIONS

The cartridge tape drive is a streaming tape drive. It is is designed to transfer data continuously. When data is transferred in discrete blocks, the drive has to slow to a stop, reverse itself, stop again, so that it can accelerate to its designed transfer rate by the time it reaches the end of the previous data block. Clearly, the efficiency of the data transfer improves as the size of the transfer is increased. For reasonable efficiency, the various tape utilities let you specify logical records composed of many physical blocks. If you are copying single files or small directories, you should consider the use of diskettes instead of cartridge tape. You should write a simple shell script to invoke these utilities to archive and restore files or directories. This ensures that the same blocking factor and options are invoked each time.

STANDARD UTEK UTILITIES

Cpio

cpio is the preferred utility for transferring files to/from removable media including diskettes and cartridge tape. Optional software packages for UTek are distributed in **cpio** format. When you use **cpio**, you should define the logical block size if using a cartridge tape (not necessary with diskettes). Use the **-N** option to set the block size. The input/output device for **cpio** is selected by using the **-V** command option. **cpio** takes its input from standard input rather than from a command argument. Standard usage is to invoke **find** and pipe its output to **cpio**. Scripts to invoke the command are suggested.

Cpio Scripts

One of the attributes of the UTek operating system is the ability to write simple programs that execute in the shell. These programs are often called *shell scripts*. The general procedure is to create a file containing the commands to be executed along with explanatory comments. These files are typically collected in the subdirectory *bin* in your home directory. You should add the argument *$HOME/bin* to the PATH variable in your *.login* or *.profile* file. Make the file executable by entering **chmod +x filename** In all of the scripts given here, the name for the script is given in the first comment line.

Copy files to flexible diskette

```
#! /bin/sh

## wdisk
## If called without arguments copies entire current
## directory to flexible diskette in cpio format.
## Otherwise, copies selected files to flexible
## diskette in cpio format.
## NOTE: files are recovered by using rdisk

if test $# -eq ∅
then

else

fi
```

NOTE

The period is part of the **find** *command line.*

Copy a directory and its contents to cartridge tape

```
#! /bin/sh

## wtape
## This shell script uses cpio to copy all files
## in the current directory to cartridge tape.

find . -print | cpio ova -N 256 -V /dev/tc
```

NOTE

The period is part of the **find** *command line.*

List contents of a cpio diskette

```
#! /bin/sh

## ldisk
## Lists contents of a cpio diskette

cpio -itv -V /dev/df
```

List contents of a cpio tape cartridge

```
#! /bin/sh

## ltape
## Lists contents of a cpio tape cartridge

cpio -itv -V /dev/tc
```

Copy files from a cpio diskette to current directory

```
#! /bin/csh -f

## disk
## Restores files created by wdisk
## from flexible diskette. cpio format
## If called without arguments extracts
## all files from diskette.
## Else it extracts the named files.

if test $# -eq Ø
then
        cpio -idvam -V /dev/df
else
        cpio -idvam -V /dev/df $*
fi
```

Copy files from a cpio tape to current directory

```
#! /bin/csh -f

## rtape
## Restores files created by wtape
## from cartridge tape. cpio format
## If called without arguments extracts
## all files from tape.
## Else it extracts the named files.

if test $# -eq Ø
then
        cpio -idvam -V /dev/tc
else
        cpio -idvam -V /dev/tc $*
fi
```

Tar

The tar utility program works with cartridge tape and diskettes. It was written for 9-track tape media and is most useful in that environment. As with **cpio**, you need to specify the correct device and block size. The cartridge tape device can be specified by using the f key on the tar command line along with the device special file name for the tape drive. Tar allows tape records to be blocked in multiples of 512 bytes.

Scripts to Invoke tar

This script creates a tar tape of all files in the current directory:

```
#! /bin/sh

## arch
## This shell uses tar to archive all files in
## the current directory

tar cvbf 256 /dev/tc64 .
```

NOTE

The period is part of the **tar** *command.*

This script restores a single file from a tar tape:

```
#! /bin/sh

## rest1
## This shell uses tar to restore a single file
## to the current directory

if $# -ne 1
then
        echo This script expects a single argument
        exit 1
else
        tar xvbf 256 /dev/tc64 $1
fi
```

This script restores all files from a tar tape:

```
#! /bin/sh

## rest
## This shell uses tar to restore all files
## to the current directory

tar xvbf 256 /dev/tc64
```

Dump/Restore

dump and **restore** are privileged commands. You must be logged in as *root* or invoke **su** to gain root status on your workstation to use them.

The sysadmin interface knows about streaming tape drives and manages the **dump** and **restore** procedure for you. This is the recommended procedure for file system back up. The routines can also be called individually.

Dump

The **dump(8)** program can be used to make complete or incremental backups of the file system(s) on your disk(s). The **b** option specifies the number of 1 kbyte blocks in a tape record. The **f** option specifies the special device file. In the example, the name of the tape drive is */dev/tc*. Thus, to do a level zero dump, enter the following command:

```
#/etc/dump Øbfu 128 /dev/tc dwØØa
```

Restore

If you want to recover a file or group of files from a backup tape, you should use the **restore -i** option. When using the **-i** option you can traverse the file system directory tree using **cd** and **ls**. Files that you want to restore must first be marked with the **add** command, and then extracted with the **extract** command. When **add** asks for the volume number, respond with a **1**. As with **dump** you should use the **b** and **f** keys to set the block size and device name on the **restore** command line.

```
#/etc/restore ibf 128 /dev/tc
```

If your system disk is corrupted to the point where you need to do a complete file system restore, you should use the procedure described in Section 8 of the *System Administration* manual for restoring from cartridge tape. After formatting the hard disk (optional), loading the miniroot file system, and loading the UTek kernel from diskette, you are instructed to create a device special file for the cartridge tape drive with the following command:

```
/dev/MAKEDEV /dev/tcsd /dev/tc
```

Replace letters *s* and *d* by the SCSI interface slot number and the device number as described earlier.

The following command creates a file system that is equivalent to the state of the system as dumped to the cartridge tape. Unlike many single-user operating systems, UTek does NOT copy an image of the disk to the backup media or copy an image of the backup media to the system disk. It copies the directory structure and the contents of all files. **restore** recreates that directory structure and file contents. The physical disk location of files in the new file system almost certainly will differ from the original.

```
/etc/buildroot  -d  /dev/tc
```

Assign/Deassign

The **assign** and **deassign** commands let you reserve an I/O device for your exclusive use. When a device is set up using *assign*, the device is owned by the same user who owns the file */etc/assign.classes*. The protection of the device is set to read/write access by the owner only. Normally the owner is the user *daemon*. Users logged on by any other user name cannot read or write the device. The **assign** command first checks to see if the device is idle and owned by the same user who owns */etc/assign.classes*. If it is, the ownership is changed to the user doing the **assign**. That user can then read or write the device. The **deassign** command changes ownership of the device back to the owner of */etc/assign.classes*. See the *assign* and *deassign*, in the *Utek Command Reference*, for details on command parameters.

For small installations, the protection of **assign** and **deassign** may be more trouble than it is worth. If you want to make the drives usable by anyone without going through the **assign** process, you can do so by adding read/write permission for all users:

```
chmod a+rw /dev/tcsd /dev/ntcsd
```

or

```
chmod a+rw /dev/dfsd
```

INTERFACES

Your workstation provides high-level interfaces for I/O devices. Table 2-1 lists the interfaces that are provided by the base configuration of the workstation.

All the interfaces in Table 2-1 are located on the computer board. You can access the LAN, RS-232-C, and GPIB interfaces by connectors that protrude through the back panel of the workstation. These connectors are accessed by lifting the cable management cover.

Table 2-1
WORKSTATION INTERFACES

Interface	Standard	Number	Connects
Local Area Network (LAN)	IEEE 802.3	1	Local area network
Dual RS-232-C	RS-232-C	2	Terminal, printer, etc.
General Purpose Interface Bus (GPIB)	IEEE 488	1	Programmable instruments

Figure 2-5 shows the LAN, RS-232-C, and GPIB connectors on back of the workstation.

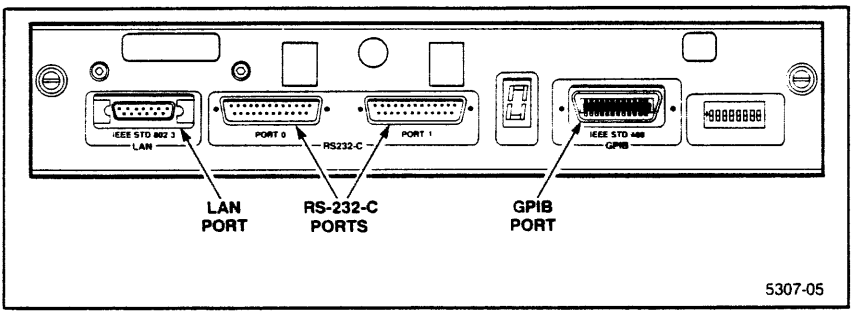

Figure 2-5. LAN, RS-232-C, and GPIB Connectors

LAN Interface

The Local Area Network (LAN) interface lets you connect your workstation to a LAN to share files and peripherals with other workstations and computer systems, including the Network File System (NFS). The LAN interface is compatible with Ethernet and conforms to IEEE Standard 802.3.

To connect your workstation to a LAN, see the documentation that came with your LAN transceiver. To configure UTek for the LAN, see the *6130 System Administration* manual.

See Section 6 for information on communicating with other machines on the network.

RS-232-C Interfaces

The two RS-232-C interfaces let you connect terminals, printers, and modems to your workstation. These interfaces transfer data to a peripheral asynchronously, in 8-bit serial format.

After you connect a peripheral to one of the RS-232-C connectors, run **sysadmin** to tell UTek how to communicate with the peripheral. The **sysadmin** command is described in the *6130 System Administration* manual.

Using Terminals with UTek

UTek supports almost any ASCII terminal, as long as its capabilities are described in the terminal capabilities database file, */etc/termcap*. To make sure UTek supports your terminal, you have to check the */etc/termcap* file. The */etc/termcap* file is composed of abbreviations, codes, and special characters, so if you have never looked in this file, you should read the description of it in *termcap(5)* in the *UTek Command Reference* manual.

One way to find which terminals are supported in the */etc/termcap* file is to use the **grep** command. Typing:

```
grep '|'/etc/termcap
```

prints one line for every terminal supported by UTek. Don't forget the quotation marks around the | character, since it is a special character.

The complete procedure for connecting a terminal to your workstation and setting it up is described in the *6130 System Administration* manual.

Connecting Your Workstation to a Modem

You can dial in to your workstation over phone lines by connecting a modem to one of the RS-232-C connectors. The procedure for setting up the modem should be described in the operator's manual for the modem. After you connect the modem to the workstation, reconfigure the RS-232-C interface by running **sysadmin** and selecting Port Configuration from the main menu.

Configuring an RS-232-C interface for a modem with **sysadmin** is described in the *6130 System Administration* manual.

General Purpose Interface Bus

The General Purpose Interface Bus (GPIB) lets you connect programmable instruments to your workstation. The GPIB conforms to the IEEE 488-1980 mechanical, electrical, and functional standards. An instrument you want to connect to the GPIB must also conform to these standards (look in its operator's manual). Among the Tektronix instruments that conform to these standards are oscilloscopes, digitizers, audio test systems, multimeters, function generators, and logic analyzers.

Section 7 describes the implementation of the GPIB on the workstation and how to write programs that control instruments connected to the workstation's GPIB. Appendix A describes the concepts of IEEE Standard 488 (GPIB). Appendix B describes the Tek Standard Codes and Formats, which are used by Tektronix GPIB instruments to communicate over a GPIB.

THE /DEV DIRECTORY

All of the peripherals connected to your workstation are associated with one or more files in the */dev* directory. If there isn't a command on your system that lets you communicate with a peripheral, you can communicate with the peripheral by reading and writing its */dev* file. Also, some commands, such as **cpio**, require you to know the name of the */dev* file associated with a peripheral.

Figure 2-6 shows an example of how files in the */dev* directory are named.

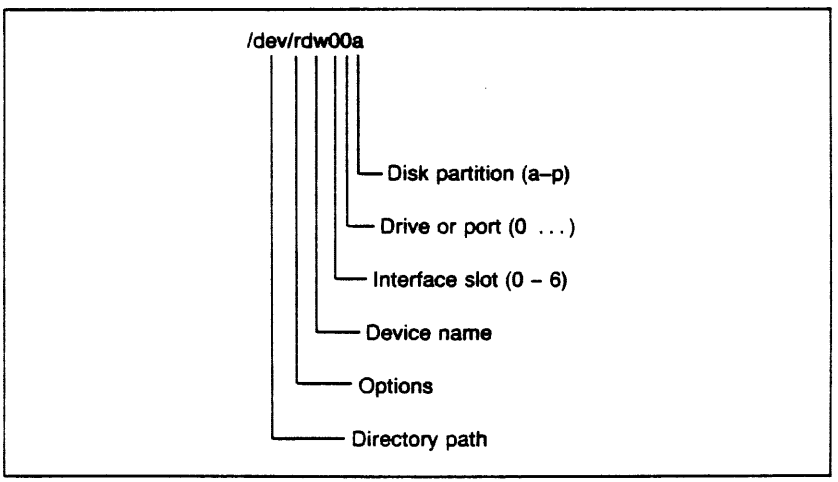

Figure 2-6. /dev Filename Conventions

The parts of a filename in the */dev* directory have the following meanings:

Directory path	All these files reside in the */dev* directory.
Options	Specifies how the device acts. In Figure 2-6, the r specifies that reads and writes to the disk are done in *raw* mode.
Device name	The abbreviation used for the device. In Figure 2-6, the device name is **dw**, which stands for Winchester disk.
Interface slot	The slot in the workstation backplane into which the interface board is plugged. The slots are numbered from 0 to 6. By convention, any devices plugged into ports provided by the computer board have an interface slot of 0. The filename in Figure 2-6 is associated with slot 0, which is the computer board.
Drive or port	The port or drive associated with this file. This is used when you have more than one of a certain device. For example, if you have two terminals, the first is terminal 0 (called *tty0*) and the second is terminal 1 (called *tty1*). In the filename in Figure 2-6, the drive is drive 0.
Disk partition	Used only for hard disk drives. Hard disk drives have one */dev* file for each segment, or *partition*, of the disk. The filename in Figure 2-6 is associated with partition a of the hard disk.

Table 2-2 lists files that are commonly found in the */dev* directory, along with the peripheral they connect.

Files in the */dev* directory that let you communicate with workstation enhancement are listed in Section 3.

Table 2-2
STANDARD FILES IN THE /DEV DIRECTORY

File	Peripheral
/dev/df	Diskette (buffered)
/dev/rdf	Diskette (raw)
/dev/tty	Your terminal, any RS-232-C port
/dev/tty00	Terminal, RS-232-C port 0
/dev/tty01	Terminal, RS-232-C port 1
/dev/ttyp0	Local Area Network
/dev/ttyp1	Local Area Network
/dev/ttyp2	Local Area Network
/dev/gpib0	GPIB interface
/dev/gpid0	GPIB configuration device
/dev/dw00a	Hard disk, partition a
/dev/rdw00a	Hard disk, raw
/dev/dw00b	Hard disk, partition b
/dev/rdw00b	Hard disk, raw
/dev/dw00l	Hard disk, partition l
/dev/rdw00l	Hard disk, raw
/dev/dw00m	Hard disk, partition m
/dev/rdw00m	Hard disk, raw
/dev/dw00n	Hard disk, partition n
/dev/rdw00n	Hard disk, raw
/dev/dw00o	Hard disk, partition o
/dev/rdw00o	Hard disk, raw
/dev/dw00p	Hard disk, partition p
/dev/rdw00p	Hard disk, raw

NOTE

The information contained in the remainder of this section is normally found in the 6130 System Administration manual, but is also included here for reference.

FORMATTING A DISKETTE

Before you can write data onto a diskette, the diskette must be in the proper format. The program that formats a diskette is **saformat**, the same program that is used to format the hard disk.

Saformat is on the *standalone utilities* diskette of the diskette distribution set. It doesn't run under UTek. You must shut the workstation down to run **saformat**. Therefore, you may find it more convenient to format a group of diskettes at one time, to be used as needed.

To format a diskette:

1. Turn the workstation off.
2. Insert the *standalone utilities* diskette (which came with the workstation) into the diskette drive.
3. Set configuration switch 5 down and switch 6 up. This selects the diskette drive as the boot device.
4. Set configuration switch 4 down. This lets you select the file you want to boot from the diskette.
5. Turn the workstation on.
6. The workstation prompts you with the following prompt:

   ```
   >>>>
   ```

7. Enter:

   ```
   df(Ø,Ø)/saformat
   ```

   This loads the **saformat** program from the standalone utilities diskette, and provides you with the menu shown in Example 2-1.

```
Drive Options

1) Quit
2) Winchester disk
3) Flexible diskette

Select by entering a number from 1 to 3:
```

Example 2-1. Saformat Top-Level Menu.

8. Select item 3 from this menu. Once you have selected *Flexible diskette*, these messages appear:

   ```
   Put the diskette to be formatted into the drive.

   Press RETURN to continue
   ```

9. Remove the standalone utilities diskette from the diskette drive and return it to its protective jacket.

CAUTION

Remember whatever data was on the diskette is lost when the diskette is formatted.

10. Insert the diskette you want to format.

11. Press <RETURN>. The menu shown in Example 2-2 displays.

```
Defaulting to 48TPI Format

      Flexible Diskette Format Command Menu

   1) Return to previous menu
   2) Quit the formatting program
   3) Select alternate disk drive
   4) Set formatting information
   5) Sweep the disk surface for defects
   6) Format the disk with the given information

Select by entering a number from 1 to 6:
```

Example 2-2. Flexible Diskette Format Command Menu.

12. Select item 6 to use the default formatting values (if you want to check the diskette for defects first, select item 5; then, when the sweep is complete, select item 6). Once you have selected this menu item, numbers are printed on your screen as cylinders are formatted.

    When the formatting is done, the Flexible Diskette Format Command menu reappears on your screen. If you want to format more diskettes, go back to Step 10. Otherwise select the menu item to quit and press <RETURN>.

INTRODUCTION TO DEVICE SPECIAL FILES

NOTE

Version 2.2 or later of UTek is required to operate the 4944 hard disk. Version 2.1 of UTek will support the 4944 optional cartridge tape. If your 6130 does not have Version 2.2, it must be upgraded before the 4944 hard disk drive can be operated. See the installation instructions that come with the Version 2.2 distribution set.

Under UTek all input/output operations are performed on files. There are no special system calls or routines allowing direct access to a program that controls disk operations. When a program requires direct access to a device such as the cartridge tape drive, the program uses a *device special file*. Device special files, or just special files, are treated much like a regular file by a program. They are opened by name, then read operations, write operations, and device control (**ioctl**) system calls are executed to transfer data, and the file is closed when the program is done. During read and write operations, the data is transferred directly to/from the device. The data is transferred to the media at its current position (there is no allocation of storage space or looking up where a file is located on the storage media). For flexible disks and cartridge tapes, the file is the entire volume. It is up to the program manipulating the volume to keep track of what is where.

Operations on special files can be identical to operations on regular files in simple cases. Thus a program that opens a file, sequentially reads it, and then closes it could work on either a regular file or a special file. There may however be some restrictions on some special files. There may also be more **ioctl** operations available on special files than on regular files. Examples of such operations are writing tape marks, skipping records or tapes marks, setting terminal port baud rates, etc. These operations would make no sense on a regular disk file.

When a special file is opened, the UTek file system must determine which device driver is to handle the requests for that file. Device drivers that handle more than one drive or port of the same type must know which unit is being referenced by the current operation. This information is encoded in two integer numbers known as the *major* and *minor device codes*. The major device code tells the UTek kernel which device driver will handle requests for the special file being processed. The minor device code is passed on to the device driver. Several pieces of information may be encoded into the 16 bits of the minor device code. For devices handled by the SCSI tape driver, referred to as the *tca driver*, the minor device code is made up of the slot number of the SCSI interface, the SCSI address of the tape controller, the unit number of the tape drive, the type of tape drive, and the density of the tape drive.

Special files are created with the shell script */dev/MAKEDEV*, which makes standard UTek special files. Device special files can be located anywhere in the UTek directory structure but by convention they are normally located in the directory */dev*. Special files can be any legal UTek file name. However, there are some conventions that can make life easier.

Device special files under UTek are named using a convention that builds the device name from the type of device, the backplane slot of the interface to the device, and the unit number or port number on the interface. The form of the device name is *xxsu*, where *xx* is a generic name for the class of device such as **tc** for tape, cartridge or **ds** for disk. The *s* is the number of the expansion backplane slot where the device interface is located. *u* is the device number for interfaces supporting multiple devices. It is computed by adding 2 times the SCSI address, to the device address which can be 0 or 1. Thus, a tape drive that is device 0 at SCSI address 2 would have a device number of (2 X 2) + 0 or 4. In some cases, there may be multiple special files for a single physical device. This is often done on tape drives to allow selection of different recording densities or for selecting whether to rewind the tape or not when the file is closed. Disk drives have separate special files for different data partitions. This allows multiple file systems to be installed on a single physical drive. Each of the multiple special files for a single device has some bits in the minor device code set differently. How the minor device code is encoded is unique for each device driver.

Device Special Files for 61TC01 and 4944

Device Special Files for Cartridge Tape

Names of special files for cartridge tape have the general form of */dev/tcsu* or */dev/ntcsu*. The *ntc* devices are the no rewind version of the *tc* devices. The *s* in the device name is the backplane slot number containing the 61KP04 SCSI interface. The *u* is a combination of the SCSI address of the tape controller and the unit number of the tape drive. The value of *u* is the SCSI address of the controller times two plus the unit number of the drive. Thus if the 61KP04 is in slot 6 of the 6130, the Cartridge Tape Controller is set to SCSI address 2, and the tape drive is internally strapped for unit 0, the recommended special device file name is */dev/tc64*. This is the default configuration as supplied by Tektronix.

Making Device Special Files for Cartridge Tape The preferred method of making special device files under Utek is to use the shell script */dev/MAKEDEV*. Given a name that follows the UTek conventions, *MAKEDEV* calculates the required minor device code and makes the special files. This driver is included in the Version 2.2 kernel so you do not need to rebuild the kernel just to support cartridge tape.

If you rebuild the UTek kernel or the *MAKEDEV* file with the *sysconf* program, you must tell *sysconf* to include the cartridge tape driver, **tca**.

After creating the device special files for the cartridge tape, you should make sure that the devices are assignable. This allows the devices to be reserved for use by a single user with the **assign** command. No one else will be able to use the cartridge tape while it is assigned. (This is not essential if you are the only user of the 6130, but it is good procedure and should be followed.) Using the UTek utility **more**, look for the following line in the file */etc/assign.classes*.

```
tape tc tc64 /dev/tc = /dev/tc /dev/tc64
```

If this line does not exist, add it with your favorite editor. This entry assumes that the 61KP04 is in slot 6 and that the Tape Controller is set for SCSI address 2. If you have installed your system differently, edit the entry to reflect those differences.

After editing *assign.classes*, you should deassign the device.

```
deassign tc
```

This sets the ownership and protection of the special files to the values that **assign** needs later.

Device Special Files for Hard Disks

Names of special files for hard disks have the general form of */dev/dssu*. The *s* in the device name is the backplane slot number containing the 61KP04 SCSI interface. The *u* is a combination of the SCSI address of the disk controller and the unit number of the disk drive. The value of *u* is the SCSI address of the controller times two plus the unit number of the drive. Thus if the 61KP04 is in slot 6 of the 6130, the Hard Disk Controller is set to SCSI address 3, and the disk drive is internally strapped for unit 0, the recommended special device file name is */dev/ds66*. This is the default configuration as supplied by Tektronix.

The UTek operating system manages the disk surface by means of logical divisions called *partitions*.

Disk partition *p* refers to the entire disk surface. Only the format program uses this partition.

Disk partition *l* refers to the user-writeable portion of the disk. It is the same as partition *p* except that it excludes those portions of the disk reserved for maintenance and diagnostic purposes. The maintenance blocks contain information regarding physical disk parameters and defect data.

Disk partition *b* refers to that portion of the disk reserved for swapping space. Tektronix does not recommend using the hard disk for swapping. In the 6130, adequate swapping space is available on the main system disk and there is a performance degradation in swapping to the external disk.

Disk partition *a* refers to that portion of the disk available for a file system if a swapping partition is reserved.

In this application explanation we use disk partition *l* for our file system.

As with the cartridge tape, there is a driver for buffered data (*/dev/ds66l*) and a driver for unbuffered data (*/dev/rds66l*). When you invoke *MAKEDEV*, all of the drivers for the special file */dev/ds66* are built automatically. You can ignore those that are unused.

Making Device Specific Files for the Hard Disk To make special device files under UTek, use the shell script */dev/MAKEDEV*. Given a name that follows the UTek conventions, *MAKEDEV* calculates the required minor device code and makes the special files. If you have followed the suggested installation procedures, */dev/ds66 MAKEDEV* builds and links all of the necessary drivers.

If you rebuild the UTek kernel or the *MAKEDEV* file with the *sysconf* program, you must tell *sysconf* to include the cartridge tape driver, tca.

FORMATTING THE HARD DISK

The routine for formatting hard disks in the 4944 is *scsifmt*. You must have superuser privileges to run it.

1. Type **su** to obtain superuser status. Enter the *root* password when prompted.
2. Type **/etc/scsifmt /dev/rds66p** (or whatever your slot and device numbers are) to invoke the formatting program.

NOTE

If you get the SCSI Format Command Menu, your disk is already formatted and you can exit the formatting process and proceed to building a file system. If you get the Select Disk Type Menu, your disk is NOT formatted and you should continue with this procedure.

3. From the Select Disk Type Menu, select the 40Mbyte SCSI disk or 80Mbyte SCSI disk depending upon which size disk you have.
4. The volume ID is an identifying character string, such as **hostname_EXT**.
5. The program then asks **Read defect data from?**

   If you received a flexible diskette with defect information with your hard disk, insert it into your 6130 and type **/dev/rdf**. Otherwise, press <RETURN> to return to the main menu. You will have to enter the defect data from your terminal. YOU **MUST** ENTER THE DEFECT INFORMATION!!
6. From the SCSI Format Command Menu, select 8 —Change Maintenance Tables.

7. From the Set Defect Information Menu, select 7 —Add Defect(s) To List. You must enter the information in the sequence that you are prompted for, such as head, cylinder (or track) and byte offset. If a defect length is not specified, enter 1. A typical drive has perhaps half as many defects as its capacity in Mbytes. It is allowed to have as many defects as its capacity in Mbytes.
8. From the Set Defect Information Menu, select 4 —Show Defect List. Verify that there are no errors in the list and that it is complete. If the list is not accurate, correct it using *Delete Defects From List* and *Add Defects To List.*
9. Select 1 —Return To Main Menu.
10. From the SCSI Format Command Menu, select 9 —Map Out Defective Sectors.
11. From the SCSI Format Command Menu, select 11 —Format The Disk With The Given Information.
12. After the disk formats, from the SCSI Format Command Menu, select 2 —Quit The Formatting Program.

This concludes the formatting procedure for an external hard disk.

BUILDING A FILE SYSTEM

You will need a name for the file system on your external disk. You may want the name to suggest the main use of this device. Some system administrators call their file system */u* and use it for user login files. You might call it */ext* to indicate it's the external disk or */ark* to indicate its main use is to archive older versions of files.

To build your filesystem:

1. Type **su** to invoke superuser privileges. (If you have just finished formatting your disk you are already the superuser.)
2. Type **cd /** to move to the root directory.
3. Type **mkdir ext** (or whatever you have chosen for your file system name).
4. If you wish this to be a public directory (read/write access by users, including yourself as a user), then enter **chmod a+rw ext** to allow access to the file system. Individual users can still set more stringent permissions on individual files or directories.
5. If you want this file system to be mounted automatically at power-up and to be checked automatically with *fsck* at power-up, you must enter it in */etc/fstab*. This is not mandatory, but is strongly recommended if you want the new file system on-line whenever the workstation is on.

   Using your favorite editor add a line to */etc/fstab* of the form:

   ```
   /dev/ds66l:/ext:rw:1:2.
   ```

   (Before you do this you might enter **cp /etc/fstab /etc/fstab.old** in case you make an error.)
6. Now enter **newfs /dev/ds66l**. This actually builds your new file system.
7. To mount your new file system, enter **mount /dev/ds66l /ext** or the equivalent entries for your installation. This mounts the device special file */dev/ds66l* as a file system in the */ext* directory and enters it in the */etc/mtab* table.
8. Exit from superuser mode.

Your new disk and its associated file system are now installed.

3

Enhancements

This section describes enhancements that are available for your workstation. You can order the enhancements described by contacting your local Tektronix office. Table 3–1 lists the part numbers and model numbers that you use to order these enhancements.

Table 3–1
ENHANCEMENT PART NUMBERS

Enhancement Description	Part or Model Number
Floor Stand	016–0804–00
Monochrome Display	61VP01
Color Display	61VP02
Dual RS–232–C Interface	61KR01
Serial Sync/Async Interface	61KR02
High Speed GPIB	61KP03
General Purpose Parallel Interface	61KP02
Hard Copy Interface	61KP01
Small Computer System Interface	61KP04
Multibus Interface	61KY01
Memory Expansion, 512–kbyte	61MP01
Memory Expansion, 1–Mbyte	61MP02
Memory Expansion, 2–Mbyte	61MP03
Network Transceiver	60KN01
Streaming Cartridge Tape Drive	61TC01
Character Printer	4644
Color Copier	4695

Floor Stand

The floor stand lets you place your workstation on the floor, beside or under your work area. Figure 3–1 shows the workstation mounted in a floor stand.

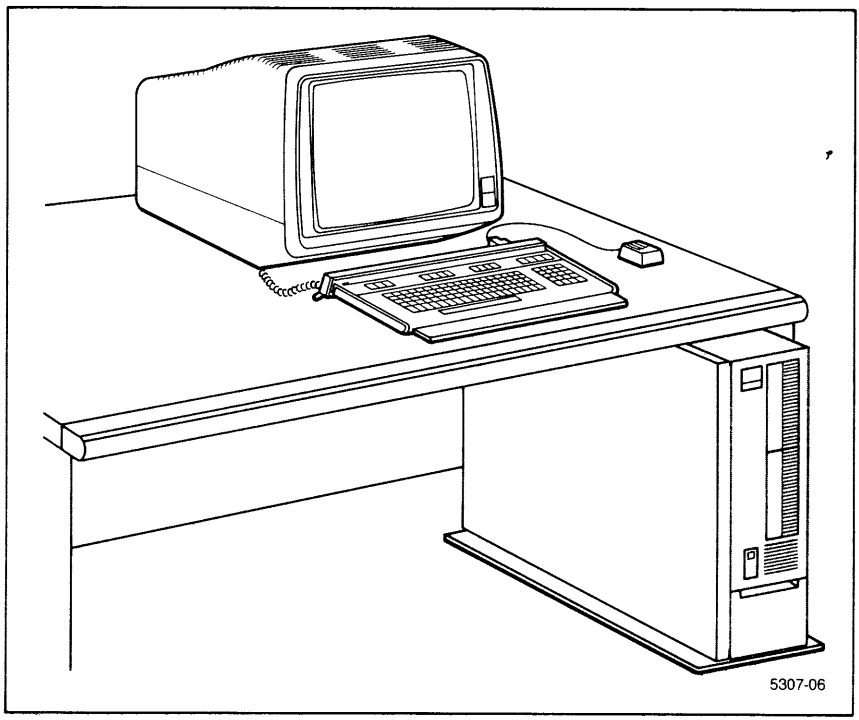

Figure 3–1. Workstation in a Floor Stand.

Displays

The bit–mapped display provides a graphical interface to the workstation. The display is called a *subsystem* because it has its own microprocessor (which resides on a circuit board called the *display processor board*), memory, and operating system to execute instructions that create images on the screen. The display subsystem consists of:

- A 13–inch color or 15–inch monochromatic bit–mapped video display
- A keyboard
- A three–button mouse
- Controlling circuit boards

Interfaces

There are a number of interfaces available that let you connect a variety of peripherals and instruments to your workstation. These interfaces include the Dual RS–232–C interface, Serial Synchronous/asynchronous interface, High Speed General Purpose Interface Bus, Hard Copy interface, Small Computer System Interface, and the Multibus interface.

These interfaces are installed in one (for single–width boards) or two (for double–width boards) of the six backplane slots provided by the workstation. Figure 3–2 shows the backplane slots and how that these circuit boards can be plug into these slots.

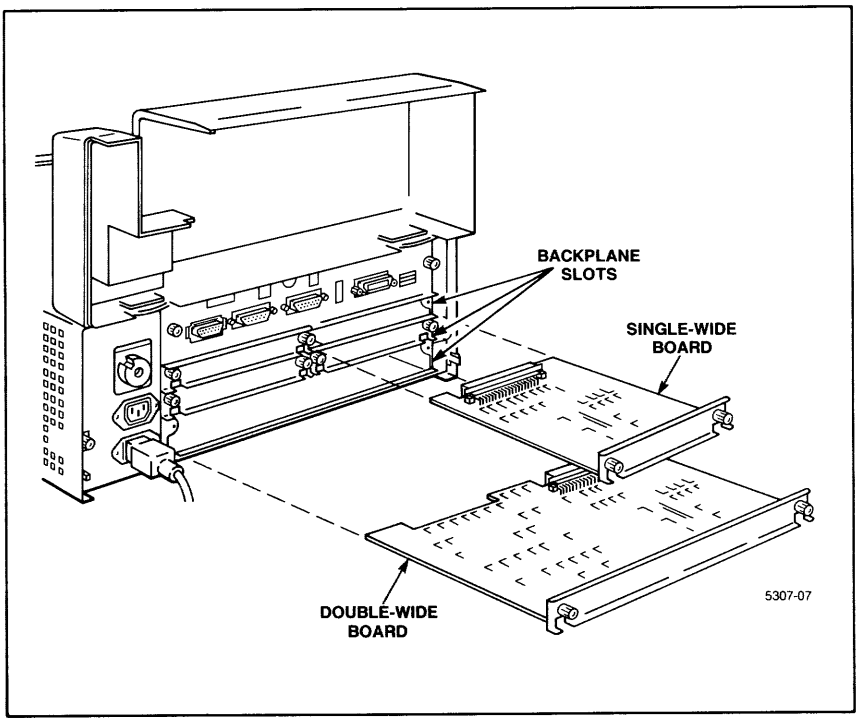

Figure 3–2. Workstation Backplane Slots.

Dual RS-232-C Interface

The Dual RS–232–C Interface is a single–width board that adds two high speed, asynchronous RS–232–C ports to your workstation. These ports transfer data at a rate up to 19.2 kbaud.

The file in the */dev* directory that lets you communicate with peripherals connected to one of the dual RS–232–C ports are in the following format:

/dev/tty*sp*

where *s* is the slot the dual RS–232–C interface is in and *p* is the number of the port (0 is on the left, 1 on the right as you face the back of the workstation).

Serial Sync/Async Interface

This single–width board provides an RS–232–C DCE port and a dual–function RS–232–C/RS–422 DCE port. The RS–232–C interface provides synchronous and asynchronous communications and supports data transfer rates up to 19.2 kbaud. The dual–function interface can be configured to operate either as a standard RS–232–C interface or a synchronous/asynchronous RS–422 interface. In the latter mode, the interface supports data rates up to 153.6 kbaud. The connectors are mounted at the rear of the board and protrude through the rear of the workstation cabinet.

High Speed GPIB Interface

The High Speed GPIB (General Purpose Interface Bus) provides an IEEE 488 interface with a data transfer rate of up to 250 kbytes/sec (listener mode) for block transfers. This rate applies to block transfers to system RAM (typical block size is 4 kbytes). This is a single–width board.

General Purpose Parallel Interface

The single–width GPPI board provides one general–purpose communications port. The GPPI transfers 24 data bits along with control lines that connect various peripherals and custom hardware to your workstation.

Hard Copy Interface

The Hard Copy interface is a single–width board that provides two ports to connect printers, plotters, and copiers to your workstation. These ports are modeled after the Centronics–style 8–bit parallel interface, modified to support color. The Hard Copy ports are plug–compatible with the Tektronix 4695 color graphics copier.

The file in the */dev* directory that lets you communicate with peripherals connected to one of the hard copy ports are in the following format:

/dev/[u][c]hc*sp*

If **u** is in the name, the hard copy port maps lowercase characters to uppercase. If **c** is in the name, the hard copy port appends a carriage return character to every line feed character it passes. *s* is the slot the hard copy interface is in and *p* is the number of the port (0 is on the left, 1 on the right as you face the back of the workstation).

Tektronix printers that plug into the Hard Copy port, such as the 4695, use the file **/dev/hc***sp*.

SCSI

The SCSI (Small Computer System Interface) Mass Storage Interface, is a single-width board that lets you add mass storage peripherals to your workstation. For access to the board, a 50-pin connector mounted on the board protrudes through the back of the workstation.

Multibus Interface

The Multibus Interface lets you use interface circuit boards that conform to the IEEE 796 (Multibus) standard to your workstation.

The Multibus Interface consists of a Multibus Repeater board and a Multibus Interface board. The repeater board plugs into one of the 6 half-width backplane slots of the workstation and sends signals through a 50 pin AMP connector, located on the rear panel of the workstation. The interface board plugs into the Multibus card cage.

For data transfers where the workstation is the initiator, the Multibus Interface is transparent to the workstation. Data transfers initiated by a Multibus device are buffered by a dual-port RAM.

Memory Expansion

You can expand the memory of your workstation with circuit boards of 0.5, 1.0, or 2.0 megabytes parity–checked random access memory. The memory expansion circuit boards plug into the backplane slots provided by the workstation. Figure 3–2 shows how these double–width boards plug in to the back of the workstation.

512-kbyte Memory Expansion Board

This is a double–width board (it requires two adjacent backplane slots) that uses 64–kbit chips. The board is fully populated and provides 512 kbytes of additional memory.

1-Mbyte Memory Expansion Board

This is a double–width board that uses 256–kbit chips and is half–populated, providing 1 megabyte of additional memory.

2-Mbyte Memory Expansion Board

This is a double–width board that uses 256–kbit chips and is fully populated, providing 2 megabytes of additional memory.

Network Transceiver

The IEEE 802.3 transceiver and connecting cable let you connect your workstation to a Local Area Network (LAN) with other workstations and computers to share files, peripherals, and to exchange electronic mail.

Streaming Cartridge Tape Drive

The streaming cartridge tape drive provides a backup media for the UTek file system. You can also use the cartridge tape drive to archive files and to transfer files between workstations.

The cartridge tape drive is a 5.25 inch drive that has a formatted storage capacity of 45 Mbytes using a standard 450′ 0.22–inch tape cartridge and 60 Mbytes using a 600′ tape cartridge. The drive uses a 4 × 6–inch tape cartridge that conforms to ANSI specifications X3.52–1977 and X3b5.82–89 for unrecorded cartridges.

The cartridge tape drive, which mounts in its own cabinet external to the workstation, requires an external Small Computer Systems Interface (SCSI) to connect to the workstation.

You can read and write the streaming cartridge tape with the **cpio** command, as is described for the diskette drive in *Section 2, Peripherals and Interfaces*. The */dev* file you read from and write to controls how the drive operates. The */dev* files that let you access the streaming cartridge tape follows these naming conventions:

/dev/[**n**]**tc***sd*

If **n** is in the name, the tape doesn't rewind at the end of I/O operations. Without the **n**, the tape rewinds at the end of I/O operations. *S* indicates the slot that the drive's interface is connected to. *d* indicates the number of the drive.

Character Printer

The Tektronix 4644 character printer provides printed output from your workstation through one of the RS-232-C ports or through a hard copy interface. The 4644 is an IBM-compatible, wide-carriage, dot matrix printer with a print speed of up to 160 characters/second. The 4644 contains 4 fonts that can be printed in one of 7 print modes.

The printer can print in seven international character sets and can print graphics.

Color Copier

The Tektronix 4695 color copier is an ink-jet printer that provides hard copies of the graphics on your workstation's display. The copier uses three colors of ink plus black ink to produce multicolor text, line drawings, and graphics on coated ink-jet copier paper or on plastic transparencies. The 4695 copier connects to your workstation through one of the Hard Copy ports. The 4695 is accessed through a file in the */dev* directory that follows this naming convention:

/dev/hc*sp*

where *s* is the slot the hard copy interface is in and *p* is the number of the hard copy port the 4695 is connected to (0 is on the left, 1 is on the right).

4

Start-up and Shutdown

This section describes how to start up your workstation and how to shut it down. It is assumed your workstation has been started up before.

If your workstation hasn't been started up before, follow the procedure described in the *First Time Start-up* section in the *6130 System Administration* manual. That section describes the procedures for creating user accounts, configuring your workstation for a local area network, and running file system checks, which must be done the first the workstation is started.

Start–up Procedures

NOTE

This description assumes that the computer board configuration switches on the back panel of your workstation are set so the system comes up in normal mode and boots UTek from the hard disk. These switches should have been set at installation. See the Configuration Switches *topic in this section for more information.*

The procedure for starting up the workstation is summarized below. Each of these steps is further described in the following paragraphs.

1. Check the settings of the computer board configuration switches.
2. Turn on the terminal that is designated as the console. If the display is designated as the console, you can omit this step because the display comes on when you turn on the workstation.
3. Turn on any other peripherals you are using with the workstation, such as terminals, printers, and tape drives.
4. Press the start/stop switch to put it in the Start position.
5. Check for errors or hardware failures in the start–up sequence.
6. Log in to the workstation when you see the

   workstation **login:**

   prompt. *Workstation* is the name assigned to your workstation by your system administrator.

Configuration Switches

The computer board *configuration switches* provide the workstation with information needed to bring the system up. The switches tell the workstation which device is the console, which device to load the operating system from, which file to load the operating system from, and other related information. Figure 4–1 shows the location of the configuration switches on the back panel of the workstation.

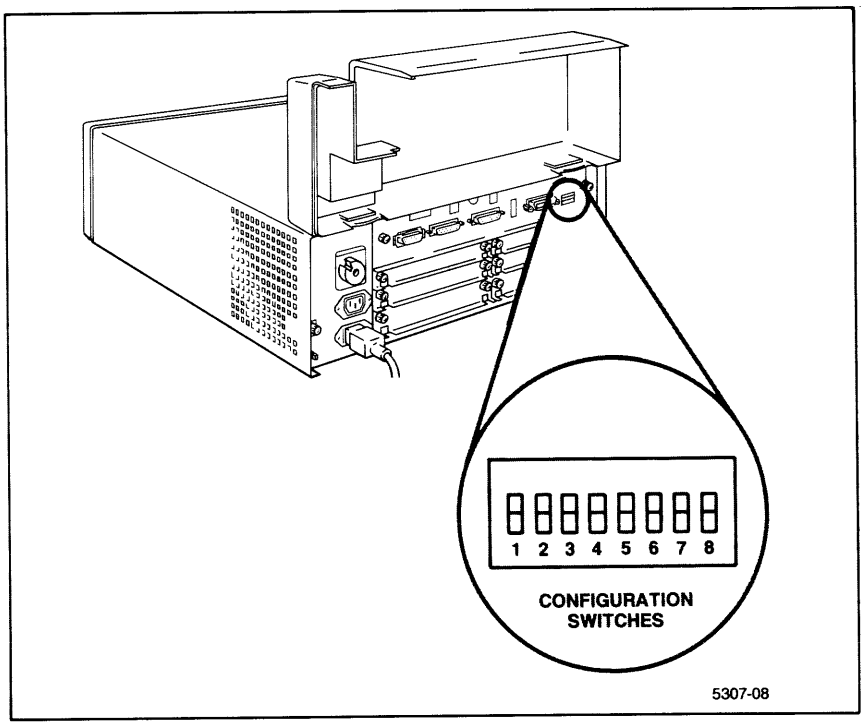

Figure 4–1. Configuration Switches.

Check that the eight configuration switches are set to match the configuration of your system. The switches should have been set properly at the factory, but check them anyway, since the wrong switch settings may cause problems.

NOTE

The configuration switches are only examined when the workstation is running its power-up sequence. Changing the position of switches while the workstation is running has no effect.

The configuration switches are numbered from 1 to 8; switch 1 is on the left, and switch 8 is on the right, as you face the back of the workstation (see Figure 4–2). The meanings of the configuration switches are discussed in the following paragraphs.

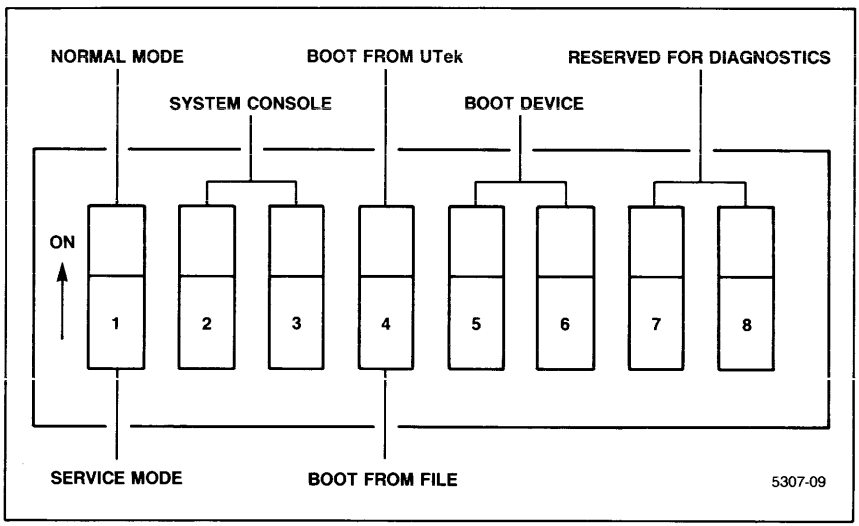

Figure 4–2. Meaning of the Configuration Switches.

Switch 1

Sets the workstation to either *normal mode* (up) or *service mode* (down). Switch 1 should usually be set to normal mode (up).

The setting of switch 1 determines the meaning of the rest of the configuration switches. If switch 1 is set to service mode (down), the rest of the configuration switches select diagnostic tests that locate faults in the workstation hardware (see the *6130 Diagnostics* manual).

If switch 1 is set to normal (up) mode, the rest of the configuration switches have their *normal* meanings, which are described in the following paragraphs.

Switches 2, 3

Select the device you want to be the *console*. There may be one display and a number of terminals connected to your workstation, but only one of these devices can be the console.

The console is the device that displays system messages. Table 4–1 shows how to set configuration switches 2 and 3 to select the device you want to be the console.

Table 4–1
SELECTING THE CONSOLE DEVICE

Console Device	Switch 2	Switch 3
Display (Option 27/28)	up	up
9600 baud RS–232–C terminal (port 1)	up	down
1200 baud RS–232–C modem/terminal (port 0)	down	up
300 baud modem/terminal (port 0)	down	down

Switch 4

Determines whether the workstation boots UTek (up) in multiuser mode, or a file that you specify (down). Switch 4 should usually be set to boot from UTek (up) in multiuser mode.

If you set Switch 4 in the down position, you are prompted for a filename to boot from. If you enter a filename, that file is booted. If you press <RETURN> without entering a filename, UTek is booted in *single-user mode*. Single–user mode should only be used to perform system administration tasks on the workstation.

Switches 5, 6

Select the *boot device*, the device from which the workstation loads the operating system. Possible boot devices are:

- Diskette
- Internal hard disk
- Local Area Network (LAN)

Table 4–2 shows how to set switches 5 and 6 to select the boot device.

Table 4–2
SELECTING THE BOOT DEVICE

Boot Device	Switch 5	Switch 6
Autoboot	up	up
Hard disk	up	down
Diskette drive	down	up
LAN port	down	down

If you select *autoboot*, the workstation searches for a device from which to boot UTek. The workstation tries to boot from (in order): the diskette drive, the hard disk, and the Local Area Network (LAN).

Switches 7, 8 Reserved for use with the Diagnostics operating system. See the *6130 Diagnostics* manual. These two switches should be up during normal system operation.

Turning on the Console and Peripherals

If you don't know which device is the console, refer back to the discussion of configuration switches 2 and 3 under the topic Configuration Switches of this section.

If you have the display (Option 27/28), you don't have to turn its power on. The display receives power from its connection to the workstation.

If you have designated a terminal connected to one of the RS–232–C ports as the console, turn it on.

Turn on printers, tape drives, and any other peripherals you are going to use with the workstation.

The Start/Stop Switch

NOTE

This discussion assumes your workstation is laying flat, as it would on a table top. If your workstation is mounted upright in a floor stand, the start/stop switch is in the lower left corner of the front panel of the workstation.

The start/stop switch is located on the lower right corner of the front panel of the workstation (see Figure 4–3).

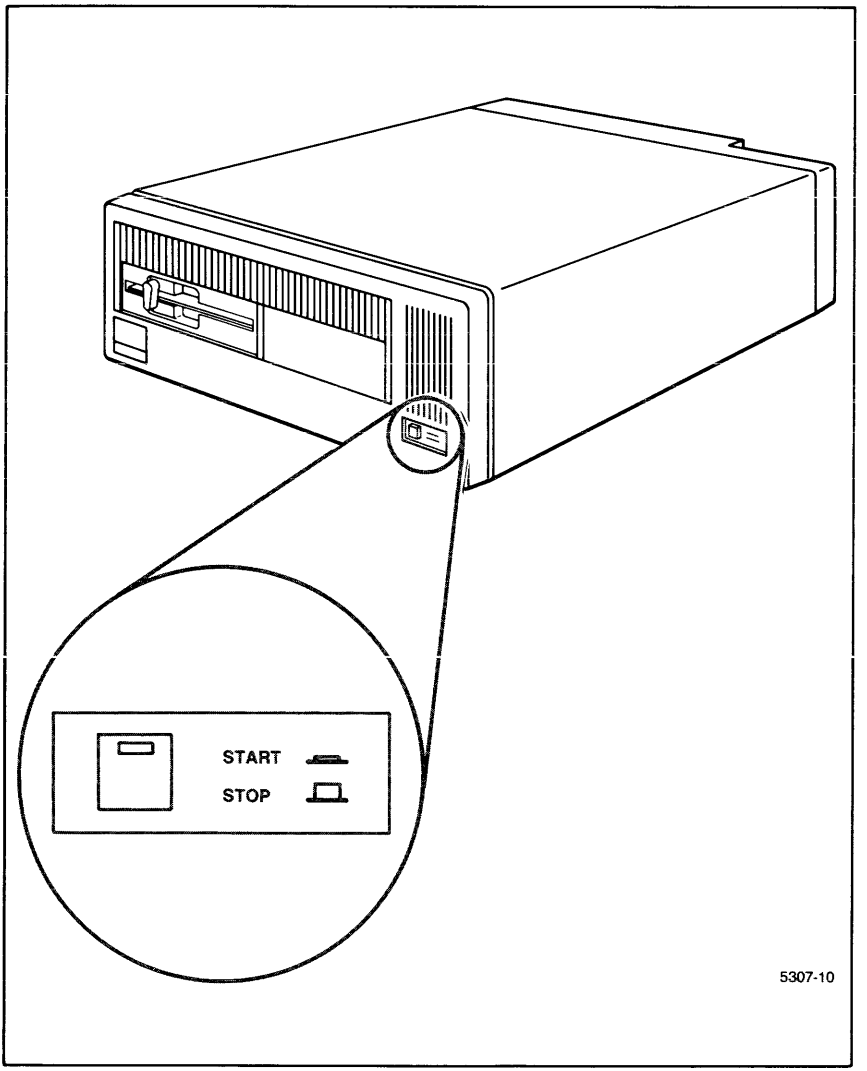

Figure 4–3. The Start/Stop Switch.

The switch has two positions: Start (in) and Stop (out). A green light on the switch turns on when the switch is pressed to Start. To start the workstation, press the start/stop switch in.

When you press the start/stop switch, the workstation begins running a series of diagnostic tests that check the workstation hardware. As each test completes successfully, a confirmation message is displayed on the console. If the start–up diagnostics don't find any errors, the UTek operating system is *booted.*

Checking for Start–up Errors

If you press in the start/stop switch and nothing happens, or if messages on the console indicate errors or failures during start–up, you should check the following:

- Is the workstation plugged in?
- Is the console connected properly to the workstation? (Check for loose connections)
- Is the baud rate of the console set properly? If not, messages on the screen will be unrecognizable.
- Are the configuration switches set for the proper console device, boot file, and boot device? (Refer back to the topic *Configuration Switches* in this section.)
- Are the diagnostic LEDs on? (Refer to the following paragraphs on Diagnostic LEDs.)

Diagnostics LEDs

If a critical error is found during start–up, the workstation stops processing. You can detect a critical error by checking the diagnostic LEDs, which indicate the results of the start–up diagnostic tests. There are two of these LEDs, located behind the cable management cover at the back of the workstation (see Figure 4–4).

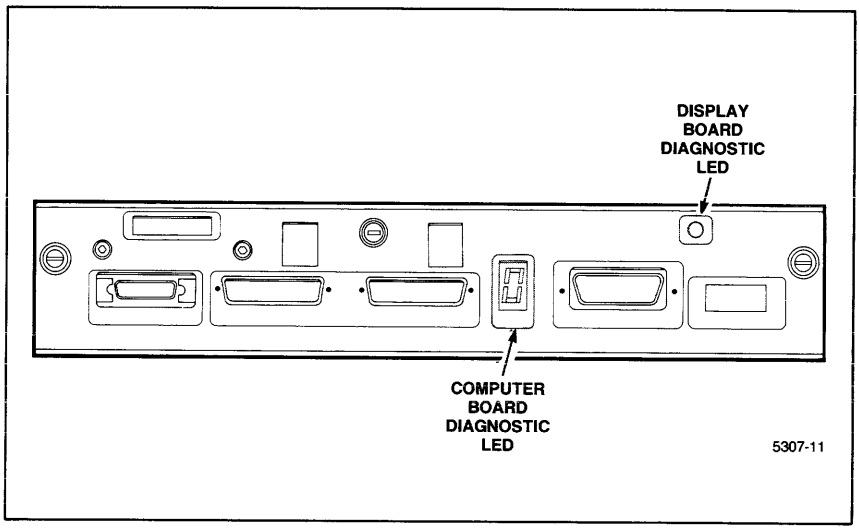

Figure 4–4. Diagnostic LEDs.

The rectangular seven–segment LED, labeled *computer board diagnostic*, is attached to the computer board and can display a hexadecimal digit (0–9, A–F). The steps the workstation goes through immediately after you press the Start/stop switch and what is displayed on the seven–segment LED are discussed below. This discussion assumes the configuration switches are set so that the UTek operating system boots from the internally–mounted hard disk (see Table 4–2).

Press Switch	When you press the start/stop switch the system resets and the seven–segment LED remains off.
ROM Diagnostics	Diagnostic tests are loaded from the workstation's read–only memory (ROM) and executed. As each test is executed its test number is displayed on the seven–segment LED. If a test fails, the test's number remains on the LED. If this type of failure occurs, see the *6130 Diagnostics* manual or contact your local Tektronix office. If all tests complete successfully, the seven–segment LED turns off.
Diagnostic OS	Then the diagnostic operating system is loaded from the hard disk and executed. The seven segments of the LED flash in a race track pattern. When the diagnostic operating system is finished running tests, the LED turns off.

UTek — The UTek operating system is loaded from the hard disk and executed. The seven segments of the LED cycle in a race track pattern. The segments flash quickly when the system is not running many processes and flash slowly when the system is busy running your programs.

The second diagnostic LED is a single round light that is attached to the display board. If the round LED is off at the completion of the diagnostic tests, the tests terminated normally. If the round LED remains on or blinks off and on, the test terminated abnormally, indicating a hardware failure in the display subsystem.

If either of these LEDs indicate a hardware failure, refer to the *6130 Diagnostics* manual.

Logging In

You know the system is up when you see this message on the screen:

workstation **login:**

where *workstation* is the name assigned to your workstation by your system administrator.

Type your login name and press <RETURN>. If your account has a password assigned to it, you are then asked for your password. Type your password and press <RETURN>. If you don't know your login name or password, ask your system administrator.

A welcome message, called the *message of the day*, is usually displayed. Then, if you are a Bourne shell user, the commands in your *.profile* file are executed. If you are a C-Shell user, the commands in your *.cshrc* and *.login* files are executed.

You may see a line that looks like the following line:

TERM = (*termtype*)

If you see this line, the system is waiting for you to enter the abbreviation of the type of terminal you are using. If you press <RETURN>, whatever is in the parenthesis is used as your terminal type. If you type some characters followed by <RETURN>, whatever you type is used as your terminal type. See the discussion of the **tset** command in the following paragraphs for more information.

After the commands in your *.profile* or *.login* file are executed, your prompt (usually $ or %) is displayed, indicating you can begin entering commands.

The tset Command

A command that is commonly put in your *.profile* file (if you are a Bourne shell user) or in your *.login* file (if you are a C-Shell user) is the **tset** command, which tells the

system what type of terminal you are using. If your *.profile* or *.login* file contains this command, a line in the following format may appear on your screen when you log in:

```
TERM = (termtype)
```

The prompt remains immediately after the parenthesis, waiting for you to enter something. If you press <RETURN> whatever is in the parenthesis, if anything, is used as your terminal type. If you enter a some characters, followed by <RETURN>, whatever you type is used as your terminal type. If the system doesn't recognize what you enter, some of the UTek programs (such as the **vi** text editor) won't work properly.

The correct abbreviations for all the terminals supported by UTek are listed in the */etc/termcap* file. To read these abbreviations, enter the following command line:

```
grep '|' /etc/termcap
```

The following lines are examples of the output of the preceding command line:

```
XQ|4107|tek4107|unicornII|Tektronix 4107:\
NH|aaa|aaa-30|ambas|ambassador|ann arbor ambassador/30 lines:\
```

Each line in the preceding example contains a list of abbreviations that you can use for a terminal. The first line gives all the abbreviations that can be used for the Tektronix 4107 terminal, separated by the vertical bar (|) character. The second line gives the abbreviations that can be used for the Ann Arbor Ambassador terminal.

The following line is a sample **tset** command from a *.profile* or *.login* file.

```
eval `tset -Q -m :?tek4107`
```

The **tset** command by itself doesn't tell the system what type of terminal you are using, it generates commands that, when executed by the **eval** command, tell the system what type of terminal you are using.

The parts of the **tset** command in the previous example have these meanings:

—Q	Keeps the **tset** command from printing messages, which describe what it is doing.
—m :?tek4107	Tells **tset** what to put in parenthesis (tek4107 in this example). In the example, if the question mark (?) is omitted, the message **TERM** = (*termtype*) isn't printed and *tek4107* is used as the terminal type.

For more information, see *tset(1)* and *termcap(5)* in the *UTek Command Reference* manual.

Shutdown Procedures

At the end of the day you should logout, but you don't need to shut down the workstation. It can keep running until there is a reason to shut it down (for example, to service the workstation).

1. Type:

   who

   to make sure there are no other users logged in on the workstation. If there are other users logged in, follow the procedure in the *6130 System Administration* manual for shutting down the workstation.

2. Log out by typing <CTRL–D>.

3. When you see the

   workstation **login:**

   prompt, press the start/stop switch to Stop (refer back to Figure 4–3). (The workstation runs some file system checks before shutting down. It take a few minutes before the workstation completely shuts down.)

4. Switch off all peripherals connected to the workstation.

If the System Halts

If you must reboot or rebuild the operating system due to a partial or full system halt, see the *6130 System Administration* manual.

5

Customizing Your Account

A number of UTek programs begin by looking for a file in your home directory with a certain name. When the program finds the file, it reads each line in the file and alters its operation based on the contents of the file.

This section provides examples of how you can use these files to tailor your account on UTek to meet your needs. The files that are discussed in this section are:

.profile	A file of commands that are executed when you log in if you are a Bourne Shell **(sh)** user.
.cshrc	A file of commands that are executed whenever you invoke the C–Shell **(csh)** to execute commands.
.login	A file of commands that are executed when you log in if you are a C–Shell **(csh)** user.
.logout	A file of commands that are executed when you log out if you are a C–Shell **(csh)** user.
.mh_profile	Sets options that control how the programs of the MH Mail System work.
.aliases	Defines aliases, or abbreviations, you can use on the To: line of your mail messages.
.exrc	Sets options that control how the **vi** text editor works.
.plan	Contains information that is printed by the **finger** command.
.project	Contains information that is printed by the **finger** command.

The .profile File

After you log in, the program that prints a prompt on your screen (usually $ or %) and reads what you type at the keyboard is called a *shell*. There are two shells standard on your workstation: the Bourne Shell (also called *the Shell* or *sh*) and the C–Shell (also called *csh*). Typically, new users on UNIX–based systems use the Bourne Shell, but check with your system administrator to be sure.

When you log in, the Shell searches for a file named *.profile* in your home directory. When the Shell finds this file, it reads and executes each line in the file as if you had typed it at the keyboard. Then the Shell issues you its first prompt, which is usually **$**, indicating it is ready to accept typed commands.

You can put any commands you want in your *.profile* file. Users commonly put commands in their *.profile* to set up their terminal, set the prompt, check for the

arrival of new mail, set variables, and program the function keys of their keyboard.

Example 5–1 shows a sample *.profile* file.

NOTE

The line numbers in Example 5-1 are not in the .profile file; they are included for reference only.

```
 1 #! /bin/sh
 2 EDIT="vi"
 3 MORE="-u -f"
 4 PATH=:${HOME}/.bin:${HOME}:$PATH
 5 PS1=" `hostname`> "
 6 PS2="? "
 7 SEDIT="prompter"
 8 export EDIT MORE PATH
 9 export PS1 PS2 SEDIT SHELL
10 iftest -s /usr/spool/mail/$USER
11 then inc
12 fi
13 eval `tset -s -m :?display -m network:? vt100`
14 stty crt susp ^Z dsusp ^Y rprnt ^R flush ^O werase ^W \
15      lnext ^V intr ^? stop ^S start ^Q erase ^H
16 who
```

Example 5–1. Sample .profile File.

The #! Line (line 1)

The #! line tells UTek which shell to use to execute commands in this file. This line is not always required, but it is good programming practice to include it in all files of commands.

The EDIT Variable (line 2)

Environment variables are string–valued variables that change the way one or more UTek commands function. Commands that set environment variables are commonly put in the *.profile* file.

The EDIT environment variable is used by the UTek **comp** command. When you are finished composing a mail message with **comp**, you are given the prompt:

What now?

If you respond with:

edit

comp reads the name of a text editor from the EDIT variable and invokes that editor for further editing of the mail message you are composing.

The MORE Variable (line 3)

The MORE environment variable sets options of the **more** command. **More** lets you read the contents of a file, or the output of a command, before it scrolls off your terminal screen.

When the **more** command is invoked, it reads what you put in the MORE variable, if anything, and alters its operation accordingly. Line 3 of Example 5–1 causes the **—u** and **—f** options to be set every time **more** is invoked. These options have the following effect on **more**:

—u Causes **more** to suppress underlining and standout mode on your terminal.

—f Causes **more** to count logical, rather than screen lines when determining how many lines to print on your screen.

See *more(1)* in the *UTek Command Reference* manual for more information.

The PATH Variable (line 4)

When you enter a command, the Shell must know the full pathname of the command in order to execute it. If you don't enter the full pathname of the command, the Shell looks for it in the directories listed in the PATH variable.

When you log in, PATH is set to you current directory (which is represented by a period) and */bin* and */usr/bin*, the directories that contain the UTek commands. You probably want to add other directories to PATH.

Line 4 of Example 5–1 adds three directories to PATH. The colons (:) separate the directory names. The first directory added to PATH is *${HOME}/.bin*. The Shell expands ${HOME} to your home directory, so *.bin* is in your home directory. The braces around HOME are necessary to distinguish it from the characters next to it. The *.bin* directory is commonly used to hold commands that you write.

The second directory added to PATH is *${HOME}*, your home directory. Putting $PATH after these three directories adds the default value of PATH (your current directory, */bin*, and */usr/bin*).

The PS1 Variable (line 5)

The PS1 variable contains the string that the Shell uses for its prompt. The default PS1 prompt is the dollar sign ($).

Line 5 of Example 5–1 shows how to put the output of a command into a Shell variable. A command enclosed in grave accents () is executed before the assignment to PS1 is made. In this example the prompt is set to the name of the host computer you are currently logged onto, followed by a *greater than* (>) sign. This is useful for keeping track of the computer you are logged onto, when you switch back and forth between several computers.

The PS2 Variable (line 6)

PS2 is the prompt the Shell uses when it reads a command line and still needs input. The default PS2 prompt is the *greater than* (>) sign. Line 6 of Example 5–1 sets PS2 to a question mark followed by a space.

The SEDIT Variable (line 7)

The SEDIT variable contains the name of the text editor that is invoked when you call **comp** to send electronic mail. Line 7 of Example 5–1 sets SEDIT to **prompter**, an editor that allows rapid composition of mail messages.

You may wonder why **comp** uses both EDIT and SEDIT. Using different variables lets you use different text editors when you are creating a new mail message and when you are editing an existing mail message.

The export Command (lines 8-9)

The **export** command puts Shell variables into your environment for other UTek commands to use. **Export** is built into the Shell, which means it isn't stored in its own file in the UTek file system.

Test for Incoming Mail

When you receive mail, it is stored in the file */usr/spool/mail/$USER*, (the system mail file), and the Shell prints the message *You have new mail.* Then you can use the **inc** and **show** commands to read your mail.

The statement on line 10 tests to see if the size of your system mail file is greater than zero. If the size is greater than zero, then you have new mail and the **inc** command (line 11) is executed to move the mail messages into your mail folders. The **fi** command (line 13) is the end of the **if** statement.

How to Set up your Terminal (line 13)

Line 13 of Example 5–1 is a combination of two commands, **eval** and **tset**, which tell UTek how to communicate with your terminal. **Tset** generates commands to set up your terminal and **eval** executes those commands.

Tset creates commands to set the TERM (terminal type) and TERMCAP (terminal capability) environment variables for the shell you are using. See *tset(1)* for a complete description of this command.

The stty Command (line 14-15)

The **stty** command, which is shown in line 14 and continued on line 15 of Example 5–1, sets options and the meaning of control characters on your terminal. For a complete description, see *stty(1)* in the *UTek Commands Reference* manual.

To find out the current settings of all **stty** options on your terminal, type:

```
stty everything
```

To find the settings of the most commonly used **stty** options, type:

```
stty all
```

To find out the settings of all **stty** options that are different from their default values, type:

```
stty
```

The who Command (line 16)

When you log into UTek, you may want to know who else is on the system. The **who** command prints the names of all users that are currently logged in.

Other .profile Possibilities

As was stated earlier in this section, you can put any commands you want in your *.profile* file. Example 5–1 showed some commands that are commonly put in a *.profile* file. The following paragraphs describe some other features you may want to put in your *.profile* file.

Functions

Functions are a new feature of the Shell that let you enter a short name to execute a long string of commands and options. You can define functions in your *.profile* file.

The syntax of functions is shown in *UTek Command Reference — sh(1).*

Other Variables

Below are some other variables you can set in your *.profile* file. Example 5–1 shows how you can assign values to Shell variables.

CDPATH	is the colon–separated list of directories that is searched when you enter the **cd** command. This is similar to the Shell's using the PATH variable to find commands.
MAILCHECK	is the number of seconds between checks for new mail. The default is 600 seconds.
MAILPATH	is a colon–separated list of files to check for new mail.

C–Shell Files

The C–Shell has some advanced features that aren't in the Shell. You can read about some of these features in *Chapter 13, The UNIX System at Berkeley*, in *Introducing the UNIX System.*

If you decide to use the C–Shell, you should change your entry in the system password file so that UTek invokes */bin/csh* when you log in. To do this you need to use the **chsh** (change shell) command. To make the C–Shell your login shell, type:

chsh *yourname* **/bin/csh**

where *yourname* is your login name.

Every time you invoke the C–Shell it executes commands from a file named *.cshrc* in your home directory. In addition, when you log in, the C–Shell executes commands from a file named *.login* in your home directory. When you log out, the C–Shell executes commands from a file named *.logout* in your home directory.

Sample .cshrc File

The *.cshrc* file should contain C–Shell specific commands that you want to execute every time you create a new C–Shell.

Example 5–2 shows a sample *.cshrc* file.

NOTE

The line numbers in Example 5-2 are not in the .cshrc file; they are included for reference only.

```
 1 set path=( . ~/.bin /bin /usr/bin )
 2 if ($?prompt) then
 3         set history=29
 4         set mail=(60 /usr/spool/mail/$USER)
 5         setenv EDIT vi
 6         setenv MORE page -u -f
 7         setenv SEDIT vi
 8 endif
 9 alias more `more -u -f \!*`
10 alias hi history
11 alias who `who \!* ¦ sort ¦ more`
```

Example 5–2. Sample .cshrc File.

The path Variable (line 1)

The C–Shell uses the **path** variable instead of the PATH environment variable, which is used by the Bourne shell. Line 1 of Example 5–2 shows how to use the **set** command to assign a value to the **path** variable.

The first directory in **path** is the current working directory, which is represented by the period (.). The second directory is named *.bin* and is a subdirectory of your home directory. In C–Shell, a tilde (~) represents your home directory. The last two directories in **path**, */bin* and */usr/bin*, contain the UTek commands. Each directory is separated by a space.

When you create a C–Shell (by logging in, executing a C–Shell file, or starting up a subshell), **path** is given a default value of:

. /bin /usr/bin

The if Statement (lines 2, 8)

Line 2 of Example 5–2 is the C–Shell **if** statement. If the expression in the parentheses is true, all the commands down to the **endif** (line 8) are executed.

The expression **$?prompt** is true if the **prompt** variable (the string that the C–Shell uses for its prompt) has been set. This is one way of testing for an interactive C–Shell.

The history Variable (line 3)

The C–Shell can store commands you type in a *history list*, so you can reenter them later. To make the C–Shell create a history list, you must set the history variable to tell the C–Shell how many previously–executed commands to keep track of.

Line 3 of Example 5–2 tells the C–Shell to remember the last 29 commands entered. Remembering too many commands causes the C–Shell to run out of memory.

To reenter a command, you type an exclamation mark (!) followed by another exclamation mark (for the last command entered), an integer, or a pattern–matching string. This feature can save you a lot of typing and is one of the major advantages of the C–Shell over the Shell. (For a complete discussion of the history list, see *Chapter 13* in *Introducing the UNIX System*.)

The mail Variable (line 4)

C–Shell uses the **mail** variable to store the name of the file that receives your mail. Line 4 tells the C–Shell to check the directory */usr/spool/mail/$USER* (which is your *system mailbox*) every 60 seconds for new mail.

Setting Environment Variables (lines 5-7)

When you use the C–Shell, the command you use to assign a value to an environment variable is **setenv**. Lines 5–7 set the EDIT, MORE, and SEDIT environment variables. These variables have the same function under the C–Shell as under the Bourne Shell.

Creating Aliases (lines 9-11)

The **alias** command lets you create a short character string to represent a long character string. This feature of the C–Shell can save you a lot of typing.

Line 9 of Example 5–2 causes:

```
more -u -f
```

to be entered when you use the **more** command.

The exclamation mark followed by the asterisk (!*) is replaced by any other command line arguments you enter to **more**. (The exclamation mark must be preceded by a backslash (\) to override its special meaning to the C–Shell.)

Line 10 lets you enter the **history** command by typing **hi**. Line 11 passes the output of **who** through **sort** and **more**.

Sample .login File

The *.login* file should contain commands that you want to execute when you first log into UTek. Example 5–3 shows a sample *.login* file.

NOTE

The line numbers in Example 5-3 are not in the .login file; they are included for reference only.

```
1  set noglob
2  eval `tset -s -Q -m :?display -m dialup:?vt100 -m network:?display`
3  stty crt susp ^Z dsusp ^Y rprnt ^R flush ^O werase ^W \
4          lnext ^V intr ^? stop ^S start ^Q
5  switch ($TERM)
6  case aaa:
7          set history=35
8          ${HOME}/.bin/setup.aaa
9          breaksw
10 case vt100:
11          set history=30
12          ${HOME}/.bin/setup.aaa
13          breaksw
14 default:
15          set history=23
16 endsw
17 set prompt="`hostname`\!"
```

Example 5–3. Sample .login File.

The noglob Variable (line 1)

Line 1 of Example 5–3 sets the Boolean C–Shell variable **noglob**. If **noglob** is set, the C–Shell doesn't try to expand characters with special meanings (like *) into filenames when you enter them as arguments to a command. This is useful in C–Shell files that are not dealing with filename expansion, such as this *.login* file.

Setting Up your Terminal (lines 2-4)

Using the **eval** and **tset** commands to set up your terminal is discussed under the *Bourne Shell* heading of this section. These commands are the same for the C–Shell as for the Bourne Shell.

Lines 3 and 4 call **stty** to set options on your terminal. The first argument, **crt**, tells UTek to set options for a CRT. The rest of the arguments define the functions of certain control characters. See *stty(1)*, *tty(4)* and *Introducing the UNIX System* for a complete discussion of **stty**.

The switch Statement (lines 5-16)

The C–Shell **switch** statement is analogous to the Bourne Shell **case** statement. Line 5 of Example 5–3 shows the **switch** statement. The string in parentheses, in this case the value of the $TERM environment variable, is successively matched against the strings in the **case** statements (lines 6 and 10). If a match is found, the

commands between the **case** statement and the **breaksw** statement (lines 9 and 13) are executed. If no match is found, the commands between the default label and its **breaksw** command are executed.

In this example, the length of the history list is altered, depending on the type of terminal used (lines 7, 11, 15). This ensures the entire history list fits on your terminal screen.

Line 8 of Example 5–3 calls a program that sets up an Ann Arbor Ambassador (abbreviated *aaa*) terminal, if you are on that type of terminal. Line 12 sets up a Digital Equiptment Corporation VT100 (abbreviated *vt100*), if you are on that type of terminal. Note these calls are to hypothetical programs in your *.bin* directory; you would need to create these programs.

The prompt Variable (line 17)

The prompt variable contains the string that the C–Shell uses for its prompt. This is analogous to the PS1 variable under the Bourne Shell. The default value of **prompt** is a percent sign (%).

Line 17 sets the prompt to the output of the **hostname** command, followed by the index of this command in the history list, which is substituted for the exclamation mark (!). See *history(1)* for a complete discussion of this index. The exclamation mark must be preceded by a backslash (\) to override its special meaning to the C–Shell.

Other .cshrc and .login Lines

The UTek C–Shell contains all the features of the Berkeley 4.2 BSD C–Shell, as well as the extensions described in the following paragraphs. Some of these features are enabled by setting variables, which can be done in your *.login* or *.cshrc* files.

File Name Completion

When you type a command, you can use abbreviations for filenames. First, set the **complete** variable by typing:

```
set complete
```

Then, when you are entering a filename as an argument to a command, type as many characters as you need to make the filename unique. Then press the <ESC> key. Below is an example of filename completion.

```
% ls
DSC.OLD   bin        cmd      lib    memos
DSC.NEW   chaosnet   cmtest   mail   netnews
bench     class      dev      mbox   news
% ls ch<ESC>
% ls chaosnet          (The cursor remains at the end of this line)
```

Press <RETURN> to enter the command.

File and Directory List

When you are entering a command, you may want to know what filenames match what you have typed so far. First, set the **list** variable by typing:

```
set list
```

Then, when you are entering a filename or directory name as an argument to a command, press <CTRL–D> to list all matching files and directories. Below is an example of the file and directory list.

```
% ls
DSC.OLD   bin        cmd      lib    memos
DSC.NEW   chaosnet   cmtest   mail   netnews
bench     class      dev      mbox   news
% ls c<CTRL-D>
chaosnet    class     cmd     cmtest
% ls c          (The cursor remains at the end of this line)
```

You can then type any more characters you need to make the name unique (if filename completion is set) and press <RETURN>.

Command Name Recognition

You can use the completion and list features when entering command names, as well as filenames and directory names. Below are examples of command completion:

```
% pass<ESC>
% passwd          (The cursor remains at the end of this line)
```

and command listing:

```
% pas<CTRL-D>
passwd    paste
% pas          (The cursor remains at the end of this line)
```

Automatic Logout

With this feature, the C–Shell logs you out if your terminal is idle for a specified period of time. You can set the **autologout** variable; for example:

```
set autologout=60
```

waits 60 minutes before logging you out. You can turn this feature off by typing:

```
unset autologout
```

When you log in, this feature is always unset.

Saving Your History List

The C–Shell can store your history list between login sessions. The list is stored in a file named *.history* when you log out and is restored the next time you log in. To set this feature, specify the number of commands you want the C–Shell to restore. For example:

```
set savehist=30
```

causes the C–Shell to store the last 30 commands you entered.

A .logout File

The *.logout* file should contain commands that you want executed when you log out of UTek. The *.logout* file typically is very short. Some common things to do from the *.logout* file are to remove temporary files and clear your terminal screen. For more information, see *clear(1)*.

MH Mail Files

UTek uses an electronic mail system called *MH*, which replaces the electronic mail system used on most other UNIX systems. MH, which was developed by Rand Corporation, is a group of programs that manipulate mail messages and mail folders. A *mail message* is a file that is sent to another user and a *mail folder* is a directory of mail messages. Table 5–1 lists the programs in MH.

Table 5–1
PROGRAMS IN THE MH MAIL SYSTEM

Command	Function
ali	List all mail aliases.
comp	Compose a mail message.
refile	Move mail messages to another mail folder.
folder	Print information about, or set, the current mail folder.
folders	List all mail folders.
forw	Forward mail messages to another user.
inc	Incorporate new mail messages into a mail folder.
mail	Send and receive mail messages.
mhl	Print a formatted listing of mail messages
next	Print the next mail message on the terminal.
pick	Select mail messages by content and do something with them.
prev	Print the previous mail message on the terminal.
prompter	A text editor for quick composition of mail messages.
rmail	Send mail from a remote site
rmf	Remove a mail folder and all mail messages in it.
rmm	Remove mail messages from a folder.
scan	Print a one line description of each mail message in a folder.
send	Send a mail message.
show	Print mail messages on the terminal.

For a complete description of each of the programs in MH, see:

- *UTek Commands Reference*
  Section 1
- *UTek Tools*
  Part 2A — Electronic Mail

Sample .mh_profile File

The file in your home directory named *.mh_profile* contains information to tailor the MH programs to your needs. Each line of the *.mh_profile* file consists of a keyword followed by a colon (:) and a character string. When an MH program is executed, it looks in the *.mh_profile* file for a keyword that is the same as the program'ss name and special keywords that apply to it. If the program finds a *.mh_profile* entry that applies to it, it responds according to the character string following the colon.

Example 5–4 shows a sample *.mh_profile* file. Note that there can't be any blank lines in the .mh_profile file and you can write keywords in any combination of uppercase and lowercase.

NOTE

The line numbers in Example 5-4 are not in the .mh_profile file; they are included for reference only.

```
1 Path: Mail
2 Current-Folder: inbox
3 Msg-Protect: 640
4 Folder-Protect: 710
5 dist: -annotate
6 forw: -annotate
7 repl: -annotate -nocc
8 send: -noformat -noambiguous -verbose
```

Example 5–4. Sample .mh_profile File.

The Path Entry (Line 1)

The Path entry contains the name of a directory in which your MH mail folders and other mail files live. If the directory does not begin with a slash (/), it is assumed to be in your HOME directory.

The Path entry is a special keyword entry that is used by all MH mail programs.

The Current-Folder Entry (Line 2)

The Current–Folder entry contains the name of the directory that is the current mail folder. This entry is maintained automatically by many of the MH programs.

The Msg-Protect Entry (Line 3)

The Msg–Protect entry contains the setting of the UTek file protection bits for mail messages that are incorporated into your mail folder with **inc**. See *chmod(1)* for a description of the UTek file protection bits.

The default value of Msg–Protect is 600 (read and write permission for the owner only).

The Folder-Protect Entry (Line 4)

The Folder–Protect entry contains the setting of the UTek file protection bits for the current mail folder.

The default value of Folder–Protect is 700 (read, write, and directory access permission for the owner only).

The Command Entries (Lines 5-8)

Lines 5–8 of Example 5–4 are *.mh_profile* entries that are only used by the command that is specified in the keyword. The *—string* entries, which follow the colon, set options of the command each time the command is invoked. The same thing could be done by entering these options on the command line. To find out what each of these options does, see Section 1 of the *UTek Commands Reference* for each command listed.

Additional lines in this format could be added for the rest of the MH programs listed in Table 5–1.

Other MH Entries

There are other MH keywords that can be set in the *.mh_profile* file. Some of these can be used in place of environment variables. For example, **prompter** and **prompter–next** can be used instead of EDIT and SEDIT. See *mh_profile(5)* for a complete list of keywords.

Sample .aliases File

A *mail alias* is a name that stands for a user's or group of users' login name when you are sending a mail message. The *.aliases* file is a list of mail aliases that you create.

When you send a mail message, **send** searches a file in your home directory named *.aliases* to see if you are using an alias on the *To:* line of your message. If **send** determines that you are using an alias, it replaces the name on the *To:* line of your message with the value from the *.aliases* file.

A line in your *.aliases* file has the format:

name: *string*

Name is the name that you want to use on the *To:* line of your mail message and *string* is the value you want substituted for that name.

Example 5–5 shows how you would create and use an alias, *group*, that contains the users *janed* and *johnd*.

```
$ cd
$ cat .aliases
group: janed, johnd
$ comp
To: group
Cc:
Subject: A test of .aliases
--------
Hi,

Just testing the MH mail alias feature.
<CTRL-d>
-------

What now? send -verbose
janed: queued to local mailer.
johnd: queued to local mailer.
```

Example 5–5. Sending Mail Using an Alias.

The lines below the *What now?* prompt show that the alias worked; a message sent to *group* was mailed to *janed* and *johnd*.

The vi Text Editor

The **vi** editor is a screen–oriented text editor for entering and revising documents, programs, and other text files. For a complete description of **vi**, see *UTek Tools*.

Sample .exrc File

When you begin editing a file, **vi** reads and executes commands from a file named *.exrc* (**vi** is based on the line editor **ex**, hence the filename *.exrc*). The *.exrc* file should contain commands that set up **vi** the way you want it when editing a file. Example 5–6 is a sample *.exrc* file.

NOTE

The line numbers in Example 5-6 are not in the .exrc file; they are included for reference only.

```
1 set autoindent wrapmargin=5 number
2 set noautowrite nomesg nomodeline
3 map q :wq
4 ab  tek Tektronix, Inc.
```

Example 5–6. Sample .exrc File.

Turning on Features of vi (line 1)

Line 1 of Example 5–6 sets the automatic indentation, margin wraparound, and line numbering features of **vi**. You could also set these options while in the editor by typing:

```
:set autoindent wrapmargin=5 number.
```

You can turn these features off by typing:

```
:set noautoindent wrapmargin=0 nonumber
```

Turning off Features of vi (line 2)

Line 2 of Example 5–6 turns off the automatic file write, message receiving while in the editor, and modeline features of **vi**. These features can be turned off while in the editor by typing:

```
:set noautowrite nomesg nomodeline
```

Mapping Characters (line 3)

Mapping lets you rename commands. Line 3 of Example 5–6 renames the **:wq** command (write to the file and quit) to **q**.

Abbreviations (line 4)

The abbreviation feature of **vi** lets a short character string stand for a longer character string while in the *insert* mode. Line 4 of Example 5–6 lets you enter the whole string *Tektronix, Inc.* by typing *tek* followed by a space. You can turn off this abbreviation by typing:

```
:unab tek
```

while in the **vi** editor.

The finger Command

Finger is a command that prints out information about users on your UTek system. Typing:

finger *loginname*

prints information about a specific user.

When another user types:

finger *yourname*

they see information about you. If you have files in your home directory named *.project* or *.plan*, **finger** adds their contents to its output.

Example 5–7 shows an example of the output of the **finger** command for a user whose login name is **joes**. The first line of **joes**'s *.project* file appears after the *Project:* heading. The entire contents of his *.plan* file appears after the *Plan:* heading.

```
$ finger joes
Login name: joes                        In real life: Joe Smith
Mail Station: 30-201                    Home machine: engr1
Office Phone: 555-1234                  Home phone:
Directory: /usr/joes
Last login Fri Apr 13 11:01 on tty1
Project: Engineering Applications
Plan:
       To build a better mousetrap
```

Example 5–7. Output of the finger Command.

6

The Local Area Network

INTRODUCTION

This section describes how to communicate with other hosts on your local area network (LAN).

You can communicate with other hosts on your local area network (LAN) with:

- The Network File System (NFS)
- The remote commands **rlogin**, **rsh**, and **rcp**
- Electronic mail
- The **telnet** and **ftp** programs

First, let's look at what a LAN is in detail.

WHAT IS A LAN?

A *local area network* links together workstations and computers so that users can access more than one workstation or computer on the LAN. Each workstation and computer connected to the LAN is called a *node* or *host* of that network.

Usually, each node is connected to a coaxial cable by devices called *transceivers*, which assure that each node on the network transmits and receives data properly. (There are other ways to connect to a LAN, but this arrangement is typical.) Each workstation has one LAN port installed, designated *lan0*. The LAN interface can be accessed through a port on the back panel of the workstation. Figure 6-1 shows the hardware components of a typical local area network.

Each node on the LAN has two unique addresses: an *Internet address* and an *Ethernet address*. The network software uses a host's Internet address when generating messages to send to another host. The Internet software layer passes the address and messages to the Ethernet layer, which sets up communication with the addressed system and passes the messages. The network hardware uses the Ethernet address to communicate with the network. Ethernet addresses are assigned at the factory by Tektronix. Each host on the LAN also has a unique *hostname* for that LAN, which you can use instead of the Internet address to generate messages to send over the network. As part of installing the workstation, you (or the system administrator) must select an Internet address and a hostname for your workstation. Be sure to register the hostname you pick with the LAN administrator, a system administrator or other user responsible for monitoring the LAN, who can tell you if the name you chose is unique on the LAN.

A node that is connected to two or more different LANs is called a *gateway node*. A gateway node lets users on one network access hosts on another network (see Figure 6-2). Through a gateway node, you can communicate with hosts on other networks using the remote commands and electronic mail.

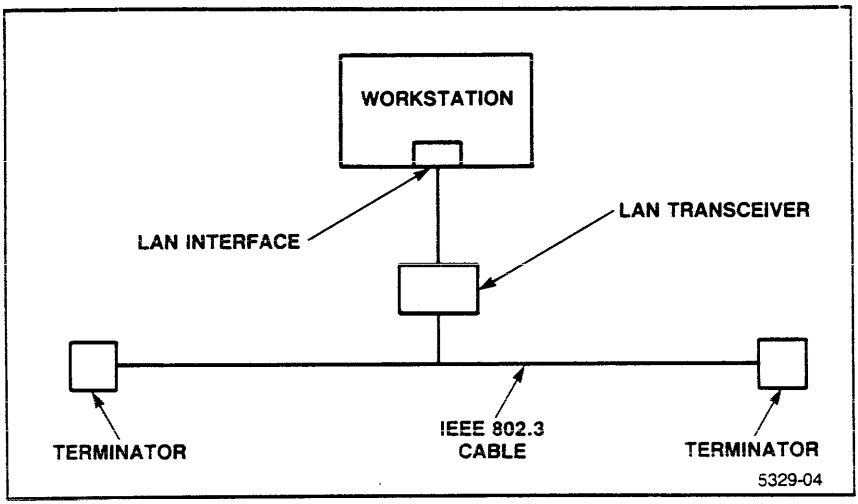

Figure 6-1. Local Area Network Components.

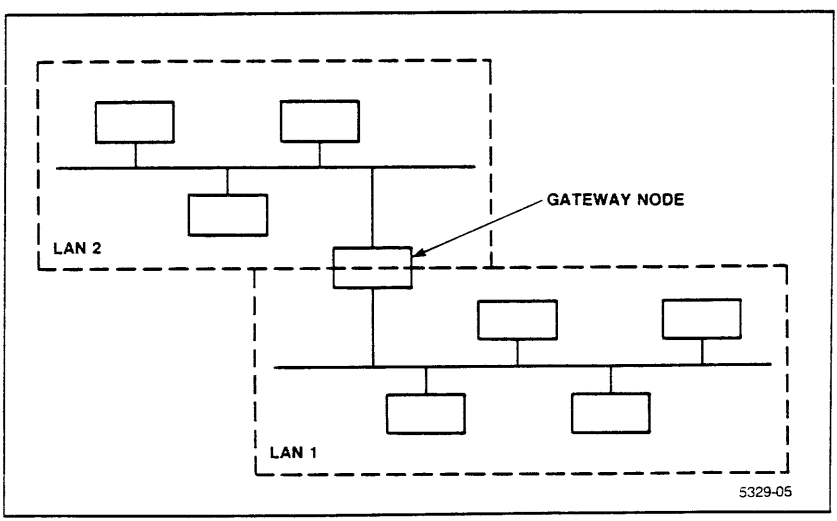

Figure 6-2. Gateway Node.

NETWORK FILE SYSTEM

The Network File System (NFS) provides you with access to files on other workstations. NFS uses an *open architecture* that allows you to access files on non-UTek systems if they are connected to your LAN, either directly or through a gateway node, and running NFS.

NFS allows transparent remote access to file systems on other hosts. It permits you to work with directories or entire file systems on remote hosts as though they existed on your own workstation. NFS can give you access to large databases, extensive documentation, and application programs, without using up disk space on your workstation or CPU time.

With NFS, you do not have to use the remote commands (**rsh**, **rlogin** and **rcp**) to run commands or access files on remote hosts. NFS can eliminate file redundancy, since you do not need a local copy of a remote file or directory.

NFS is designed as a network interface, rather than an extension of UTek or any other UNIX operating system, that lets the NFS user share data with different hosts and operating systems. The resulting *heterogeneous computing environment* can take advantage of different process capabilities. An example of such an environment is shown in Figure 6-3. The UTek workstation using NFS can share data with a personal computer or mainframe without data conversion problems.

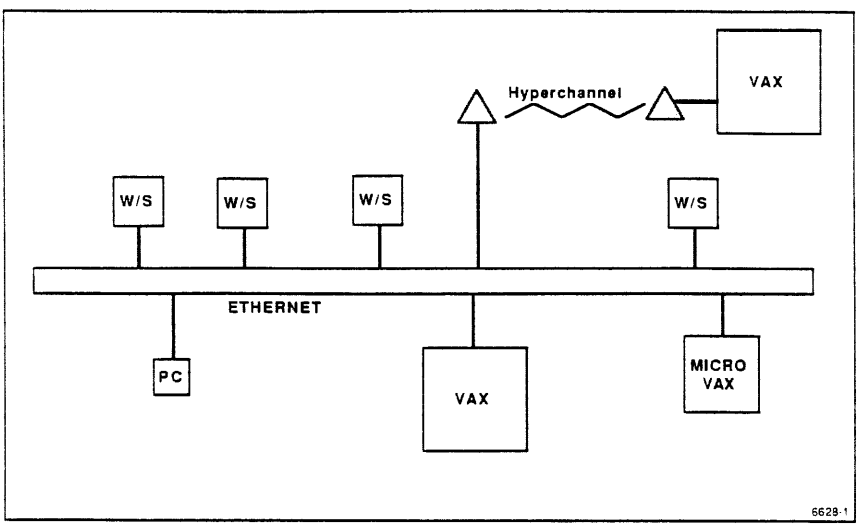

Figure 6-3. Typical Heterogeneous Computing Environment.

Client and Server Modes

Before you can access remote files, directories, or commands, two things must be done:

1. The remote system must make the files, directories, or commands available (*export*).
2. The local system must obtain the files, directories, or commands (*mount*).

A user who *mounts* a file system or directory from a remote workstation is using NFS in *client* mode. As a client, you can choose a point in your file system to have the desired remote file system or directory mounted. A *mount* is a point in the file system to which the remote host's file system (or directories within the file system) are "attached." There is no physical connection between the file systems, however; the mount point is the point in the directory structure where access to the remote file system becomes transparent. The remote file system appears to you (the user) as a branch in your workstation's directory tree.

The host providing the file system or directory to the client is in NFS *server* mode. It allows the client to use its resources. A host can be a *dedicated server*, such as one that contains a special database. A host can be both client and server at once, providing access to directories to users on remote hosts (server mode) while simultaneously accessing file systems on other remote hosts for workstation users (client mode).

Stateless versus Stateful Servers

Server mode is relatively simple because it is *stateless*. Stateless means the NFS server does not remember any previous client requests or other client information. Instead, the client process must keep track of such things as which NFS files are open and where the client is in an individual file. That allows any operation to be requested more than once, with each request containing all the information necessary to service it. Besides simplicity, a stateless server improves performance and allows easy crash recovery because the server doesn't know if the client crashes (and the client can also recover from a server crash).

In contrast, a *stateful* server does keep track of the client, with consequent decrease in performance, increase in complexity, and difficulty in recovering from system crashes.

How NFS Works

This section shows graphically what NFS does.

If you are working at workstation A, and you want to use the online manual pages from workstation B, the directories containing the manual pages must be mounted on your workstation file system.

NOTE

Files can only be mounted by the superuser (root) *(or* system administrator*). It is possible to mount entire file systems or only portions of the file system tree. The syntax and procedures for mounting files are covered in the* NFS Reference Manual.

Suppose your simplified file system on workstation A looks like Figure 6-4:

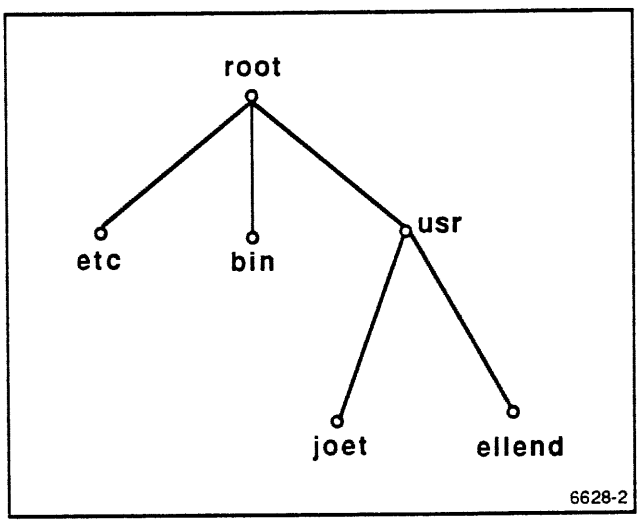

Figure 6-4. The File System on Workstation A.

You want to get the manual pages from workstation B. Its file system looks like Figure 6-5:

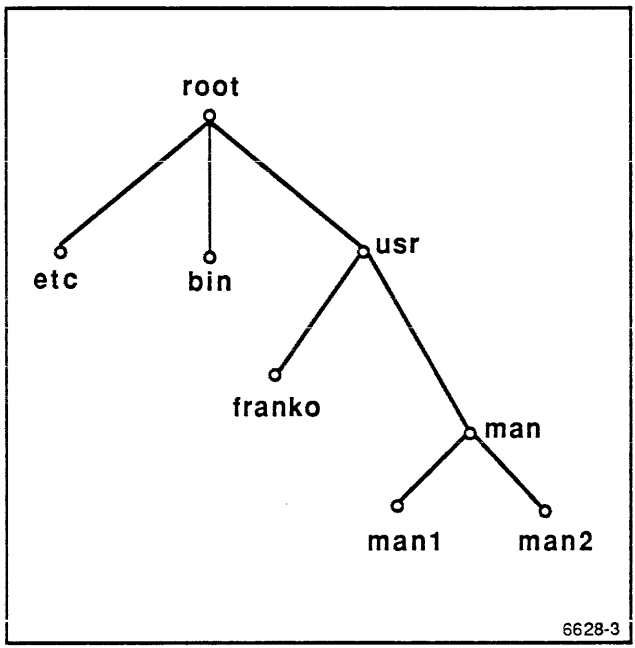

Figure 6-5. The File System on Workstation B.

The "mount point" for the directories on workstation A can be anywhere on its file system. For example, if the remote files are mounted at the *usr* directory, then you can access the manual pages directory, */usr/man*, at that point.

When the remote file system is mounted, your file system on workstation A is connected to workstation B as shown in Figure 6-5. The manual pages are now effectively part of workstation A's file space, and you (as user) can access them as though they belonged to you.

It is possible for a workstation to have more than one remotely mounted file system. For example, you can access a directory called *Work_Files* on workstation C, if that directory is mounted at a different mount point within workstation A's file system, as shown in Figures 6-6 and 6-7.

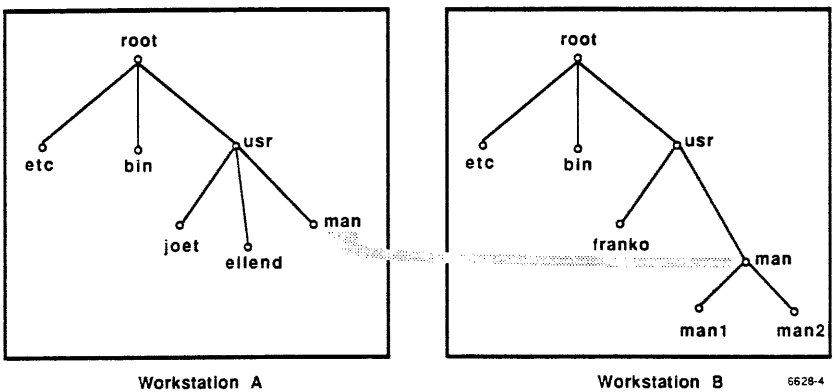

Figure 6-6. The Remotely Mounted Directory.

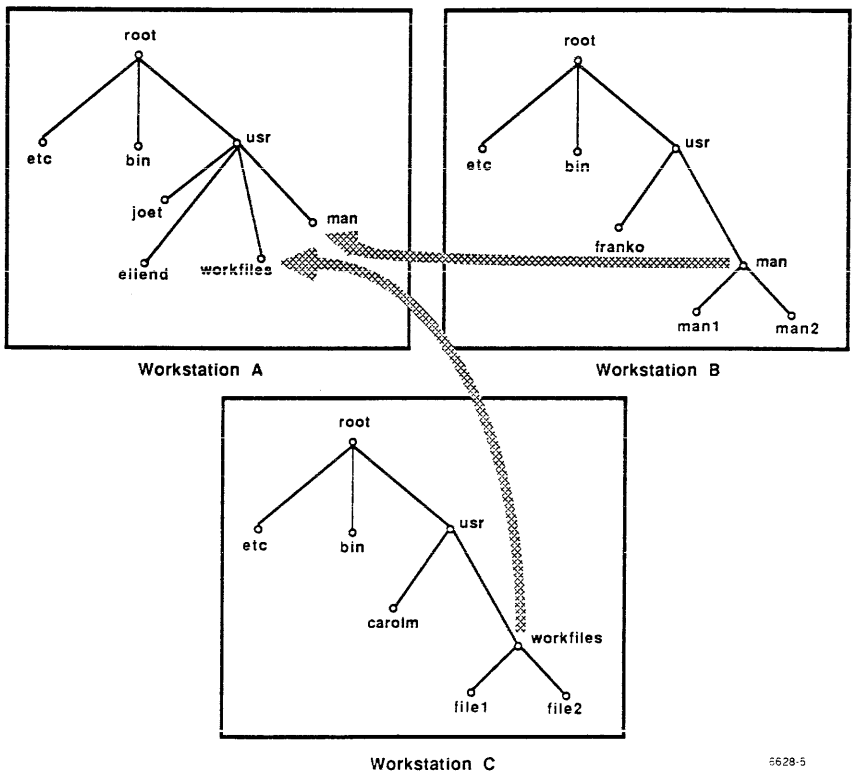

Figure 6-7. Mounting Another Remote Directory.

Finding Mounted File Systems

The */etc/fstab* file contains a list of the file systems currently mounted. You can use the more or cat command to find out if the file system you want is currently mounted. For example, if you wanted to know if the man pages were mounted, you could type:

```
more /etc/fstab
```

A typical */etc/fstab* is shown in Example 6-1.

Example 6-1. .

```
/dev/ds00a              /      2.4    rw                  1      1
stationb:/usr/User_Man   /usr   nfs    bg,rw,soft          0      0
stationc:/usr/Work_File  /usr   nfs    fg,rw,soft          0      0
```

The example */etc/fstab* lists the root file system */dev/ds00a*, running UTek 3.0 and up, with read and write permissions. The two remotely-mounted file systems (in this case, they are really directories) are shown with their home machines and mount points on the home machines, the mount point on the local machine (*/usr*), plus other information. The system administrator can edit the */etc/fstab* file; if you need more information on this file, refer to the *NFS Reference Manual*.

The /etc/hosts.equiv File

The */etc/hosts.equiv* file contains a list of trusted hosts; that is, hosts (or specific users on a host) that can access files on your workstation. When a command is received from a remote host, NFS checks the */etc/hosts.equiv* file. A line consisting of a simple host name entry means that anyone logging in from that host who has an */etc/passwd* is trusted. (An entry in the */etc/passwd* file is not equivalent to having an account on the local host – you can have a "null" password entry simply for execution of remote commands.) If a particular user on a remote host is given access to files, then a second field consisting of the user's login name is added. It is separated from the first field by a single space. Note that although anyone can read this file, only *root* can edit it.

For NFS, two other types of entries are possible. A line consisting of *+@group* means that all hosts in that network group are to be trusted. A line consisting of *-@group* means that all hosts in the given network group are NOT to be trusted. Therefore, a negative entry in the */etc/hosts.equiv* can prevent a remote file mount. A sample */etc/hosts.equiv* is shown in Example 6-2.

Example 6-2. .

```
tekstation1
tekstation2 mikey
+@engineering1
-@engineering2
```

The */etc/hosts.equiv* file is also used by the remote commands and is discussed under that topic later in this section.

The .rhosts File

The *.rhosts* file is local to user accounts, that is, you create this file in your home directory to control access to your own account by users on a remote host. The same format used for */etc/hosts.equiv* is used for *.rhosts*. However, a minus entry in */etc/hosts.equiv* is overridden by the *.rhosts* file, if that file has a positive entry.

The *.rhosts* file is also used by the remote commands and is discussed under that topic later in this section.

Yellow Pages

The Yellow Pages (YP) are an optional network data service. The YP service provides automatic network security for each host by double-checking access privileges whenever a remote host tries to mount local files or directories or a remote user wants local service. It is not necessary to use the YP service to prevent unauthorized access to local files or services, but the YP service makes the task easier and the policy consistent.

The YP service maintains information files, such as password, group, net address, network host, and mail alias files. These files are normally found in the directory */etc* on every machine in the network. With the YP, only a subset of the network hosts—the master server and slave server(s)—have these files in data form. More importantly, if one of these files is changed, the changes are sent to the master server and automatically distributed to the slave server(s). Without the YP, the system administrator must maintain these files on each workstation in the NFS network. On a small network, it is not difficult to manually maintain the files in */etc*, but it is still time-consuming. However, on a large network, the task can be monumental.

The on Command

You can execute commands on a remote system—even if that system is not binary compatible with your workstation—with the **on** command. The **on** command allows you to execute commands on any system, provided that system has *exported* the directory that contains the command.

The syntax of the **on** command is:

on [-i] [-n] [-d] *host command* [*argument*]

where -i is interactive mode, -n is no input mode (for background jobs), -d is debug mode, *host* is the remote machine on which you are running the command, *command* is the command you are running, and *argument* is any argument(s) to the command. See *on*(1c) for more information.

THE REMOTE COMMANDS

If you don't have Network File System (NFS) access to other workstations on the network, you still may be able to access those workstations with the remote commands.

You can log in to another machine on the network with the **rlogin** (remote login) command, you can execute UTek commands on another machine on the network with the **rsh** (remote shell) command, and you can copy files to and from another machine on the network with the **rcp** (remote copy) command.

The rlogin Command

You can log in to another workstation connected to the same LAN as your workstation or to a machine on another network (if your network contains a gateway to that network) with the **rlogin** command.

The format of **rlogin** is:

rlogin *machine* [–l *loginname*]

where *machine* is the name of the computer or workstation you are logging in to. The –l argument lets you log in to another workstation under another login name.

You could log in to a remote machine named *engr1* by typing:

```
rlogin engr1
```

You can log out of the other machine and return to your workstation by logging out as you normally would (by pressing <CTRL-D> or typing **logout**).

You can send a limited number of signals back to your own workstation while logged into another machine by beginning a command line with a tilde (~). You can tell your workstation log you out of the remote machine by typing:

```
~.
```

If you are a C-Shell, user you can pause your session on the remote machine by typing:

```
~<CTRL-Z>
```

and resume your session with the fg command. You can change the command character from the tilde (~) if you want to. See *rlogin(1N)* for more information.

The rsh Command

The **rsh** (remote shell) command executes a single UTek command on another machine on your LAN or on a machine on another network (if your network contains a gateway to that network).

The format of **rsh** is:

rsh *machine* [–l *loginname*] *command*

where *command* is the name of a UTek command to be executed on the remote machine. The –l option lets you specify a login name other than your own.

The following example uses the **rsh** command and the **cat** command to read the contents of a file named */usr/joe/datafile* that is on the machine named *engr1*.

```
rsh engr1 cat /usr/joe/datafile
```

A user named *joe* could use **rsh** to list the contents of his bin directory on *engr1* by typing:

```
rsh engr1 ls /usr/joe/bin
```

When you use **rsh** to execute a command that contains characters that have a special meaning to your shell (such as |, >, and *), you should remember the following rules:

- Special characters are normally interpreted by your local machine.
- You can have the special characters interpreted by the remote machine by enclosing the character in quotation marks.

For example, the following command uses the **who** command to create a file named *localfile* on your workstation.

```
rsh engr1 who > localfile
```

But if you enclose the > character in double quotation marks, the same **rsh** command creates a file on *engr1*.

```
rsh engr1 who ">" remotefile
```

Your rsh Environment

When you use **rsh** to execute a command on a remote machine, environment variables on the remote machine control how the command is executed. These variables are set differently for Bourne Shell users than for C-Shell users.

If you are a Bourne Shell user, your HOME, SHELL, and USER environment variables are taken from the */etc/passwd* file. Your PATH environment variable is given a default value. If you are a C-Shell user, these environment variables are set in the same way, but the commands in your *.cshrc* file are executed, which lets you change the value of the environment variables.

If you don't enter the full pathname of the command you are executing with **rsh**, the remote machine begins looking for the command in the directories listed in the PATH variable. If you specify a filename as an argument to **rsh** and the filename doesn't begin with a slash (/), the remote machine begins looking for the file in the directory specified in the HOME variable.

To find the values of your environment variables on the remote machine, use the following **rsh** command:

rsh *machine* **printenv**

where *machine* is the name of the remote machine.

See *rsh(1N)* in the *UTek Command Reference* manual for more on the **rsh** command.

The rcp Command

The **rcp** (remote copy) command copies files between workstations on your LAN or between a workstation on your LAN and a machine on another network (if your network contains a gateway to that network). The format of **rcp** is:

rcp *fromfile tofile*

Either *fromfile* or *tofile* is a file on your workstation, while the other file is on the remote machine. You specify a file on a remote machine with the syntax:

machine:filename

If filename doesn't begin with a slash, the remote machine looks for the file in your home directory on the remote machine.

For example, a user named *joe* could copy a file named *data* from his home directory on *engr1* to a file named *data* on his workstation with either of the following two commands:

```
rcp engr1:/usr/joe/data data
rcp engr1:data data
```

The –r option of **rcp** lets you copy a directory and everything below it to a remote machine. The –r option also lets you copy more than one file to a remote machine.

See *rcp(1N)* in the *UTek Command Reference* manual for more information on the **rcp** command and its options.

Remote Command Protection

To access another machine on the network with the **rlogin**, **rsh**, and **rcp** commands, you must have:

- Remote command access to the other machine, which is permitted by entries in the files named */etc/hosts.equiv* and *.rhosts* on that machine.
- An account on the other machine or access to another user's account on that machine.

When you enter one of the remote commands (**rlogin**, **rsh**, or **rcp**) to access another machine on the network, the remote machine takes the following steps to decide whether or not to let you access it:

1. The remote machine checks to see if your workstation is listed in the */etc/hosts.equiv* file. If your machine is listed in this file then you and all other users on your workstation can access the remote machine.
2. If your machine is not listed in the */etc/hosts.equiv* file, the remote machine looks for a file named *.rhosts* in your home directory (or in the home directory of the person you are trying to log in as) on the remote machine. If you are listed in this file, then you can access the remote machine.
3. The remote machine checks to see if you (or the person you are trying to log in as) are listed in the password file (*/etc/passwd*). If you are listed in the password file and satisfy either 1 or 2, then you are allowed to access the remote machine.

Even if you gain access to a remote machine, there is no guarantee you can access files on that machine. The remote commands check the protection bits (permissions) of a file before letting you access that file.

The /etc/hosts.equiv File

The */etc/hosts.equiv* file contains the names of all the machines on the network whose users can access your workstation with the **rlogin**, **rsh**, and **rcp** commands. Similarly, the */etc/hosts.equiv* files on other machines control who can access those machines. For more on the */etc/hosts.equiv* file, see the *6130 System Administration* manual and *hosts.equiv(5)* in the *UTek Command Reference* manual.

The .rhosts File

If you want to gain access to a remote machine or if you want to **rlogin** to a remote machine without having to enter a password, you should create a file named *.rhosts* in your home directory on the remote machine (or in the home directory of the person you are going to log in as, if you are using the –l option of **rlogin**).

Each line in your *.rhosts* file has the format:

hostname loginname

where *hostname* is the name of your local machine and *loginname* is your login name or the login name of someone else you want to be able to log in to your account.

For example, if a user named *joe* wants to **rlogin** to *engr1* from *workst1* without entering a password, he should put the following line in the *.rhosts* file in his home directory on *engr1*:

```
workst1 joe
```

In addition, if *joe* wanted a user named *mary* to be able to **rlogin** to his account on *engr1* from *workst1*, he could put the following line in his *.rhosts* file:

```
workst1 mary
```

For more on the *.rhosts* file, see the *6130 System Administration* manual.

ELECTRONIC MAIL

By using the programs of the MH mail system, you can exchange electronic mail with users on other machines on your network. If your network contains a gateway node, you can also exchange mail with users on machines on other networks.

If you have never sent mail using the MH mail programs, you should read *The MH Mail System* in *UTek Tools (Volume 1)* before reading this section.

To send mail to another user on your network, create a *To:* line in your mail message that has the following format:

To: *user@hostname*

where *user* is the name of the user and *hostname* is the name of the machine that person is on.

Example 6-3 shows how a user can send electronic mail to a user named *joe* on the machine named *engr1*.

In Example 6-3, the command line invokes the **comp** program with the **prompter** text editor to create a mail message. The user enters the mail message and presses <CTRL-D> at the end of the message. The **comp** program responds with **What now?**. The user types **send -v**, which sends the mail message and prints messages about what it is doing.

```
$ comp -editor prompter
To: joe@engr1
Cc:
Subject: Meeting today
--------
Let's meet in the conference room today at 1:ØØ
to discuss the widget proposal.
<CTRL-D>
--------

What now? send -v
joe@engr1... Connecting to engr1.ether
joe@engr1... Sent
```

Example 6-3. Sending Mail to Another Machine.

The last two lines of Example 6-3 show the response of the **send** program. The **Connecting to engr1.ether** message is printed when the user's machine is establishing a connection to *engr1*. The **Sent** message is printed when the mail message is sent to *engr1*.

To send mail to a user on a network that is connected to your network through a gateway node, you must use the syntax of the mail system that is on the gateway node on the **To:** line of your mail message.

Your mail system can be configured to know the paths through your gateway to machines on other networks, so you can use the same *username@machine* syntax to send mail to those machines. See the *6130 System Administration* manual for more information.

If your gateway has a UNIX or UNIX-like operating system, you can send mail to a machine on another network by specifying all the machines the mail message must pass through, separated by exclamation marks (!). For example, if you want to send mail to a user named *joe* on a machine named *engr1* and the gateway between your network and Joe's network is a UNIX host named *host1*, you would use the following **To:** line in your mail message:

```
To: host1!engr1!joe
```

For more on addressing mail to other networks, see *mailaddr(7)* in the *UTek Command Reference* manual.

Forwarding Your Mail

If you have accounts on more than one machine on the network, you can receive all your mail on your home machine, even if that mail is sent to another machine. To have all your mail forwarded to your home machine, create a file named *.forward* in your home directory on every machine you have accounts on *except* your home machine.

The *.forward* file should contain one line that has the following format:

yourname@yourmachine

where *yourname* is your login name and *yourmachine* is the name of your home machine.

THE UPTIME COMMAND

The **uptime** command with the **–r** option prints the status of the other machines on your network. Example 6-4 shows an example output of **uptime –r**.

The values of the first line of output are used to explain the meaning of the columns.

billthecat	The name of the machine.
up 1+03:47	How long the machine has been up or down. The number 1+03:47 shows that this machine has been up for 1 day plus 3 hours and 47 minutes.
0 users	The number of users logged in to the machine.
load 0.01, 0.00, 0.00	The load average of the machine. The *load average* is an average of the number of processes that are ready to be run over a period of time. The first number is averaged over the last minute, the second number over the last 5 minutes, and the third number is averaged over the last 15 minutes.

See *uptime(1N)* in the *UTek Command Reference* manual for more information.

There is an older version of the **uptime –r** command that is provided for compatibility with other systems based on Version 4.2 BSD UNIX. This command, called **ruptime**, puts an extra load on the local area network and should therefore be avoided. See *ruptime(1N)* in the *UTek Command Reference* manual for more information.

```
$ uptime -r
billthecat    up 1+Ø3:47, Ø users, load  Ø.Ø1,  Ø.ØØ,  Ø.ØØ
c3po        down   19:42
greeto        up   18:48, Ø users, load  Ø.21,  Ø.Ø4,  Ø.Ø2
gumby       down    Ø:1Ø
nomad       down 2+ØØ:51
olympus       up   19:34, 1 user, load  Ø.Ø6,  Ø.ØØ,  Ø.ØØ
shark         up   19:26, 38 users, load  6.26,  7.7Ø,  9.65
tinker        up   19:34, 2 users, load  Ø.17,  Ø.24,  Ø.17
```

Example 6-4. The uptime –r Command.

TELNET AND FTP

The **telnet** and **ftp** programs let you communicate with machines on your network using different network protocols. These commands are primarily used to communicate with machines on your network that aren't 6130 workstations. **Telnet** is an interactive program that lets you establish a connection with another machine using the TELNET protocol. **Ftp** is an interactive program that lets you transfer files between your workstation and another machine using the ARPANET File Transfer Protocol.

For more on these commands, see *telnet(1N)* and *ftp(1N)* in the *UTek Command Reference* manual.

/* */ 7

Programming

This section provides an overview of the programming facilities available for the workstation. Some of the software discussed in this section is optional and therefore may not be on your workstation.

Shell Programming

You can use the standard shells of the workstation, */bin/sh* and */bin/csh*, as programming languages as well as interactive command interpreters. Both of these shells contain variables, parameter passing, string substitution, and control-flow constructs (such as *if* statements and *while* loops) that let you write shell programs (also called *shell procedures* or *shell scripts*).

Shell programs are generally faster to write and debug than programs written in other programming languages. Writing a shell program is easy because each line in the program is a call to a working UTek program, written just as you would type it at the keyboard. Debugging is easy because you don't have to recompile, relink, and reload your program every time you make a change.

See *sh(1)* and *csh(1)* sections of the *UTek Command Reference*, the *Common Tools* section of *UTek Tools*, and in the *Programming the UNIX Shell* section of *Introducing the UNIX System*. In Section 5, *Customizing your Account*, Example 5–1 is a Bourne Shell program and Examples 5–2 and 5–3 are C–Shell programs.

Programming Languages

The C, FORTRAN, BASIC, and Pascal programming languages are available on UTek. These languages are documented in various books, which are introduced in this section. Tektronix enhancements to these languages and system–specific information about the languages are documented in the *Programming Languages* section of *UTek Tools*. How to invoke these compilers, interpreters, and tools is documented in the *UTek Command Reference*.

C

The C programming language is the most commonly–used language of the UTek system. Most of the UTek kernel, shells, tools, and utilities are written in C. The syntax and usage of C is documented in *The C Programming Language*. Extensions to C and information about the Tektronix implementation of the language are described in *UTek Tools*. How to invoke the C compiler is documented in *cc(1)* in the *UTek Command Reference*.

The C compiler used on UTek, **cc**, is based on Berkeley's 4.2 BSD version of the compiler but has been extended to be compatible with AT&T's System V version of the compiler.

A large number of libraries and *include* files defining commonly-used data structures and functions are provided in the */usr/lib* and */usr/include* directories.

FORTRAN

The FORTRAN language provided by UTek is Fortran 77, the official American National Standard FORTRAN language. Fortran 77, which is based on Fortran 66, is described in *Fortran 77 Reference*. Where **f77** deviates from, or expands upon, the Fortran 77 standard is documented in the section entitled *The FORTRAN Compiler* in *UTek Tools*. How to invoke the FORTRAN compiler is documented in *f77(1)* in the *UTek Command Reference*.

In addition to Fortran 77, UTek provides EFL (an Extended FORTRAN Language) and Ratfor (Rational FORTRAN). Ratfor and EFL are documented in *ratfor(1)* and *efl(1)* in the *UTek Command Reference* and in the *Ratfor* and *EFL* sections of *UTek Tools*.

BASIC

Tektronix ANSI BASIC is an extended version of the American National Standards Institute's proposed BASIC programming language. All of the standard features and most of the optional features of ANSI BASIC are implemented. Tektronix extensions to the ANSI standard make the language more versatile and better suited to engineering applications. To get started using Tektronix ANSI BASIC see the *Tektronix ANSI BASIC Learning Guide*. For a complete description of the language see the *Tektronix ANSI BASIC Keyword Dictionary*.

UTek provides a BASIC compiler and an interactive BASIC compiler. The BASIC compiler, called **bbc**, reads a BASIC program and produces executable code. For more information, see *bbc(1)* in the *UTek Command Reference*. The interactive BASIC compiler, called **basic**, lets you enter, debug, and run a BASIC program; see *basic(1)* in the *UTek Command Reference*.

Pascal

The Pascal language available on UTek is based on Version 2.0 of the Berkeley Pascal language. You can find a description of the Tektronix implementation of Pascal in the *Pascal* section of *UTek Tools*. The Pascal language is described in the *Pascal User Manual and Report*. How to invoke the Pascal compiler, **pc**, is documented in *pc(1)* of the *UTek Command Reference*.

Programming Support Tools

The programs described in the following paragraphs are provided by UTek to help you write, debug, and execute application programs on the workstation.

Linking Object Files

The UTek link editor **ld** combines several object files (files that end in *.o*) into one, resolves external references, and searches libraries. **Ld** can combine object files produced from programs written in different languages. If no errors are found, **ld** creates an executable file named *a.out*. For more on **ld**, see *ld(1)* in the *UTek Command Reference*.

Maintaining Source Code

Some problems often encountered in a large programming project are: keeping your software up to date, maintaining multiple versions of programs, and coordinating the efforts of more than one programmer. UTek provides programs to help you solve these problems.

make

Make is a program that simplifies the task of keeping software up to date. **Make** keeps track of which files need to be reprocessed or recompiled after a change is made in some part of the code. When **make** finds the files that need to be updated, it follows built–in rules or rules you provide for producing an up–to–date program. For more on **make**, see the section entitled *Make* in *UTek Tools*.

RCS

The Revision Control System (RCS) is a group of programs that maintain multiple versions of a source program in a single file. RCS lets you store and retrieve different versions of a program, merge different revisions of a program together, and control who may access and modify source files. RCS also saves you disk space by storing only the changes made for each revision of a program. For more on RCS, see the section entitled *RCS* in *UTek Tools*. To use RCS and **make** together, see the section entitled *Using RCS and Make Together* in *UTek Tools*.

cb

The C beautifier program **cb** reads a C program from the standard input, reformats it with proper spacing and indentation, and prints the *beautified* version to the standard output. For example, to see the effects of **cb** on the file *program.c*, type:

```
cb < program.c
```

To format the C program and save the output in a file named *beautiful.c*, type:

```
cb < program.c > beautiful.c
```

For more on **cb**, see *cb(1)* in the *UTek Command Reference.*

Debugging Aids

The programs described below help you locate compiler and run–time errors in your programs.

sdb

You can use the symbolic debugger **sdb** to track down run–time errors in your FORTRAN or C programs. The **sdb** program examines *core* files, which are images of memory of an aborted program, and monitors and controls a running program.

The **sdb** program lets you interact with the program you are debugging at the source language level. When debugging a *core* file produced by an aborted program, **sdb** tells you which line in the program caused the error and lets you access all variables in the program by name. For more on **sdb**, see the *Sdb* section of *UTek Tools.*

adb

The **adb** program is a general purpose debugger that finds run–time errors in programs at the assembly language level. For more on **adb**, see *adb(1)* in the *UTek Command Reference* and the *Adb* section of *UTek Tools.*

lint

The **lint** program examines C language source programs for code that is likely to contain bugs or is non–portable, or wasteful. **Lint** also checks the type usage of your program more strictly than the C compiler. For more on **lint**, see the *lint(1)* in the *UTek Command Reference* manual.

error

The **error** program analyzes and describes error messages produced by a number of compilers and programming tools. **Error** replaces the painful, traditional method of scribbling error messages down on paper as they scroll off your screen, by inserting the messages in your source file, above the line that caused the error.

For more on **error**, see *error(1)* in the *UTek Command Reference.*

Error Messages

Table 7–1 lists error messages that you may get when running a program, along with the most common causes of the errors.

Table 7–1
RUN–TIME ERROR MESSAGES

Message	Common Causes
Bad magic number	Input filename does not end in the proper suffix. See the following text on file naming.
Bus Error	Types of arguments and parameters don't match. I/O problems — reading past EOF, reading a closed file, bad file pointers, etc. Arrays improperly dimensioned. Referencing a nonexistent bus device. NULL or uninitialized pointer, subscript out of range.
Memory Fault	Subscripting arrays past the memory allocation for your program. NULL or uninitialized pointer, subscript out of range. Attempts to reference data outside valid address space. Parity errors in address space. Recursion errors.
Trace/BPT Trap	Trace bit set in the Processor Status Word. Trying to tell the program to go to an address that doesn't exist. Executing data or destroyed instructions.
Floating Exception	Invalid floating point operation: overflow, underflow, divide by zero, log of a negative number, float to integer conversion overflow.
Segmentation Fault	Subscript out of range. Using a string that is not NULL terminated.
Illegal Instruction	Attempt to execute a privileged instruction, such as *halt*. Execution of nonsense instruction (For example, jumping to a register).

File Naming

UTek has rules for naming files containing source, object, and executable programs. You can name your files anything you want as long as the names end in the suffixes listed below:

.b	BASIC source file
.f	FORTRAN source file
.F	FORTRAN source file with C code embedded
.e	EFL source file
.r	Ratfor source file
.p	Pascal source file
.c	C source file

Object files produced by a compiler end in the *.o* suffix. Executable programs produced by a compiler or the loader are placed in a file named *a.out* unless you rename it; see the documentation of the compiler you are using in the *UTek Command Reference* manual.

Graphics

You can write programs that generate computer graphics by using the Graphical Kernel System (GKS) primitives, which are available to programs written in the C, FORTRAN, and BASIC programming languages. For more on GKS see *GKS C*, *GKS FORTRAN*, and the *BASIC Keyword Reference*.

8

GPIB Programming

Each GPIB interface uses a TMS9914A GPIB controller integrated circuit. The high-speed GPIB interface boards include additional hardware that raises the data rate on the bus.

The GPIB implementation for Tektronix workstations complies with the IEEE Std. 488-1978 and its supplement IEEE Std. If you are unfamiliar with the GPIB structure and protocol, read Appendix A, GPIB Concepts, before continuing in this appendix.

This section describes the implementation of the IEEE GPIB Standard for the 6130 Intelligent Graphics Workstation. The following topics are described in this section:

- The structure of the GPIB driver and ways to access it.
- GPIB configuration: how to define characteristics of each GPIB interface on the workstation.
- GPIB subsets supported by the GPIB A and B drivers.
- GPIB example programs written in the C language.

Section 9, BASIC GPIB Support, describes:

- Tektronix ANSI BASIC support for GPIB
- GPIB programming hints

Appendix A, GPIB Concepts, discusses GPIB structure and protocol.

Appendix B, Tektronix Standards, Codes, and Formats, describes the language used by Tektronix IEEE 488 instruments to communicate over a GPIB.

Appendix C, ASCII/GPIB Chart

THE GPIB DRIVER

An application program written for a Tektronix Intelligent Graphics Workstation communicates with instruments attached to a GPIB port through a system program called the *GPIB driver.*

NOTE

ANSI BASIC only works with the gpiba driver.

The GPIB driver handles all low-level GPIB tasks, but does not perform any interpretation of device-dependent data received from GPIB instruments. Transfer protocols must be handled by the programming language (for example, C) and the application program.

Application programs read and write either unformatted data, or data formatted according to the Tektronix Standard Codes and Formats. For more on this standard, see Appendix B.

The GPIB driver consists of:

- One *port configuration* component for each GPIB interface on your workstation. This component initializes the GPIB interface and provides diagnostic messages to application programs.
- An *operational* component that lets your application programs communicate with the GPIB interface and instruments connected to the GPIB.

Figure 8-1 shows the organization of the GPIB driver.

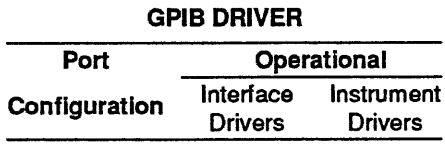

Figure 8-1. GPIB Driver Organization

Port Configuration Driver

The port configuration component, called the *port configuration driver*, is used to set up the GPIB interface before you run your application program. The port configuration driver also provides diagnostics when your program is running. Your application program never needs to access the port configuration driver.

The port configuration driver of the GPIB drivers has names that are in the following format:

/dev/gpid*n*

where *n* is the *slot number* of the GPIB interface. The *slot number* is the number of the slot in the workstation's backplane that contains the GPIB interface board.

There are now two gpib drivers. The old driver is now **gpiba** and the new driver is **gpibb**. The new driver fixes many of the known problems, plus some enhancements which the gpiba driver does not have. See Table 8-4 for a comparision of the two drivers. For additional information about the new gpibb driver, refer to the *6130/4132 Exceptions and Extension* manual.

For more on the port configuration driver, see *gpid* in the *6130/4132 Extensions & Exceptions* manual.

Operational Components

The operational component of the GPIB driver is divided into two parts:

- Interface drivers
- Instrument drivers

Your application program can use either of these two drivers to communicate with GPIB instruments. If your program accesses the GPIB through an interface driver, your program must address each instrument as a talker or listener, using GPIB protocols. Then, your program can send device-dependent commands and data to the instrument over the bus. With this method you must know GPIB protocols, but you can make diverse instruments interact in complex ways.

If your application program uses the instrument drivers to communicate with instruments on the GPIB, you don't have to worry about GPIB protocols. Your program communicates with instruments directly. This method simplifies the communication by letting you control a GPIB instrument as if it were any other I/O device on your workstation.

There is one interface driver for each GPIB interface on your workstation. There can be up to 15 instrument drivers (one for each instrument on the bus) for each GPIB interface on your workstation.

Interface Drivers

Your application program can control instruments connected to the GPIB by sending GPIB protocol commands directly to the GPIB interface driver. By communicating with the interface driver, your program can set up complex interactions between instruments on the bus. By using the interface driver, you have complete control over the bus and over every aspect of bus traffic.

By using the interface driver, your application program controls an instrument by following GPIB protocol. This means your program must send primary (and possibly secondary) addresses, send GPIB messages and, perhaps, device-dependent commands and data to communicate with an instrument. The instrument may respond by sending data or status information to the controller.

One interface driver exists for each GPIB interface on your workstation, for a maximum of seven. The interface drivers are named:

/dev/gpib*n*

where *n* is the slot number of the GPIB interface. The name of the built-in GPIB's interface driver is **/dev/gpib0.**

An application program that uses an interface driver must open a GPIB interface (**/dev/gpib***n*) for reading and writing. Then, the program can communicate with the interface and can address and control instruments attached to the bus by following GPIB protocols.

See *gpib* in the *UTek Command Reference*, for further details on the GPIB interface driver.

Instrument Drivers

By setting up a GPIB instrument driver to control an instrument connected to the bus, your application program can communicate with the instrument without having to know GPIB protocols. In fact, your program doesn't even have to know that the instrument is on a GPIB.

Some examples of GPIB instrument driver names are:

/dev/dmm	A digital multimeter
/dev/pulse	A pulse generator
/dev/scope	An oscilloscope

Notice that the name of each instrument driver begins with **/dev**. When the instrument drivers are created, they will be put in that directory.

An application program that communicates with instrument drivers controls the instruments as if each were the only instrument connected to the bus. The program doesn't need to know whether the instruments are connected to the same GPIB port, or to different ports.

This type of application program begins by opening the instrument driver (for example, **/dev/dmm**) and then issues read and write commands to the instrument driver to control the instrument. The program never has to be concerned with GPIB protocols.

For more information on GPIB instrument drivers, see *gins* in the *UTek Command Reference* manual.

CONFIGURING THE GPIB

The UTek operating system provides commands to help you configure the GPIB. With these commands, you can control the configuration of all the GPIB interface options on your workstation.

The **gpconf** command lets you change the configuration of a GPIB interface by adding, changing, or removing instruments from the current configuration. **Gpconf** can also be used to display the current configuration of your GPIB driver, and/or the instrument connected to them.

The **gpinit** command lets you initialize a GPIB interface or an instrument. The **gpstat** command lets you check the status of instruments on the GPIB. The **gprm** command removes instruments from the current GPIB configuration.

Some of these functions are also available through the programming language you use to send data over the GPIB (C, for example). Usually, however, you will find it more convenient and efficient to use the UTek commands instead of the programming language commands. Refer to the Programming Considerations topic in Section 9 for details.

Table 8-1 summarizes the GPIB commands.

Table 8-1
GPIB COMMANDS

Command	Description
gpconf	Configure GPIB instrument/interface driver
gpinit	Initialize GPIB instrument/interface driver
gprm	Remove a GPIB instrument driver
gpstat	Examine GPIB instrument/interface driver status

You can find more information about these commands in the *UTek Command Reference* manual, Section 1.

Configuring Interface Drivers

A GPIB interface driver is defined and created for each GPIB interface when the system is booted. Because the initial definition of the GPIB interface driver assumes there are no instruments connected to the bus, you must execute the **gpconf** command to configure the interface driver before attempting to use the driver.

Interface driver configuration settings assume default values when not explicitly assigned. Table 8-2 shows these default values and gives their meanings.

Table 8-2
DEFAULT INTERFACE DRIVER CONFIGURATION

Switch	Meaning
addr 0	Primary address is 0
htime 5	Timeout delay of 5 seconds
ptime 0.1	Maximum poll time of 100 milliseconds
eom EOI	End-of-message byte is End or Identify (EOI)
dma	Enables Direct Memory Access, if present
sc	The interface is a *system controller*
tcs	Interface takes control synchronously
–std1	Disable short T1 delay
–vstd1	Disable very short T1 delay

The configuration of a GPIB interface driver affects communication with any instruments attached to the bus.

Configuring Instrument Drivers

You can also use **gpconf** to configure GPIB instrument drivers, which control instruments attached to a GPIB. You decide the name of each instrument driver. For example, a digital multimeter might be named **/dev/dmm**. You also decide the communications characteristics of each instrument driver.

Arguments to **gpconf** determine the name and characteristics of the instrument driver. If you do not specify an argument, default settings for GPIB instrument drivers are used. Table 8-3 shows the default **gpconf** settings for GPIB instrument drivers and their meanings.

Table 8-3
DEFAULT INSTRUMENT DRIVER CONFIGURATION

Switch*	Meaning
htime 5	Timeout delay of 5 seconds
ptime 0.1	Maximum poll time of 100 milliseconds
eom EOI	End-of-message byte is End or Identify (EOI)
dma	Enables Direct Memory Access, if present
poll	Enables automatic polling

* There is no default address for instruments; the address must always be specified. Also there is no default slot; The slot number must always be specified.

SUPPORTED GPIB SUBSETS

The initial implementation of the GPIB is restricted to the GPIB subsets shown in Table 8-4. Subsequent releases of the GPIB will expand GPIB capabilities.

Table 8-4
GPIB SUBSETS

Function	Subset gpiba driver	Subset gpibb driver	Description
Source Handshake	SH1	SH1	Complete capability
Acceptor Handshake	AH1	AH1	Complete capability
Talker	T8	T6	Basic talker, No talk-only mode, Unaddress if MLA received
Extended Talker	TE0	TE0	No capability
Listener	L4	L4	Basic listener,
Extended Listener	LE0	LE0	No capability
Service Request	SR0	SR1	No capability
Remote/Local	RL0	RL0	No capability
Parallel Poll	PP0	PP2	No capability
Device Clear	DC0	DC1	No capability
Device Trigger	DT0	DT1	No capability
Controller	C1	C1	System controller
	C2	C2	Send IFC and take charge
	C3	C3	Send REN
	C4	C4	Respond to SRQ
	C25	C5	Send interface messages, Parallel poll, Take control synchronously
Electrical	E2	E2	Three-state drivers

The currently supported subset only implements part of the GPIB system controller functions. In particular, the A driver is incapable of passing or receiving control. The B driver can pass and receive control.

GPIB Program Examples

These examples are presented to serve as a guide for you to use in writing shell scripts and programs to control GPIB instruments. The programs here are written in C for use with the workstation's compiler. Other compilers on your workstation will obviously function with other programming languages.

Example Number One

This first GPIB programming example illustrates using a *shell script* to configure and use GPIB devices (instruments).

```
#! /bin/sh
#
# Copy a 4051 tape to a set of files
#
# Usage: tpcp [directory-name]
#
# If a directory name is specified, it will be created, and
# the file(s) placed there. Otherwise, the file(s) will be
# put in the current directory.
#
# The following is a list of the GPCONF of the device drivers necessary
# to make this script work. It includes the necessary message terminators
# to prevent sanity gpib timeouts on the workstation by missing a delimiter.
# There are cases where the 4924 will NOT send an EOI e.g. HEADER, ERROR.

# ========== slot b ==========
# Actually re-configure the drivers in case there is an error or difference
# from what is really wanted.

# Tape drive in this program is set to primary address #4.

  # Since the file must be found before a header operation, allow a long
  # handshake timeout to allow the finding of the file prior to this function.
  # Header requires secondary address of 9. (HEADER)
gpconf headr slot b addr 4.9 htime 45 ptime 0.1 eom CR -poll

  # Binary data is stored on tape in special format. Let tape decode it
  # for you. Binary is indicated by secondary address of 14 (READ)
gpconf binary slot b addr 4.14 htime 5 ptime 0.1 eom EOI -poll

  # Allow the tape drive to send tape data without interpretation. Just talk
  # the drive. This function is indicated by secondary address of 26. (TALK)
gpconf readit slot b addr 4.26 htime 5 ptime 0.1 eom EOI -poll

  # Find the file number indicated (ship file number over the bus as device
  # dependent data). Find is indicated by secondary address of 27. (FIND)
gpconf find slot b addr 4.27 htime 5 ptime 0.1 eom EOI -poll

  # Clear the error channel. No other GPIB commands will be executed until
  # the error channel is cleared. Since the tape may have to rewind, keep
  # a long handshake timeout value to wait for the tape to stop.
gpconf error slot b addr 4.30 htime 45 ptime 0.1 eom CR -poll
```

```
 # The line below shows the configuration for the port that this program is
 # to be used on.  Change all the slot #s to the actual slot # you are using.

# gpib5 slot b addr 0 htime 5 ptime 0.1 eom EOI sc tcs

#  First determine whether or not a directory name has been passed as an
#  argument in the call of this script.

case $# in
0)                              # No argument passed, continue on
                continue
                ;;
1)                              # Argument was passed, mkdir then continue on
                mkdir $1
                cd $1
                ;;
*)                              # Too many arguments passed, warn then exit
                echo Too many arguments passed. Usage: $0 [dir]
                echo Program exiting
                exit 1
                ;;
esac

#  We are working with 4051 style tapes which have headers and file numbers.
#  We also have to find out whether the particular file is ASCII, BINARY or
#  LAST.  Files other than ASCII or BINARY are skipped. LAST indicates the
#  of the tape, therefore the end of the script.

#  Set the file number to 0 to start.  It will be incremented each time a file
#  is processed.
FILENUM=0

#  Start the loop.  The first time through the filename is "".
while test "$FILETYPE" != "LAST"
do
                FILENUM=`expr $FILENUM + 1`  # increment file number
                echo $FILENUM > /dev/find  # find the file
                FILETYPE=`gprd -80 headr | awk '{print $2}'`  # read/decode the header
                echo $FILENUM > /dev/find  # find the file again
                case $FILETYPE in                               # decide the type it is
                LAST)
                        gprd -10 error > /dev/null  # 4924 has error on last
                        ;;
                ASCII)
                        echo $FILENUM > /dev/find               # find the file again
                        #  The gpread is similar to the gprd example C program
                        #  It is used to transfer the data from the 4924 to
```

```
                        # the system. The tr translates the <CR>'s to <LF>'s
                        # and then the data is put into a file (e.g. file.2).
                        gpread readit | tr ' 15' ' 12' > file.$FILENUM
                        # At the end of each file, the 4924 has an error to
                        # report (that it is at end of file).
                        # This clears out the error. This also only reads
                        # ten characters max from the device (in case no EOI
                        # has been sent.
                        ERROR='gprd -10 error'
                        case $ERROR in
                        12*)        F error (expected)
                                    continue
                                    ;;
                        *)          # ABORT, wrong error
                                    echo error: $ERROR
                                    exit 1
                                    ;;
                        esac
                        ;;
                BINARY)
                        echo $FILENUM > /dev/find  # find the file again
                        # See above comments on gpread.
                        gpread binary > file.$FILENUM
                        # See above comments on clearing errors.
                        ERROR='gprd -10 error'
                        case $ERROR in
                        12*)        # EOF error (expected)
                                    continue
                                    ;;
                        *)          # ABORT, wrong error
                                    echo error: $ERROR
                                    exit 1
                                    ;;
                        esac
                        ;;
                *)      # If not ASCII or BINARY, skip it. (usually a NEW)
                        echo 2>&1 'basename $0': file $FILENUM X
                                    is $FILETYPE - skipping
                        ;;
                esac
done                    # Loop back to make run on next file on tape.
```

Example Number Two

This second GPIB programming example shows a GPIB program written in C for a serial poll on the bus.

```
/*  This is a program file to return the value from a serial poll performed
 * by the GIOCSPOLL command in GPIB(4) in the
 * 6130/4132 Exceptions and Extensions manual
 * This command will send a SPE and a
 * SPD.
 *
 ******************************************************/

#include <stdio.h>
#include <strings.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <box/gpibb.h>
#include <box/gpb_ioctl.h>
#include <signal.h>
#include <sys/file.h>

#define DEV_NAM 20

char *myname;

main(ac,av)
int ac;
char *av[];
{
                short int   errno, i;

                int gpfd, (*signal())();

                char device[DEV_NAM], tmpstrng[DEV_NAM];
                unsigned char stb;
                char *dev, *tmps;

                dev = &device[0];
                tmps = &tmpstrng[0];
                myname = av[0];

                switch(ac) {
                case 1:
                                fprintf(stderr,"At lease one argument
        required with %s command\n",myname);
                                fprintf(stderr,"Usage: %s driver \n",myname);
```

```
                exit(1);
                break;
default:        /* one or more arguments passed, process normally */
                break;
}

/* The above checks to see that at least one argument is passed */

/* now to find the device to be accessed                        */
while (ac— > 1)                 {

                (void) strcpy(dev,*++av);

                /* now attempt to open the device to be accessed */

                if ((gpfd = open(dev,O_RDONLY,0)) == -1) {

                /* Couldn't open it that way, try adding /dev
                 * in front of the name & try again             */
                                (void) strcpy (tmps,"/dev/");
                                (void) strcat (tmps,device);

                /* now attempt to open the device to be accessed */

                                if ((gpfd = open(tmps,O_RDONLY,0)) == -1) {
                /* Failed to open device, exit program)         */
                                fprintf(stderr,"%s: cannot access
                                gpib device driver '%s'\n",myname,tmps);
                                                exit(4);
                                }
                }

                if (ioctl(gpfd,GIOCSPOLL,&stb) == -1) {
                   fprintf(stderr,"%s: gpib(4) GIOCSPOLL
                     routine failed\n",myname);
                   fprintf(stderr,"Error returned from ioctl
                     is %d\n",errno);
                                exit(5);
                }
                /* Print out received status byte               */
                printf("%s: received %d, 0x%x, '%c' ",X
                                /open inmyname, stb, stb, stb);
                for (i=0; i <=7 ; i++)                          {
                                printf("%c",(stb << i) & 128 ? '1' : '0');
                                if (i == 3) printf(" ");
}
                printf(" STB\n");
```

```
                    }
                    exit(0);
}
```

Example Number Three

This third GPIB programming example shows how a program was written in C to read from the bus.

```
/* This is a program to test reading small buffers from the GPIB so
                that no data is lost between reads. The routine should exit
                when the EOM is received. The buffer size is specified in
                the call to this routine: foo [size] file. If size is not
                specified it will default to 256.                          */

#include <stdio.h>
#include <sys/file.h>
#include <ctype.h>

#define BUFSIZE 7500        /* Maximum size allowed for data xfers    */
#define NAMLEN  255
#define LOCK    1
#define UNLOCK  0
#define STRLEN  80
#define SIZE    500         /* Default size if no size value given    */
#define TRUE    1
#define FALSE   0
#define STDOUT  1

/*  This program intended for use with a GPIB Instrument Driver.
 *  not with the Interface driver.  If used with the Interface driver
 *  insure that all the devices and the interface have already been
 *  properly configured as talkers and listeners.
 */

static int          print_flg, count_flg;

main(argc,argv)
int                 argc;
char                *argv[];
{
                    int         fd, siz=SIZE, charin, doit();
                    char        name[NAMLEN], *c, myname[NAMLEN];

                    print_flg = FALSE;
                    count_flg = FALSE;

                    /* Load the program name into a variable for error messages*/
                    (void) strcpy (myname,*argv);

                    /* This program is set up with a few options that turn out to be
```

```
 * useful in working with various devices and shell scripts.  Any
 * number passed will be used as the MAXIMUM buffer size of data
 * to be received.  This allows the user to read x bytes of data
 * and then to read more data from the same device at a later time
 * without losing any data (e.g. reading a binary block count prior
 * to receiving the actual data).
 *
 * The options allowed are as follows:
 * (d)          Decode all non printing ASCII characters into
 *              a form such as '<13>' for a <CR>.  Prints the
 *              decimal equivalent value.
 *
 * (c)          Print out a count of the actual number of bytes
 *              received over the bus.  This data is sent over
 *              the STDERR channel so it does NOT mix with the
 *              actual data received.  This can be used to verify
 *              large block reception.
 */

/* Scan each argument for - option                              */
while ((—argc) > 0 && (*++argv)[0] == '-')
                /* Now scan the argument char by char           */
                for (c = argv[0]+1; *c != ' '; c++)
                switch (*c)
                {
                case 'd': print_flg = TRUE;                     /* Decimal decode*/
                                break;
                case 'c': count_flg = TRUE;                     /* counter flag*/
                                break;
                /* Numeric size decoding.  Allows the number to
                 * be included anywhere in an argument preceeded
                 * with a (-) and be accepted as a size value
                 */
                case '0': case '1': case '2': case '3': case '4':
                case '5': case '6': case '7': case '8': case '9':
                                siz = atoi(c++);
                                 while(*c >= 0x30 && *c <= 0x39) c++;
                                 *c—;
                                break;
                default: /* unknown, exit with error */
                                fprintf(stderr,"Usage: %s [-c] [-d] [-N]
                                device where N is a number.\n",myname);
                                exit(1);
                                break;
                }
                /* No arguments passed.  Must at least pass the device driver
                 * name to get the program to work.
```

```
 */
if (argc != 1)
{
fprintf(stderr,"Usage: %s [-c] [-d] [-N] device where N is a
                number.\n",myname);
exit(1);
}
else
{

/* Here is where the Device Driver is opened for data
 * transfers. This particular construction allows the
 * user to specify either the short name 'dmm', or
 * the full path '/dev/dmm'. If it cannot open the
 * specified driver it prepends a '/dev/' to the
 * passed driver name and trys again. If it fails
 * then the program will exit.
 * ##### NOTE You should not have a file of the same
 *   name as a device driver in the working directory
 *   or this program will fail when it attempts to
 *   open the file rather than the device driver
 */
if ((fd = open(*argv,O_RDONLY,0)) == -1)
 {
  /* It failed, add the '/dev/' to the driver
   * name passed
   */
  (void) strcpy(name,"/dev/");
  (void) strcpy(name+5,*argv);
  if ((fd = open(name,O_RDONLY,0)) == -1)
   {
                fprintf(stderr,"%s: cannot open input
                file %s\n",argv[0],argv[1]);
                exit(1);
   }
 }
}

/* This is where the actual transfer occurs. The function doit
 * returns the number of bytes received.
 */
charin = doit(fd,siz);

/* Close the file descriptor before exiting the program     */
(void) close(fd);

/* Now check the count flag. If set print out the count value*/
```

```
                if (count_flg == TRUE)
                {
                 if (charin > 0)
                        printf("      Number of bytes received was %d.\n",charin);
                 else
                        printf("No characters received\n");
                }
                exit(0);
}

/* This is the section of the program where the data transfer actually
 * takes place.  The two parameters passed to this routine are the file
 * descriptor of the Device Driver to be used, and the size parameter
 * which tells the system how many bytes to transfer (maximum).
 */
int                     /* this function is of type int                */
doit(fd,siz)
int             fd, siz;
{
                char    buf[BUFSIZE], s[STRLEN];
                int     charin, tmp;

                /* Here we attempt to read size bytes into buffer 'buf' from 'fd'*/
                charin = read(fd,buf,siz);
                if (charin == -1)       /* Error in read, printout error  */
                {
                        (void) strcpy(s,"gprd read");
                        perror(s);
                }

                /* How do we ship the data to STDOUT.  Is it straight or decoded
                 * decimal equivalents for nonprinting ASCII values
                 */
                if (print_flg == TRUE)  /* was print parameter passed? */
                {
                        for (tmp=0; tmp < charin; tmp++)
                        {
                         if (buf[tmp] > 31 && buf[tmp] < 129)    /* range test*/
                                printf("%c",buf[tmp]);
                         else
                                printf("<%d>",buf[tmp]);
                        }
                }
                else            /* No decoding due, ship it to STDOUT with write function*/
                        tmp = write(STDOUT,buf,charin);         /* send to std out */
                                                                /*with no change*/
```

```
    /* Check to see that write completed successfully.*/
                    if (tmp == -1) perror("gprd write"); /* Check to see that write completed*/

                    /* return the number of bytes received in the read statement above*/
                    return(charin);
}
```

Example Number Four

This GPIB programming example shows you how to incorporate service requests within a program.

```
/*************************************************************************
 *
 *  The purpose of this program is to show how to incorporate service
 *  request handling into a program on an interrupt basis.  The program
 *  only supports one instrument driver.
 *
 *  NOTE:  This is NOT A COMPLETE PROGRAM.  Much of the I/O and argument
 *  decoding has been removed.  This program segment does show how to use
 *  both the accessing of the Device Driver Configuration Table structure
 *  and how to enable SRQ to interrupt your program.  The various parts
 *  and pieces need to be added to the appropriate places in your own
 *  program and modified to fit your needs.
 *
 *************************************************************************/

#include <stdio.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <box/gpibb.h>
#include <box/gpb_ioctl.h>
#include <signal.h>
#include <sys/file.h>

#define         TRUE        1
#define FALSE   0
#define BUFSIZE 128
#define BUFIN   65000
#define ASCII   0
#define PCN     1
#define LOCK    1
#define UNLOCK  0


struct          gpibconf confptr;
int             mask, omask, gpfd;
char            autopoll = FALSE, print_mode_flag = FALSE;

main(argv,argc)
char *argv[];
int  argc;
{
```

```
/* In here is a bunch of declaration and initialization stuff
 .
 .
 .
*/

int             process_grp, tmp, one, running;
int             inthndl(), endprg();

mask = 1 << (SIGURG -1);
one = 1;
running = TRUE;
                .
/* get the current process group number for future use in
 * the program for interrupt handling of SRQ's
 */
process_grp = getpgrp(0);

/* Open the GPIB Device Driver as gpfd (gpib File Descriptor)
 * an argument to the program call
 */
gpfd = gpopen(*++argv);

/* Now to get the current configuration of the device.
 * Read the configuration data into the defined structure
 */
if ((tmp = ioctl(gpfd,GIOCGCONF,&confptr)) == -1)
{
                /* If unable to read the device data must be a problem.
                 * Alert the user and exit the program. (Saftey feature)
                 */
  fprintf(stderr,"Unable to read configuration data from %s\n",*argv);
                closeit(2);
}
/* Test for autopoll set. If it is NOT set, be sure to set it off
 * when the program exits.
 * NOTE: ##### It is recommended that AUTOPOLL be set to off
 * when no programs are running to prevent the possiblility of a
 * device generating a SRQ and having the workstaion perform a serial
 * poll with no program running. When this happens the signal is
 * lost and no interrupt driven program will be able to respond to
 * any future SRQ's. When this happens, the device driver must be
 * explicity polled to clear out the pending SRQ. This should be
 * visible with the 'gpstat' command.
 */
if ((confptr.gc_flags & GF_POLLME) != GF_POLLME )
{
```

```
                /* Enable autopoll                                  */
                confptr.gc_flags |= GF_POLLME;          /* set pollme bit*/
                /* Set the autopoll flag to proper state            */
                autopoll = FALSE;
                /* Write the data structure back out to the device driver*/
                (void) ioctl(gpfd,GIOCSCONF,&confptr);
} else                          /* Set the autopoll flag to proper state*/
                autopoll = TRUE;

/* enable the process interrupt to interrupt the program
 * with the set-process-group ioctl                                 */
if (ioctl(gpfd,TIOCSPGRP,&process_grp) == -1)
{
                fprintf(stderr,"%s unable to set TIOCSPGRP\n");
                closeit(3);
}

/*              Now set device to ASYNC for notification when
                the device status changes.  SRQ interrupts will
                not work unless this is done.                    */
if (ioctl(gpfd,FIOASYNC,&one) == -1)
{
                fprintf(stderr,"%s unable to set FIOASYNC for interface\n");
                closeit(4);
}

/* Now set up the interrupt handler and the exiting program
   routines                                                         */
(void) signal(SIGURG,inthndl)  /* Urgent signal (SRQ)   */
(void) signal(SIGINT,endprg);/* Program interrupt CTRL C*/

/* Main loop of program that will be interrupted by the above
   enabled interrupts                                               */
while (running)
{
                /* Get command from stdin
                 * This is where you do the bulk of your program.
                 * When interrupts occur for SRQ, then the program
                 * will branch to the 'inthndl() function following
                    .
                    .
                    .
                    .
                */


}
```

```
}

/* This is the main interrupt handler for SRQ's. If all the above steps
 * have been performed properly, then when SRQ becomes asserted this
 * routine will poll the specified device 'gpfd'. If more than one device
 * needs to be polled, then pass a structure/array of gpdf's and loop
 * through the structure/array until either all have been polled, or single
 * device has been found that was asserting SRQ (indicated by bit 6 being
 * set, the RQS bit of the Status ByTe).
 */
inthndl()
{
        extern      int         gpfd;
        int         i;
        unsigned char status;

        printf("SRQ seen\n"); /* Message to alert user SRQ happened*/

        /* Poll the Device Driver (instrument) and report the STB*/

        if (ioctl(gpfd,GIOCSPOLL,&status) == -1)
        {
                fprintf(stderr,"Unable to perform serial poll\n");
                exit(5);
        }
        /* Report the status byte in usual forms                    */
        printf("Status byte = %d decimal, %x hex, ",status, status);

        /* Following reports status byte in binary form             */
        for (i = 0; i <= 7; i++)
        {
                printf("%c", (status << i) & 128 ? '1' : '0');
                if (i == 3) printf(" ");
        }
        printf(" binary\n");
        (void) fflush(stdout);
}

/* This is an interrupt routine called upon exit from the program to
 * insure that all opened files and flags have been returned to their
 * proper conditions before the program actually ends
 */
endprg()
{
        /* Tell the user that the program was aborted               */
        fprintf(stderr,"Program aborting\n");
        printf("Sigint seen\n");
```

```
                /* The following routine call resets the Device Driver parameters to
                 * those found upon entry into the program
                 */
                closeit(1);
}

/* Routine to get the Device Driver configuration and reset the autopoll
 * feature only if it was changed by this program. If more than the
 * autopoll is modified then insure that all relevant parameters are
 * returned to their proper settings in this routine
 */
closeit(token)
int             token;
{
                /* clear the autopoll if autopoll == FALSE                */
                if (autopoll == FALSE)
                {
                        /* First get the current settings                 */
                        (void) ioctl(gpfd,GIOCGCONF,&confptr);

                        /* Then reset the parameters of interest  */
                        confptr.gc_flags &= ~GF_POLLME; /* diable pollme bit*/

                        /* Then write the changed values back out to the device
                         * driver.
                         */
                        (void) ioctl(gpfd,GIOCSCONF,&confptr);
                }
                /* Close out the GPIB file descriptor                     */
                (void) close(gpfd);
                exit(token);
}

/* Generic type of open routine for a Device Driver. Allows passing
 * just the name of the driver and not necessarily the whole path name.
 *
 * NOTE: #### This routine requries that there not be a file/directory
 * of the same name as the device driver attempting to be opened. In
 * case of doubt, the whole path name should be safe in all cases.
 */
gpopen(name)
char            name[];
{
                int             gpfd;

                char            s[BUFSIZE];
```

```
                    /*  Now attempt to open device to get the file descriptor  */
                    if ((gpfd = open(name, O_RDWR, 0)) == -1)
                    {
                    /* can't open, add /dev to name and try again              */
                                (void) strcpy (s,"/dev/");
                                (void) strcat (s,name);
                                if ((gpfd = open(s,O_RDWR,0)) == -1)
                                {
                                /* Can't open this one either, must not be configured */
            fprintf(stderr,"Unable to open device %s\n", s);
                                  exit(3);
                                }
                    }
                    return(gpfd);
}

/*  A routine to print out the non-printing ASCII characters as a decimal
 *  value surrounded with "<" and ">" (e.g. <13><10>).
 */
print(buff, len)
char            buff[];
int             len;
{
                int             i;

                switch (print_mode_flag)      /* global-external variable  */
                {
                case 0:         /*Print to STDOUT as received, no decoding*/
                                /*  The following is used to disable SRQ interrupts*/
                                omask = sigblock(mask);

                                (void) write(1,"Received <",10);
                                (void) write(1,buff,len);/* write out
                                                * complete buffer          */
                                (void) write(1,">\n",2);

                                /*  The following is used to enable SRQ interrupts*/
                                mask = sigsetmask(omask);

                                break;
                case 1:         /*Perform the decoding for non-printing ASCII*/
                                /*  The following is used to disable SRQ interrupts*/
                                omask = sigblock(mask);

                                for (i = 0; i < len; i++)
                                {
                                        if (buff[i] > 31 && buff[i] < 128)
```

```
                                                    printf("%c",buff[i]);
                                        else printf("<%d>",buff[i] & 255);
                                }
                                printf("\n");

                                /* The following is used to enable SRQ interrupts*/
                                mask = sigsetmask(omask);

                                break;
                default: /* Not a valid selection */
                                fprintf(stderr,"Not a valid selection for the print flag option\n");
                                break;
                }
}
```

9

BASIC GPIB Support

The facilities of the UTek operating system, in conjunction with your application program, control the GPIB and the instruments connected to it. The Tektronix ANSI BASIC programming language includes a special *GPIB extension*, which makes optimal use of the operating system's GPIB facilities.

BASIC supports three approaches to GPIB programming:

- High-level Approach

  The high-level approach makes it easy for you to write a GPIB application program. You don't need to know the actual GPIB commands and protocol, as set forth in the IEEE standards. This type of application program communicates with devices on the bus through individual GPIB *instrument drivers*.

  Even if you are familiar with GPIB commands and protocol, you will find that the instrument approach is the right one for the job. Whenever your GPIB application requires only one device, or a limited number of physical devices with minimal interaction among them, programming by the high-level approach almost surely is advantageous. The application takes less time to code and debug if the program perceives the GPIB as a set of logically independent entities.

  On the other hand, the high-level approach limits the number of active GPIB instruments to fifteen. Also, this method does not accommodate multiple instrument addressing or universal GPIB commands, such as IFC (Interface Clear).

- Low-level Approach

  The low-level approach requires an intimate knowledge of GPIB commands and protocol. The program accesses devices on the bus through an *interface driver*. Although more difficult to program, this method offers the possibility of complex interactions among instruments on the interface. When you need this level of sophisticated control over the bus, it is available for you.

  The number of instruments interacting over the bus is limited only by the GPIB itself. Multiple instrument addressing is possible with the low-level approach.

- Shared I/O Approach

  The shared I/O approach to GPIB programming lets programs using high-level and low-level approaches share their I/O statements, resulting in a flexible interface. This concept lets Tektronix ANSI BASIC programs combine user formatting (similar to **PRINT USING** with **IMAGE**), Tektronix standard GPIB formatting, and raw binary data to create a single GPIB message block. Output is buffered for you until you decide to transmit.

  Shared I/O also lets the program receive a GPIB transmission and accept the data according to Tektronix standard GPIB format, raw binary data, or as an undetermined number of GPIB message units to be interpreted by the program.

The GPIB extension to Tektronix ANSI BASIC consists of subroutines, conditions, and functions. The next subsections summarize the GPIB subroutines. Each describes the subroutines associated with one of the three approaches to GPIB programming in BASIC.

Some features provided in Tektronix ANSI BASIC are treated separately. These include predefined conditions and condition handling, GPIB program functions, and asynchronous I/O. The special features are discussed under Programming Considerations, later in this appendix.

SUBROUTINES: INSTRUMENT DRIVER

The form of your **G_OPEN** call determines the type of driver referenced. To designate an instrument driver, the name supplied must start with **/dev**. But no matter which approach to GPIB programming you choose, BASIC communicates to I/O devices on the bus through a *logical unit number*, or *LU*. The LU becomes associated with a particular GPIB driver by means of **G_OPEN**.

Once open, any LU can be assigned to your program using **G_ASSIGN**. Some GPIB subroutines communicate with the assigned LU only. Others can communicate with any opened driver, assigned or not.

Table 9-1 summarizes the subroutines supporting the high-level GPIB approach through individual instrument drivers.

Table 9-1
GPIB SUBPROGRAMS: INSTRUMENT

Name	Summary
G_ASSIGN	Assign driver to current program
G_CLOSE	Close a logical unit for I/O
*G_CMD**	Send commands to an LU
G_FETCHBUFF$	Fetch current copy of LU's buffer
G_GTL	Send Go To Local (GTL)
G_INPUT	Get formatted data from an LU
G_INPUTB	Get unformatted data from an LU
G_INPUTS	Get string array data from an LU
G_IOERROR	Test for asynchronous I/O error
G_OPEN	Open the driver as a logical unit
G_POLL	Perform serial poll based on LU
G_PREREAD	Read message block asynchronously
G_PRINT	Put a formatted message to an LU
G_PRINTB	Put unformatted message to an LU
G_PRINTF	Output user formatted message to LU
G_RELEASE	Release assigned driver
G_RESET	Change driver configuration
G_SDC	Send Selective Device Clear (SDC)

* **G_CMD** is capable of transmitting both commands and data. However, with an instrument driver this routine should be used to send addressed commands only.

Notice that instrument drivers can send or receive formatted or unformatted data. Several routines in this group provide the ability to send "low-level" (interface driver) type commands even though dealing with "high-level" (instrument driver) GPIB communications. These subprograms are:

G_CMD Send addressed commands only: not data

G_GTL Send the *Go To Local* GPIB command (GTL)

G_SDC Send the *Selective Device Clear* command (SDC)

The **G_POLL** routine only applies to instrument drivers. Invoking this subprogram performs a serial poll on the specified instruments to determine which one is requesting service (asserting SRQ).

G_POLL returns a status byte to your program for interpretation and possible action. This action deasserts SRQ (service request) on the specified logical unit. Polling is discussed toward the end of this section.

SUBROUTINES: INTERFACE DRIVER

Many of the BASIC subroutines that support high-level GPIB communications can also be used for low-level communication with a GPIB interface driver. The actual operation of these routines differ depending on the type of driver (instrument or interface) referenced.

Table 9-2 summarizes the Tektronix ANSI BASIC subprograms associated with the low-level approach to GPIB programming.

Notice that many of the interface driver routines send low-level GPIB commands over the assigned logical unit, giving you direct control of bus traffic.

The subprogram **G_SETDRI** lets you set up the configuration of a GPIB interface driver for a workstation according to your specifications. It works in conjunction with the system **gpconf** command, described in the first part of this section. Together they serve to define communication characteristics for each GPIB port.

The parameters you set with **G_SETDRI** and **gpconf** affect the way in which you communicate with the interface itself and, therefore, to any I/O devices connected to the GPIB interface. The subroutine **G_RESET** lets you modify the interface driver's configuration without having to close if first.

Some of the interface parameter settings can be overridden by using **G_OPEN** to define an individual instrument driver as a separate logical unit.

Table 9-2
GPIB SUBPROGRAMS: INTERFACE

Name	Summary
G_APOLL	Perform serial poll based on address
G_ASSIGN	Assign driver to current program
G_CLOSE	Close a logical unit for I/O
G_CMD	Send commands or data
G_DCL	Clear all instruments
G_FETCHBUFF$	Fetch current copy of LU's buffer
G_GET	Send Group Execute Trigger (GET)
G_GTL	Send Go To Local (GTL)
G_IFC	Send Interface Clear (IFC)
G_INPUT	Get formatted data from an LU
G_INPUTB	Get unformatted data from an LU
G_INPUTS	Get string array data from an LU
G_IOERROR	Test for asynchronous I/O error
G_LISTEN	Listen-address bus instruments
G_LLO	Send Local Lock Out (LLO)
G_OPEN	Open the driver as a logical unit
G_PPD	Parallel Poll Disable (PPD)
G_PPE	Parallel Poll Enable (PPE)
G_PPOLL	Perform parallel poll
G_PPU	Parallel Poll Unconfigure (PPU)
G_PREREAD	Read message block asynchronously
G_PRINT	Put a formatted message to an LU
G_PRINTB	Put unformatted message to an LU
G_PRINTF	Output formatted message to an LU
G_RELEASE	Release assigned driver
G_RENOFF	Release Remote Enable (REN) line
G_RENON	Assert Remote Enable (REN) line
G_RESET	Change interface driver configuration
G_SDC	Send Selective Device Clear (SDC)
G_SETDRI	Set interface driver configuration
G_TALK	Talk-address bus instruments
G_UNL	Send Unlisten (UNL) message
G_UNT	Send Untalk (UNT) message
G_WEND	Start asynchronous communications

SUBROUTINES: SHARED I/O SUPPORT

Many of the low-level subprograms could also be used for communicating with instruments as well. The routines that can communicate with either instruments or interfaces make up the BASIC shared I/O support facilities. The routines in this group are those presented in both Table 9-1 and Table 9-2.

Table 9-3 summarizes the shared I/O support subprograms and the functions they perform. Remember that these routines may behave differently, depending on whether they are applied to interfaces or to instruments.

Table 9-3
GPIB SUBPROGRAMS: SHARED I/O

Name	Summary
G_ASSIGN	Assign driver to current program
G_CLOSE	Close a logical unit for I/O
*G_CMD**	Send commands or data
G_FETCHBUFF$	Fetch current copy of LU's buffer
G_GTL	Send Go To Local (GTL)
G_INPUT	Get formatted data from an LU
G_INPUTB	Get unformatted data from an LU
G_INPUTS	Get string array data from an LU
G_IOERROR	Test for asynchronous I/O error
G_OPEN	Open the driver as a logical unit
G_PREREAD	Read message block asynchronously
G_PRINT	Put a formatted message to an LU
G_PRINTB	Put unformatted message to an LU
G_PRINTF	Put user formatted message to an LU
G_RELEASE	Release assigned driver
G_RESET	Change driver configuration
G_SDC	Send Selective Device Clear (SDC)

* When applied to an instrument, addressed commands only may be sent.

Notice that the shared I/O routines include those that can open and close a GPIB driver (**G_OPEN** and **G_CLOSE**) and those that assign and release the program's current GPIB driver (**G_ASSIGN** and **G_RELEASE**).

Both formatted and unformatted data transfers are also supported in both the low- and high-level approaches to GPIB programming in BASIC. With the shared I/O approach, your application program creates a *message*, or *message block*, from individual *message units*. The message units themselves may be constructed by using any mixture of the GPIB print routines shown in the table.

G_PRINTB and **G_INPUTB** handle unformatted binary data, allowing you direct control of data transfers. **G_PRINT** and **G_INPUT** treat data according to the Tektronix Standard Codes and Formats described in Appendix B. **G_PRINTF** works something like the **PRINT USING** statement in conjunction with an **IMAGE**, providing user formatting features for GPIB. **G_INPUTS** reads data consisting of a variable number of message units into a string array for program analysis.

Another shared routine of interest, **G_FETCHBUFF$**, returns the current contents of the data buffer data for the specified logical unit. The buffer contains unformatted binary data, just as it was when sent or received. This routine is commonly used by programmers when debugging a GPIB control program.

PROGRAMMING CONSIDERATIONS

When programming the GPIB in Tektronix ANSI BASIC, you can make use of the powerful data storage and manipulation facilities offered. These include extended array processing, string arrays, double-precision and floating point arithmetic, extended I/O, and built-in graphics capabilities.

One feature to keep in mind is BASIC's exception handling facility. With it, you can *trap* errors that may arise during program execution, analyze the error and take appropriate action. Exception handling is further described later in this section.

For details on any of the BASIC subprograms, conditions, or functions in the GPIB extension, refer to the *Tektronix ANSI BASIC Keyword Dictionary*. Introductory information on the BASIC language and the Tektronix implementation of it can be found in *Tektronix ANSI BASIC Users Guide.*

Exception Handling

A program exception is an error that occurs during execution causing your program to be interrupted. Some examples of program exceptions are arithmetic overflow or underflow, and addressing errors. In order to handle program exceptions, BASIC includes the **WHEN EXCEPTION IN**
USE ... END WHEN statements. With these, you can route errors to *exception handlers.*

Exception handlers can be either *global* or *local.* A local exception handler is coded within your program whereas a global exception handler is an independent program executed whenever an error occurs. A global exception handler is defined between **HANDLER** and **END HANDLER** statements.

Within the handler you can use the following statements:

RETRY	Re-execute the program line that caused the exception.
CONTINUE	Return to the program at the line following the line in which the exception occurred.
EXIT HANDLER	Pass the exception to the next higher **WHEN** block, if one exists. Otherwise, the exception is handled by the normal system exception handler.
STOP	Causes the entire program to cease execution. This is equivalent to an abort job request.

These exception handling facilities can greatly aid in program debugging. You can also use them when running programs that need to take care of errors that would otherwise cause the program to abort. The program can then perform necessary clean-up operations itself, including saving status and data if necessary, in case other attempts to recover should fail.

Condition Handling

A *condition*, also called a *software interrupt*, is a software event that may arise at any time during the execution of a program. Programs can respond to conditions whenever they occur, no matter what they are doing at the time. Conditions, in BASIC, are usually defined by CONDITION statements.

Several conditions that can occur in GPIB programs have been defined in Tektronix ANSI BASIC. These conditions pass control to *condition handlers*. These handlers, like the exception handlers previously discussed, can be either local or global.

A *local handler* is code within your program designed to handle the interrupt condition, while a global condition handler is a callable subprogram. When both a global and a local handler are associated with the same condition the local handler takes precedence.

Table 9-4 lists Tektronix ANSI BASIC predefined interrupt conditions and their meanings.

Table 9-4
BASIC GPIB CONDITIONS

Name	Meaning	Driver Affected
*GPIB_DCL**	DCL or SDC line asserted	Interface only
GPIB_DONE	Async transmission done	Interface or Instrument
*GPIB_IFC**	IFC line asserted	Interface only
*GPIB_MLA**	MLA line asserted	Interface only
*GPIB_MTA**	MTA line asserted	Interface only
GPIB_SRQ	SRQ line asserted	Interface only
GPIB_TCT	TCT line asserted	Interface only

* Function not implemented in initial release.

GPIB predefined conditions, like any conditions, are manipulated by two sets of BASIC statements:

- **SET** and **ASK**
- **ON** and **OFF**

The first pair lets you assign or inquire the **PRIORITY, PENDING,** or **ENABLED** status for the specified logical unit. The second pair associates or disassociates a condition handler with one of the GPIB predefined vector conditions. For details, refer to *Tektronix ANSI BASIC Keyword Dictionary* for details on these commands and how to use them.

Notice that the **GPIB_DONE** condition is the only one which is valid for both instruments and interfaces. This condition is associated with the asynchronous I/O feature, described next.

Asynchronous Data Transfers

Transmission or reception of large quantities of data over the GPIB can be time-consuming. The speed of execution of a GPIB program, when transfers are *synchronous*, depends upon the the speed of the slowest I/O device connected to the GPIB port. Therefore, a data transfer method has been devised for Tektronix ANSI BASIC called *asynchronous I/O*.

Asynchronous I/O lets a program initiate a data transfer with a device on the bus and then continue processing. For example, your program could perform an input operation and then initiate another input operation while processing data from the first read. The only requirement is that the input operations specify different logical units. The devices themselves may be connected to the same GPIB port.

The ability to perform asynchronous data transfers is a characteristic of a GPIB interface, and it affects all devices on the interface. Each GPIB interface is enabled for asynchronous I/O by setting its *sync* parameter with **G_SETDRI**.

Note that individual instrument drivers can also be declared asynchronous as well. But since all the instruments addressed as separate logical units may use the same bus, setting *sync* is preferable for program consistency. Collisions with transmissions from what may logically appear to be unrelated instruments is usually resolved by the system, but two asynchronous transmissions on the same bus could interfere with the timing of a sequence of commands if not correctly programmed.

Table 9-5 summarizes the BASIC subroutines that can be used as part of the asynchronous I/O package.

Table 9-5
GPIB SUBPROGRAMS: ASYNCHRONOUS I/O

Name	Summary
G_INPUT	Get formatted data from an LU
G_INPUTB	Get unformatted data from an LU
G_INPUTS	Get string array data from an LU
G_IOERROR	Test for asynchronous I/O error
G_PREREAD	Read message block asynchronously
G_PRINT	Put formatted message to an LU
G_PRINTB	Put unformatted message to an LU
G_PRINTF	Put user formatted message to an LU
G_WEND	Start asynchronous communications

Asynchronous input is done by calling **G_PREREAD**, which returns immediately to the executing program while simultaneously receiving input from the specified logical unit. Invoke one of the routines **G_PRINT**, **G_PRINTF**, or **G_PRINTB** with the *send-message* parameter set to 1 (for send). After formatting the message block into the specified logical unit's buffer, these functions return to the executing program while transmitting the message block.

To transfer data asynchronously between two instruments on a GPIB interface independent of workstation control, first talk address the desired instrument. Then, listen address the appropriate instruments and invoke the **G_WEND** subprogram. **G_WEND** returns to the executing program while the devices communicate among themselves, without workstation intervention. When the I/O mode is synchronous, **G_WEND** does not return to the caller until the data transfer completes.

Of course, there must be a way to notify the program when a transfer involving asynchronous I/O has completed, so that the program can then process the data as necessary. This is accomplished by the **GPIB_DONE** condition, briefly discussed before. **GPIB_DONE** is raised for the appropriate logical unit when an asynchronous data transfer is complete.

The **G_IOERROR** subroutine causes any exceptions that occurred during the asynchronous I/O on the specified logical unit to be felt by the calling program. If **G_IOERROR** raises no exceptions, then the transfer completed normally. Exception handling is discussed earlier in this section.

Interface and Instrument Polling

Most GPIB-compatible I/O devices are constructed with the ability to send an interrupt over the bus. This interrupt, called *SRQ*, signals that the device requires servicing of some kind. Since a number of devices may be connected to a GPIB, the program needs some way of to distinguish which device or devices are asserting SRQ. This method is called *polling*. A GPIB controller can poll devices in *parallel* or in *serial* fashion.

The GPIB driver includes an advanced polling mechanism based on the GPIB service request (SRQ) function. This feature can be enabled by setting the *poll* parameter in the **gpconf** command. When enabled, and SRQ is asserted, a serial poll is performed by the system to determine which instrument requires attention. As soon as the requesting instrument's status byte is read, SRQ is deasserted.

If a GPIB program uses the individual instrument (high-level) approach, the program is notified when its own instrument has requested service. A program accessing the interface (low-level) is notified when SRQ is asserted by any device on the GPIB port not associated with a specific instrument driver. In general, system polling should be enabled when communicating to the GPIB through instrument drivers and disabled when communicating through an interface driver.

Tektronix ANSI BASIC provides serial polling facilities also. The routines **G_POLL** and **G_APOLL** can be applied to instruments and interfaces respectively. Using these routines makes the polling process easy: when you call one, it automatically executes a polling loop, checks each device for SRQ asserted, and returns that device's status byte to the program, thereby clearing SRQ.

Parallel polling is possible only through the low-level interface approach. The routines **G_PPE**, **G_PPD**, **G_PPU**, and **G_PPOLL** are used for parallel polling from a BASIC program. See the *Tektronix ANSI BASIC Keyword Dictionary* for details.

BASIC GPIB Functions

Finally, Tektronix ANSI BASIC provides several functions associated with GPIB processing and useful within GPIB programs. Each function returns a numerical value to the calling program. Table 9-6 summarizes the predefined GPIB functions and return values.

Table 9-6
BASIC GPIB FUNCTIONS

Name	Meaning	Value
G_LAG	Listen address group	32
G_MA	My address	0*
G_SAG	Secondary address group	96
G_TAG	Talk address group	64

* Returns the primary address of the workstation on the assigned GPIB interface. This value is set with **G_SETDRI**, default 0.

Expression formation is made easier by using these functions. For example:

Talk Address, instrument *n*: **G_TAG** + n

My Talk Address is: **G_TAG + G_MA**

GPIB Concepts

The IEEE 488 standard defines three aspects of an instrument's interface:

Mechanical	The connector and the cable.
Electrical	The electrical levels for logical signals and how the signals are sent and received.
Functional	The tasks that an instrument's interface can perform, such as sending data, receiving data, triggering the instrument, etc.

Using this interface standard, instruments can be designed to have a basic level of compatibility with other instruments that meet the standard.

MECHANICAL ELEMENTS

IEEE 488 specifies a standard connector and cable for linking instruments to ensure that GPIB instruments are pin-compatible. The GPIB connector has 24 pins, with 16 assigned to specific signals and 8 to shields and grounds.

Figure A-1 shows the GPIB connector and pin assignments.

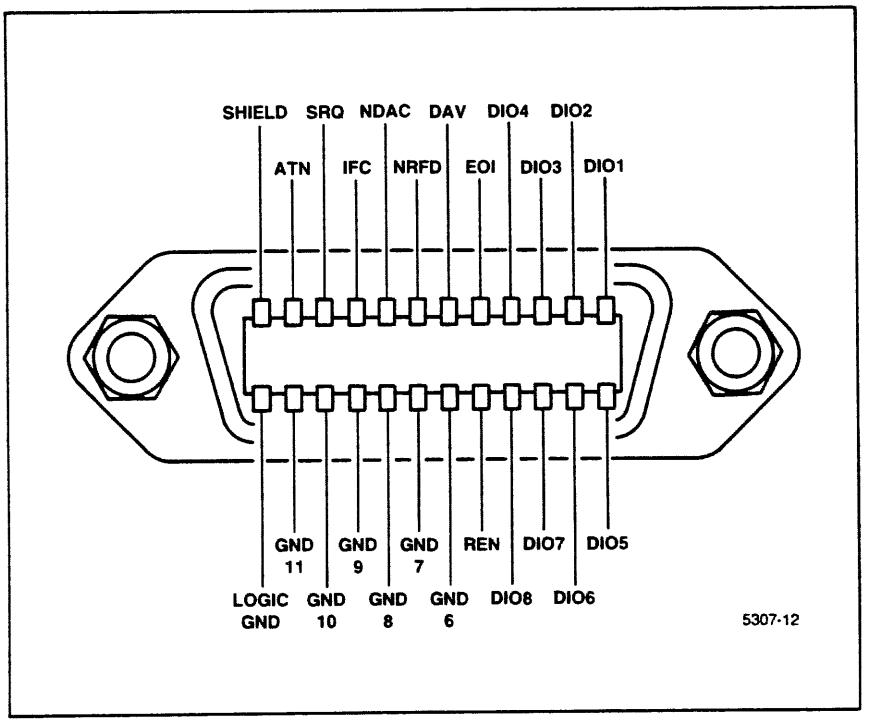

Figure A-1. GPIB Connector.

Allowable Configurations

Instruments can be connected to the GPIB in *linear* or *star* configurations, or in a combination of both (Figure A-2).

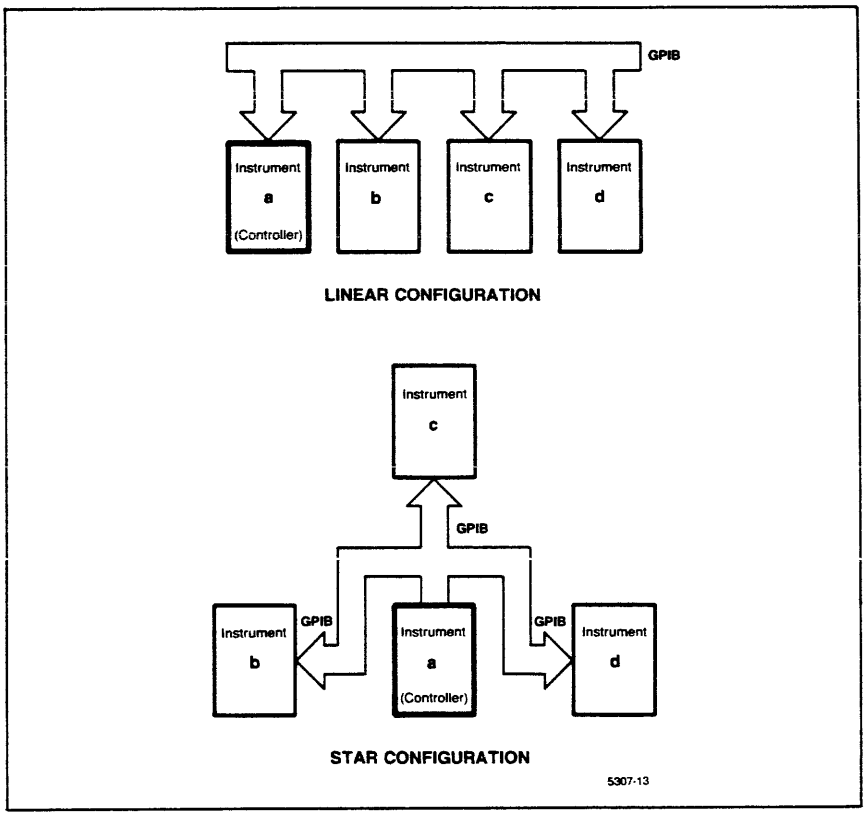

Figure A-2. Allowable GPIB Configurations.

Restrictions

Instruments connected to a single bus cannot be separated by more than two meters for each instrument on the bus. In addition, the total cable length of the bus cannot exceed 20 meters.

To maintain proper electrical characteristics on the bus, a device load must be connected for every two meters of cable length, and at least two-thirds of the instruments connected to the bus must be powered on. (For more information, see *IEEE Standard 488-1978.*)

Although instruments are usually spaced no more than two meters apart, they can be separated further if the required number of device loads is grouped at any point on the bus. More than 15 instruments can be interfaced to a single bus if they don't connect directly to the bus, but are interfaced through a primary device. Such a scheme can be used with programmable plug-ins attached to a central device, where the central device is addressed with a primary address code and the plug-ins are addressed with a secondary address code (see the topic *Instrument Addresses* in this section).

ELECTRICAL ELEMENTS

The IEEE 488-1978 standard defines the voltages and current values required at connector nodes. All specifications are based on the use of TTL technology. The logical states are defined as follows:

Logical State	Electrical Signal Levels
0	Corresponds to voltages ≥ 2.0 volts and ≤ 5.2 volts (high state)
1	Corresponds to voltages ≥ 0 volts and ≤ 0.8 volts (low state)

Messages can be sent as either active or passive true signals. Passive true and false signals occur in the high state and must be carried on a signal line using open collector devices.

FUNCTIONAL ELEMENTS

Interface functions provide the facilities through which instruments send, process, and receive messages. IEEE 488 defines ten different interface functions, described in the following paragraphs. The abbreviations for these functions, which are in parenthesis, are taken from the IEEE 488 standard and are commonly used when describing the functions.

Acceptor Handshake (AH)

The AH function provides an instrument with the capability to guarantee proper reception of data. The AH function delays initiation or termination of a data transfer until the instrument is ready to receive the next data byte.

Source Handshake (SH)

The SH function works together with the AH function on a listening device to guarantee proper transfer of messages. The SH function controls the initiation and termination of the transfer of data bytes.

Listener (L) and Listener Extended (LE)

The L and LE functions provide an instrument with the capability to receive device-dependent data over the interface. This capability exists only when the instrument is addressed to listen. The L function uses a 1-byte address; the LE function uses a 2-byte address. In all other aspects, the capabilities of both functions are the same.

Talker (T) and Talker Extended (TE)

The T and TE functions provide an instrument with the capability to send device-dependent data over the interface. This capability exists only when the instrument is addressed to talk. The T function uses a 1-byte address; the TE function uses a 2-byte address. In all other respects, the capabilities of both functions are the same.

Device Clear (DC)

The DC function provides an instrument with the capability to be cleared or initialized, either individually or as part of a group of instruments.

Device Trigger (DT)

The DT function provides an instrument with capability to have its basic operation started, either individually or as part of a group of instruments.

Remote/Local (RL)

The RL function provides an instrument with the capability to select between two sources of input. The function indicates to the instrument that either input information from its front panel switches (local) or corresponding information from the GPIB interface is to be used.

Service Request (SR)

The SR function provides an instrument with the capability to request service from the controller in charge of the interface.

Parallel Poll (PP)

The PP function provides an instrument with the capability to send a status message to the controller without being addressed to talk.

Controller (C)

The C function provides an instrument with the capability to send *device addresses, universal commands,* and *addressed commands* to other instruments over the interface. It may also provide the capability to determine which instruments require service.

INSTRUMENT ADDRESSES

Every instrument on the bus has one or more addresses. The types of addresses an instrument can have are: a *primary address*, *a listen address*, a *talk address*, and a *secondary address*.

Primary Address

Every instrument connected to the bus has a unique *primary address*. You can set the primary addresses of most of these instruments. On some instruments, this address is set by a series of binary switches located at the rear of the instrument. Figure A-3 shows an example of how to set the primary address of an instrument that uses binary switches.

Each of the address switches on the instrument represents a binary digit with a value of 1 if the switch is in the *on* position, or 0 if the switch is in the *off* position. Reading from right to left, the place value of each succeeding switch increases by a factor of 2; thus, the rightmost switch has a place value of 1, the second switch from the right has a place value of 2, the third a place value of 4, and so on.

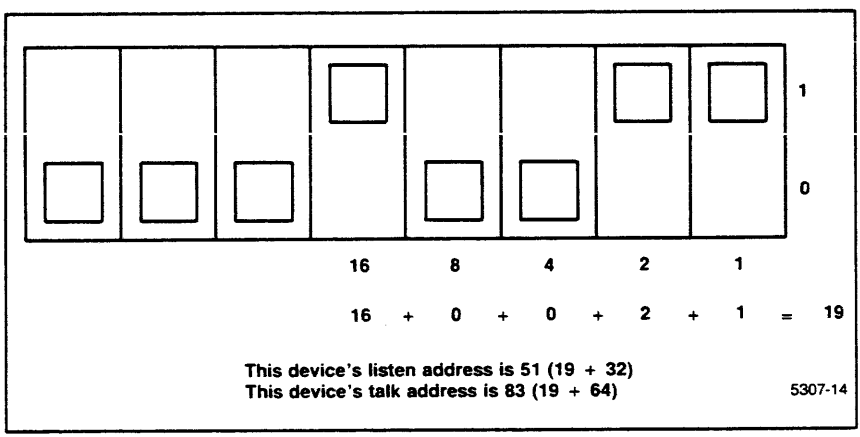

Figure A-3. Example Primary Address Setting.

To determine the value of the address represented by a given switch setting, simply add up the place values of all the switches in the *on* position. In Figure A-3, switches with place values of 16, 2, and 1 are in the *on* position. Since 16 + 2 + 1 = 19, an instrument with switches set in these positions would have a primary address of 19.

No two instruments on the GPIB can have the same primary address. Valid primary addresses range from 0 to 30.

Some instruments have their primary addresses preset by the manufacturer and can be changed only by qualified service personnel. If you are in doubt, check the user's manual of the instrument.

Listen Address

Every instrument that can receive device-dependent messages over the bus has a unique *listen address*. When an instrument senses its listen address on the bus with the ATN signal line asserted, the instrument prepares to receive data sent over the bus. When ATN is unasserted, the instrument receives data.

An instrument's *listen address* is determined by adding 32 to its *primary address*. Thus an instrument with primary address 19 has a listen address of 19 + 32 = 51.

Any number of instruments on the bus may be addressed to listen at the same time.

Talk Address

Every instrument that can send data over the bus has a unique *talk address*. When an instrument senses its talk address on the bus with ATN asserted, the instrument prepares to send data over the bus. When ATN is then unasserted, the instrument sends data.

Only one instrument on the bus may be addressed to talk at a time. When an instrument addressed to talk senses another instrument's talk address on the data lines with ATN asserted, the first instrument automatically *untalks* itself.

An instrument's talk address is determined by adding 64 to its primary address. Thus, an instrument with primary address 19 has a talk address of 19 + 64 = 83.

Secondary Address

Some instruments support a special addressing scheme called *secondary addressing*, which uses addresses in the range from 96 to 126. Not all IEEE 488 compatible instruments support secondary addressing.

There are several variations of secondary addressing implementations. When in doubt, check the user's manual for the instrument.

GPIB BUSES

The 16 GPIB signal lines can be divided into three *buses*: the *data bus*, the *management bus*, and the *handshake* bus (Figure A-4).

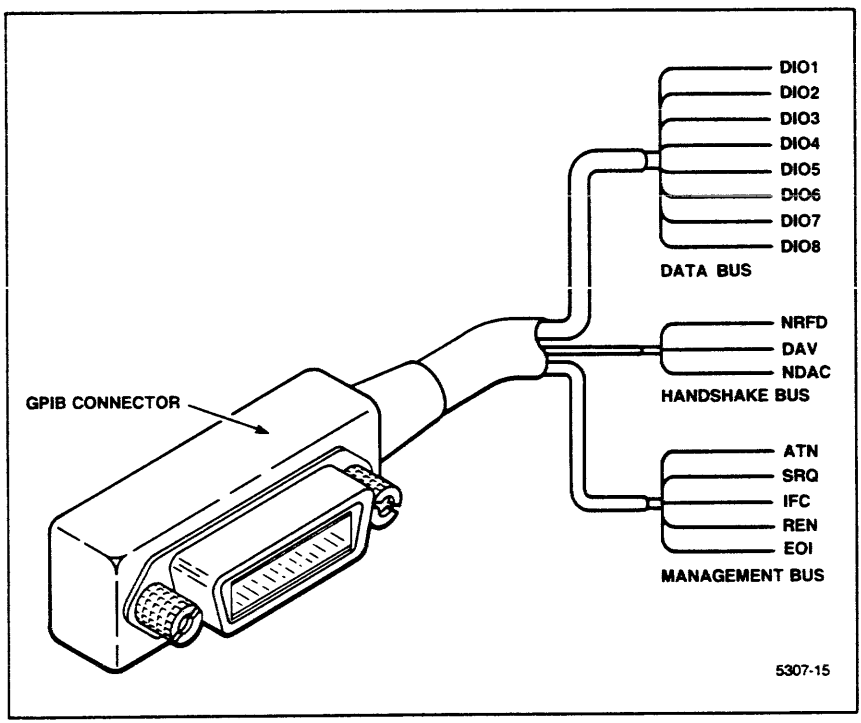

Figure A-4. Data, Management, and Handshake Buses.

The *data bus* is composed of eight signal lines that carry data to be transferred on the GPIB. The *management bus* is composed of five signal lines that control data transfers. The *handshake bus* is composed of three signal lines that synchronize data transfers between instruments.

Data Bus

The data bus contains eight bidirectional signal lines, named DI01 through DI08. One byte of information is transferred over the bus at a time. DI01 carries the least significant bit of the byte and DI08 carries the most significant bit.

Each byte of information transferred on the data bus represents either a command, a device address, or a device-dependent message. Data bytes can be formatted in ASCII code or in device-dependent binary code.

Management Bus

The management bus is a group of five signal lines (ATN, EOI, IFC, REN, and SRQ) used in managing data transfers on the data bus. The definitions for these lines are listed in the following paragraphs.

Attention (ATN)

The ATN management line is activated by the controller to send *universal commands* and *addressed commands*, and to designate instruments as talkers and listeners for an upcoming data transfer.

When ATN is asserted, messages sent on the data bus are interpreted as commands or addresses. When ATN is unasserted, messages sent on the data bus are interpreted as device-dependent messages. Only instruments that have been addressed by the controller to talk or listen can take part in a device-dependent data transfer.

End or Identify (EOI)

The EOI signal line can be used by any talker to indicate the end of a data transfer sequence. Talkers that use EOI activate the EOI line as the last byte of information is being transferred.

Interface Clear (IFC)

The IFC line is activated by the system controller to:

- Unaddress all talk-addressed and listen-addressed instruments on the bus.
- Take controller-in-charge status.
- Take all instruments out of serial poll mode (same as sending SPD, which is described under the topic *Addressed Commands* later in this section).

Only the system controller can activate this signal line.

Remote Enable (REN)

The REN signal line is activated by the system controller to give all instruments on the bus the capability of being placed under remote (program) control. Only the system controller can activate this line.

When the REN signal line is activated, instruments that receive their listen addresses on the data bus accept and execute commands from the controller-in-charge. When REN is deactivated, all instruments on the bus revert to front-panel control.

Service Request (SRQ)

The SRQ line can be activated by any instrument on the bus to request service from the controller. The controller responds (if programmed to do so) by serial polling all instruments on the bus in order to find the instrument requesting service. The SRQ line is deactivated when the instrument requesting service is polled.

Handshake Bus

Three lines (NRFD, DAV, and NDAC) comprise the handshake bus. These three lines control the sequence of operations each time a byte is transferred on the data bus. This sequence is *not* under user control, but information about the three transfer lines is presented here for completeness.

Not Ready For Data (NRFD)

An active NFRD line indicates that one or more of the listeners is not ready to receive the next data byte. When the NRFD line goes inactive (indicating all listeners are ready to receive data), the talker places the next data byte on the data bus and activates the DAV signal line.

Data Valid (DAV)

The DAV line is activated by the talker shortly after the talker places a valid data byte on the data bus. This tells each listener to capture the data byte currently on the bus.

Not Data Accepted (NDAC)

The NDAC line is held active by each listener until the listener captures the data byte on the data bus. When all listeners have captured the data byte, NDAC goes inactive. This tells the talker to take the byte off the data bus.

GPIB COMMUNICATION PROTOCOL

Each instrument on the bus at any given time may be a *controller*, a *talker*, or a *listener*. Some instruments have two or even three of these capabilities. For example, some instruments are talk-only, others are listen-only, others can talk and listen, others can talk, listen, and control.

Controllers are instruments that assign talk and listen status to other instruments on the bus. Since only one instrument can talk at a time, and since it is seldom desirable for every instrument to listen, a controller is needed to designate which instrument is to talk and which are to listen during any data transfer.

Talkers are instruments that can put data onto the eight data lines. Once on the data lines, data can be read by any active listeners. Only one instrument can talk at one time, to eliminate possible confusion.

Listeners are instruments that can read data from the data lines. Any number of instruments can be listening to the talker at one time. The rate at which the talker can put information onto the data lines is restricted to the rate at which the slowest active listener on the bus can accept the data.

Controllers, talkers, and listeners each have special properties and requirements that you must remember when writing programs to control instrument networks. The following subsections describe these properties and requirements.

Controllers

There are two kinds of controllers on the GPIB: the *system controller* and the *controller-in-charge*

A GPIB network can have at most one instrument acting as the system controller and one instrument acting as the controller-in-charge. The system controller and the controller-in-charge may be (and often are) the same instrument.

A GPIB configuration may include any number of instruments capable of acting as the system controller or controller-in-charge, subject only to the limitations on the total number of instruments allowed on the bus.

Once a GPIB network has been set up, system control cannot be passed from instrument to instrument; controller-in-charge status can. Once an instrument is the system controller, no other instrument on the bus can assume that role without your reconfiguring the GPIB network.

Only the system controller can affect the status of the Interface Clear (IFC) and Remote Enable (REN) management lines. Asserting the IFC line makes the system controller the controller-in-charge, and untalks, unlistens, and disables serial polling for all instruments.

Asserting REN enables the remote operation of instruments by the controller.

NOTE

Asserting REN does not automatically put all instruments on the bus into remote state, but simply allows the controller in charge to put them into this state.

Any instrument acting as controller-in-charge may pass control to any other instrument on the bus capable of assuming control.

Talkers

A *talker* is an instrument that has sensed its talk address on the data lines with ATN asserted. When a talker is addressed to talk, it sends data on the data lines when ATN is then unasserted.

Only one instrument on the bus may be addressed to talk at a time. When an instrument addressed to talk recognizes any other talk address on the data lines with ATN asserted, the instrument automatically *untalks* itself. (When an instrument is *untalked*, it does not place data on the data lines when ATN is asserted.)

An instrument may also untalk itself when it recognizes its listen address on the data lines with ATN asserted. In this case, the instrument becomes a listener when ATN is unasserted.

Listeners

A *listener* is an instrument that has sensed its listen address being sent on the data lines with ATN asserted. After an instrument is addressed to listen, it accepts information coming on the data lines when ATN is unasserted.

Any number of instruments on the bus may be addressed to listen at the same time. When an instrument previously addressed to listen senses its talk address being sent on the data lines with ATN asserted, the instrument may *unlisten* itself and become a talker when ATN is unasserted.

UNIVERSAL COMMANDS

Universal commands are commands that are obeyed by all instruments on the bus with the appropriate subsets of the IEEE 488 interface functions implemented. The controller sends universal commands by placing certain values on the data lines with ATN asserted.

The universal commands include: Device Clear (DCL), Local Lockout (LLO), Parallel Poll Unconfigure (PPU), Serial Poll Disable (SPD), and Serial Poll Enable (SPE).

Also included in this description are Unlisten (UNL), and Untalk (UNT). Although not commands in the strict sense, the values of UNL and UNT act like universal commands when sent on the data lines with ATN asserted.

Device Clear (DCL)

The DCL command *clears* (initializes) all instruments on the bus that have a DC1 or DC2 subset of the DC interface function.

To send the DCL command, the controller places the value 20 on the data lines with ATN asserted.

Local Lockout (LLO)

The LLO command *locks out* the front panels of all instruments on the bus that have an RL1 subset of the RL interface function. (Devices with RL0 or RL2 subsets of the RL interface function ignore LLO.) After receiving the LLO command and its listen address, an instrument ignores any subsequent inputs from front panel control switches with corresponding remote controls, and only obeys commands coming over the GPIB interface.

To send the LLO command, the controller places the value 17 on the data lines with ATN asserted.

Parallel Poll Unconfigure (PPU)

The PPU command unconfigures all instruments on the bus for parallel polling.

To send the PPU command, the controller places the value 21 on the data lines with ATN asserted.

Serial Poll Disable (SPD)

The SPD command returns all instruments on the bus from the *serial poll enabled* state.

To send the SPD command, the controller places the value 25 on the data lines with ATN asserted.

Serial Poll Enable (SPE)

The SPE command puts all instruments on the bus with an SR1 subset of the SR interface function into the *serial poll enabled* state. In this state, each instrument sends the controller its status byte, instead of the its normal output, after the instrument receives its talk address on the data lines with ATN asserted.

To send the SPE command, the controller places the value 24 on the data lines with ATN asserted.

Unlisten (UNL)

The UNL command takes all listen-addressed instruments on the bus out of the listen-addressed state.

To send the UNL command, the controller places the value 63 on the data lines with ATN asserted.

Untalk (UNT)

The UNT command takes any talk-addressed instrument on the bus out of the talk-addressed state.

To send the UNT command, the controller places the value 95 on the data lines with ATN asserted.

ADDRESSED COMMANDS

Addressed commands are commands that are sent to specific instruments on the bus. The controller sends addressed commands by placing certain values on the data lines with ATN asserted. Addressed commands include Group Execute Trigger (GET), Go to Local (GTL), Parallel Poll Configure (PPC), Selected Device Clear (SDC), and Take Control (TCT). Also discussed are Parallel Poll Enable (PPE), Parallel Poll Disable (PPD), which function as secondary commands in conjunction with PPC.

All of the addressed commands, except TCT, require that the instrument receiving the command be listen-addressed. The TCT command requires that the instrument be talk-addressed.

Group Execute Trigger (GET)

The GET command causes all listen-addressed instruments incorporating a DT1 subset of the DT interface function to begin an operation (for example, for measurement instruments to make their measurements, output devices to output their signals, and so on)

To send the GET command, the controller places the value 8 on the data lines with ATN asserted.

Go To Local (GTL)

The GTL command causes all listen-addressed instruments to obey incoming commands from their front panel control switches. These instruments may store, but not respond to, commands coming through the GPIB interface until the instrument is once again listen-addressed.

To send the GTL command, the controller places the value 1 on the data lines with ATN asserted.

Parallel Poll Configure (PPC)

The PPC command enables a listen-addressed instrument incorporating a PP1 or PP2 subset of the PP interface function to respond to a parallel poll.

To send the PPC command, the controller places the value 5 on the data lines with ATN asserted.

Parallel Poll Enable (PPE)

The PPE command designates the sense and the DIO line on which instruments incorporating a PP1 subset of the PP interface function will respond to a parallel poll. If the instrument's individual status bit matches the assigned sense when the parallel poll is executed, the instrument asserts its assigned data line.

Instruments incorporating a PP2 subset of the PP interface function have their senses and DIO lines hardwired; such instruments can't be configured by the controller to respond to a parallel poll. They may be enabled or disabled for parallel poll by the controller.

Parallel Poll Disable (PPD)

The PPD command unconfigures a previously configured instrument from responding to a parallel poll.

Selected Device Clear (SDC)

The SDC command *clears* or initializes all listen-addressed instruments.

To send the SDC command, the controller sends a value of 4 on the data lines with ATN asserted.

Take Control (TCT)

The TCT command passes controller-in-charge status to a talk-addressed instrument.

To send the TCT command, the controller places the value 9 on the data lines with ATN asserted.

SERIAL POLLING

Serial polling is a means of reading serially the individual status messages of all instruments configured and enabled to respond to a serial poll.

Status Bytes

Each instrument on the GPIB incorporating the SR1 subset of the SR interface function has an eight-bit status byte. The status byte's contents describe (by means of a device-dependent code) the instrument's status.

Requesting Service

The *coding* of an instrument's status byte is almost entirely up to the instrument's designer, with one restriction: bit 7 (the second-most significant bit) is reserved to indicate whether or not an instrument is requesting service from the controller-in-charge.

Bit 7 of the status byte is known as the RQS or *requesting service* bit. A value of 1 indicates that an instrument is requesting service from the controller-in-charge. A value of 0 indicates that the instrument is not requesting service.

Instruments must do two things to request service:

1. Set the RQS bit of the status byte to 1.
2. Assert SRQ.

Conducting Serial Polls

The controller can be programmed to generate an interrupt and to serially poll each instrument on the bus, to determine which instrument is requesting service when the controller recognizes an instrument on the bus asserting SRQ.

NOTE

The controller can be programmed to conduct a serial poll anytime ; it does not have to receive a request for service from an instrument on the bus.

The controller conducts the poll by sending the SPE (serial poll enable) command, followed by a sequence of talk addresses. As each talk address is received, the instrument that is talk-addressed sends its status byte to the controller. The controller can then check the status byte to see if the RQS bit is set. Receiving the status byte from the instrument requesting service clears the RQS bit of that instrument (sets the bit back to 0).

When the instrument asserting SRQ is discovered, the controller usually terminates the serial poll (by sending the SPD command), and transfers control to a user-defined SRQ handler routine for the instrument requesting service. Reading the instrument's status byte clears the instrument's RQS bit. However, the factors that caused the instrument to request service in the first place must be handled, or the instrument may simply request service again and again.

PARALLEL POLLING

Parallel polling is a means of simultaneously reading the individual status messages of all instruments configured to respond to a parallel poll.

Individual Status Messages

All instruments with a PP function have an individual status message. This message can be set to *true* or *false* either remotely (by the controller) or locally (by the instrument itself), depending on the instrument's design.

Configuring the Bus for Parallel Polling

A series of PPC and PPE commands designates which instruments respond to parallel polls, which data line each instrument is to respond on, and on which *sense* of its individual status message the instrument is to assert its assigned data line.

Instruments incorporating the PP2 subset the PP interface function are configured locally. Their sense and data lines are set by the manufacturer, and cannot be changed by the controller.

Conducting the Parallel Poll

The controller conducts a parallel poll by asserting ATN and EOI simultaneously.

When instruments configured for parallel polling sense ATN and EOI asserted simultaneously, they check their individual status messages. If the message sense matches the sense assigned to the instrument, the instrument asserts its assigned line.

The result of the parallel poll is a bit-encoded integer from which the controller can calculate which instruments asserted their data lines during the parallel poll.

Tektronix Standard Codes and Formats

This appendix describes the Tektronix Standard Codes and Formats, a language used by Tektronix IEEE 488 instruments to communicate over a GPIB.

On a workstation, application programs are responsible for being compatible with the Tektronix Codes and Formats standard. The workstation doesn't directly conform to Tektronix Codes and Formats.

COMPATIBILITY

Using the GPIB is like using the telephone system. In both cases, a physical connection can be established and data can be transmitted, so that one person or one device can talk to another.

On the telephone system, unless both people speak and understand the same language, very little communication can take place. In addition to having a common language, they must have a common vocabulary.

Similar problems can arise between instruments that exchange data. The IEEE 488 standard defines a *telephone system* describing how the physical communications system is to be used, but it does not define the *language* sent over the bus. This can cause incompatibilities, even among devices that meet the standard.

For example, suppose a digital multimeter (DMM) has made a measurement of +3.75 volts and now has to transmit this information over the GPIB to a computer. Eight data lines are available to send information in byte-serial fashion.

The first question is: *What codes should be used to encode the five characters?* The 488 standard recommends ASCII code, but the DMM designer is free to choose any available code. If binary-coded decimal (BCD) is selected but the computer only understands ASCII, the DMM and the computer will be incompatible, even though both devices meet the IEEE 488 standard.

Suppose the DMM does send ASCII code. The question now is: *What format is the data to be in?* Does the DMM send the character sequence + 3.75, the most-significant-byte first, least-significant-byte first, or some other way? Again, the IEEE 488 standard says nothing, and the designer is free to create incompatibility.

Designers should use a common format. The Tektronix Codes and Formats standard specifies that instruments are to send ASCII data with the most significant byte first.

HUMAN INTERFACE

If you are designing measurement systems, you must be intimately involved with the instrument-to-instrument communication process. Instruments should be designed to communicate with one another in ways that you can easily understand when designing a measurement system.

For example, suppose a GPIB-programmable power supply needs to be set to 20.0 volts. The power supply can be designed in one of two ways: easy on the designer or easy on the user.

The first way is to design the instrument with minimal intelligence, so that it accepts an obscure code, which it in turn interprets and executes. Some power supplies must receive the character sequence *08E3* in order to put out 20 volts. The *0* stands for the 0-to-36 volt range, and the *8E3* is the hexadecimal code of the machine-language instruction to carry out the command.

The second way is to design the instrument with a microprocessor and intelligence to accept easily-understood numbers. A power supply designed this way outputs 20 volts when it receives a character sequence that is recognizable to people, as well as to the power supply. An example might be *VPOS 20.*

This second design method is a great deal more convenient for writing and debugging programs that communicate with the instrument.

In the future, most instruments will be intelligent and designed to interact with people. The Tektronix Codes and Formats standard promotes this type of instrument friendliness.

REPRESENTING NUMBERS

Because most GPIB instruments use ASCII characters to send and receive data, Tektronix has chosen ASCII coding as standard.

In addition, most instruments that send or receive numbers use the ANSI X3.42 standard format. This format states there are three types of numbers — integers, floating point, and exponents (called NR1, NR2, and NR3 numbers formats) — and they should be sent with the most significant character first. Table B-1 shows examples of these formats.

Unless there are numeric needs that are not met by this format, Tektronix instruments use the ANSI X3.42 Standard.

Table B-1
NUMBER FORMATS (ANSI X3.42)

Format	Example	Notes
NR1	375 +8960 -328 +00000	Value of 0 must not contain a minus sign.
NR2	+12.589 1.37592 -00037.5 0.000	Floating point should be preceded by at least one significant digit.
NR3	-1.51E+03 +51.2E-07 +00.0E+00	Value of 0 must contain NR2 zero followed by a zero exponent.

DEVICE-DEPENDENT MESSAGE STRUCTURE

A *message* represents a given amount of information whose beginning and end are defined. It is communicated between a talking device and one or more listening devices.

The Tektronix Codes and Formats standard defines the structure of device-dependent messages as follows:

- A message begins when ATN is unasserted, after the talking device is addressed to talk and the listening device(s) are addressed to listen.
- A message is composed of one or more *message units*, separated by a message unit delimiter.
- A message ends when the talking device asserts EOI.

There are two message unit types: *program message units* and *measurement message units*.

Overall Message Format

Program and measurement message units may be concatenated with message unit delimiters between them, and are terminated with a message terminator.

Such structure enables multiple message units to be strung together and output or input with a single controller program statement. Having both the message unit delimiter and the message terminator allows the possibility of two levels of processing.

A NULL MESSAGE is used for the talked-with-nothing-to-say message.

Message Terminator

The message terminator is the highest order separator defined by this standard.

Program Message Unit

A program message unit is a single instruction. It is used to program some action (which may change a setting, elicit an output, etc.).

A program message unit contains a header, header delimiter, data and data delimiter.

Measurement Message Unit

A measurement message unit differs from the program message unit in that it allows a header bypass; disallows a data field bypass; restricts the header to alpha-only characters and restricts the data types to exclude character and link data.

The measurement message unit represents a single measurement result. This format is preferred for sending the output of an instrument's measurement process (typically to a controller).

Measurement message units contain: an alpha header, header delimiter, non-character data, and a data delimiter.

Data Types

- Character Data
- Link Data
- ANSI Numeric Data Types
- ASCII-Encoded Special Numeric Data
- Floating Point Data
- String Data
- Binary Block Data
- End Block Data

MESSAGE CONVENTIONS

While standardizing the language used on the bus fosters greater compatibility between devices, it does not solve all compatibility problems. Well-defined operational conventions are also needed.

End of Message

Both talking and listening devices should agree on when a message ends. Obvious difficulties can arise when talker and listener don't agree; if the listener thinks the message has ended too soon, it garbles part of the message, while if it doesn't think the message is ended when it actually has, the listener *hangs* the bus waiting for a message that will never come.

The Tektronix Codes and Formats standard solves this problem by specifying that:

- Talking devices should assert EOI when they send the last byte of a message.
- Listening devices should understand that EOI becoming asserted signifies that the last byte of a message has been sent.

STATUS BYTES

The IEEE 488 standard defines a means for an instrument to send a status byte to the computer but, except for bit 7, the standard doesn't define the meaning of the bits. (The IEEE 488 standard defines bit 7 to tell whether or not a device is requesting service.)

However, there is a need for instruments to report status information or errors to the controller, so the Tektronix Codes and Formats standard establishes a status byte convention. One need is for instruments to report if they are executing a command, or ready to receive another command. Bit 5 is used for this purpose.

Another common need is for instruments to report if they are encountering abnormal conditions. Examples of abnormal conditions are internal error conditions within the device, erroneous program data sent to the device, incomplete or erroneous measurement data, or device-dependent limit or alarm conditions. There are more complex conditions besides busy/ready or normal/abnormal. These are listed in Table B-2.

Certain instruments may have conditions that are peculiar to them. To report these status states, bit 8 indicates that the status byte is an uncommon type.

Providing a standard coding for the status byte enhances the convenience to the person programming the system. If all the instruments have common status byte codings, then a common status byte handling routine can be written for all instruments, instead of a separate one for each instrument.

Table B-2
STATUS BYTE DEFINITIONS

Conditions	Binary	Decimal X=0	Decimal X=1
Abnormal			
Command error	011X 0001	97	113
Execution error	011X 0010	98	114
Internal error	011X 0011	99	115
Power fail	011X 0100	100	116
Execution warning	011X 0101	101	117
Internal warning	011X 0110	102	118
Normal			
No status to report out of the ordinary	000X 0000	0	16
Power on	010X 0001	65	81
Operation complete	010X 0010	66	82
User request	010X 0011	67	83
Request control	010X 0100	68	84
Passed control	010X 0101	69	85

QUERIES

Even with all the possibilities allowed by the status bytes, it is often necessary to send more detailed information from an instrument to a computer. This can be done with *queries.*

Queries take the form of a header followed by a question mark. Here are some example queries and their uses:

EVENT? This query performs two functions:

Return more detail about the event reported in the last serial poll status byte.

Provide a mechanism by which a controller may get information about events when the devices RQS assertion capability has been disabled.

SET? Requests an instrument to send the controller its present settings and other current state information. Sending this information back to the instrument at a later time returns the instrument to the state it was in when queried. This query makes it possible to develop a program using an instrument's front panel as input to the computer. Using this feature, a programmer never needs to know the instrument's GPIB commands.

ID? Makes an instrument identify itself by sending such information as its instrument type, model number, version of firmware, etc. This feature is useful for identifying a particular device and for self-configuring systems.

Defining a standard way to elicit responses from an instrument enhances the convenience to the system designer.

ADDITIONAL FEATURES

Besides standardizing the language that instruments use to communicate, the Tektronix Codes and Formats standard specifies certain instrument characteristics that guarantee maximum friendliness and dependable operation. Some examples are:

- An instrument always says something when made a talker. If it has nothing to say, it sends a byte of all ones with the EOI signal. This lets the listening device know that no meaningful data is forthcoming, and prevents tying up the GPIB while one device waits for another to talk.
- A listening device always handshakes. It doesn't stop handshaking just because it doesn't understand or can't execute a message. After receiving EOI, if the device is confused, it sends out a service request and, when serial polled, notifies the controller that the command can't be executed as sent.

  Under no circumstances does a device execute a message it doesn't understand.
- Instruments always send numbers in correct NR1, NR2, or NR3 formats, but receive numbers forgivingly. For example, instruments should accept such incorrect values as -0 or NR3 numbers without decimal points.
- If an instrument receives a number whose precision is greater than the instrument can handle internally, the number is rounded off (not truncated) to enhance accuracy.
- Instruments receive both characters and arguments in uppercase and lowercase and equate them (for example, a = A, b = B, and so on). This is important because some computer terminals can't send both uppercase and lowercase characters.
- An instrument sending data about its front panel uses headers and character arguments that correspond to the front panel's labels.

Glossary

alias
A different, usually shorter, name for a UTek command. Aliases exist in the C-Shell only. See *alias(1)* and *unalias(1)* for more information.

.aliases
A file stored in your home directory that defines *mail aliases* that you can use on the To: and Cc: lines of a mail message when you send electronic mail.

argument
Anything you enter to the right of the command name. Arguments change the action of a command. See also *option*.

autoboot
When the workstation is in autoboot mode, it automatically finds which device contains the operating system and then boots the operating system. Autoboot is selected with the configuration switches on the back panel of the workstation.

background
A command that doesn't tie up your terminal is said to be running in the background. You can enter a background command by ending the command line with an ampersand (&). The C-Shell provides *job control*, so you can move background jobs to the foreground (the Bourne shell doesn't provide job control).

base configuration
All the components that come standard with the workstation. Workstations consist of a base configuration, *options*, and *enhancements*.

BASIC
A high-level programming language designed to be easy to use. BASIC is an acronym for *Beginners All-purpose Symbolic Instruction Code*. The BASIC language on the workstation is based on the proposed ANSI standard BASIC language.

boot
The operation of loading an operating system and, possibly, other system software (from tape, hard disk, or diskette).

boot device
The storage device that the operating system is loaded from. The workstation usually boots its operating system from the hard disk.

Bourne shell
The standard *shell* of UTek. This shell was developed at AT&T Bell Laboratories.

C
The most commonly-used programming language on the workstation. Most of the operating system and utilities on the workstation are written in C.

command
A function performed for you by the *shell* (called a built-in command) or by a program that resides in a file in a directory in the file system.

configuration switches
The switches located behind the cable management cover at the back of the workstation that select which devices are the console and boot device. The switches also select diagnostic hardware tests.

console
The device that receives system error and diagnostic messages. You can select the device you want to be the console with the configuration switches on the back panel of the workstation.

control character
Special characters with special meanings to the system. You create control characters by holding down the <CTRL> key on your keyboard and simultaneously pressing another key, much like you would use the <SHIFT> key to produce uppercase characters.

cpio
Cpio is a command that archives files onto a storage media, such as diskette, streaming cartridge tape, or 9-track tape.

C-Shell
The *shell* that allows *aliases*, *history lists*, and *job control*. This shell was developed at the University of California at Berkeley.

.cshrc
A file that contains commands executed every time you invoke a C-Shell (for example, when you log in with the C-Shell as your login shell and when you

execute a C-Shell program). The *.cshrc* file, which is stored in your home directory, is usually used to set the **path** variable and to create *aliases*.

current working directory
The directory you are currently working in. To find the name of your current working directory, use the **pwd** (print working directory) command.

DCE
Data circuit-terminating equipment. A device that performs line interfacing functions between data terminal equipment (DTE) and data transmission lines. DCE is synonymous with *modem*.

/dev
The directory on the workstation containing files that control access to input/output and storage devices.

diagnostics
A group of programs that test components of the workstation for faults.

directory
A *node* in the hierarchical UTek file system that contains files.

disk
A disk is a storage place for files. Disks rotate like phonograph records and store files as magnetic patterns on the disk surface. Storing a file on a disk is called *writing the file on the disk*. See also *diskette* and *hard disk*.

diskette
A small flexible disk, often called a floppy disk, on which you store files or data.

display processor board
A printed circuit board that provides computing resources for the *display subsystem*. The display processor board mounts inside the workstation, above the computer board.

display subsystem
A high-resolution, bit-mapped, color or monochrome video display with a keyboard, mouse, cabling, and controlling circuit boards.

DTE
Data terminal equipment. A device that can send and/or receive messages. DTE usually refers to computers and terminals.

enhancement
A product you can add to your workstation to expand its capability.

environment
A group of variables whose values supply programs with information. The Bourne shell and the C-Shell each have different commands that let you assign values to environment variables and let you examine the values of environment variables. See *printenv(1)*, *export(1)*, *setenv(1)*, and *environ(5)* for more information.

/etc/hosts.equiv
A file that contains the names of other workstations you can access over a *local area network* (LAN).

/etc/termcap
A file that contains codes and abbreviations that tell the operating system how to communicate with various types of terminals. *Termcap* is short for *terminal capabilities database*.

Ethernet
A *local area network* developed by Digital Equipment Corp., Xerox, and Intel. Ethernet uses a coaxial cable to transfer data between up to several hundred machines at a rate of 10 megabits per second.

export
If you use the Bourne shell, you use the **export** command to put variables in your environment.

.exrc
A file, stored in your home directory, that contains commands that set up the **vi** editor.

filename
Every file in the file system has a name. A filename can be up to 255 characters long and can contain special characters as well as alphabetic and numeric characters. Beware of using *metacharacters* in your filenames. Filenames beginning with a period (.) are treated special by UTek.

filename completion
A feature of the C-Shell that lets you enter filenames as arguments to commands by typing just enough characters to make the filename unique and then pressing the <ESC> key.

filename listing
A feature of the C-Shell that lets you find out what filenames match what you have typed so far when entering a command.

finger
A command that prints information about users on your system.

foreground
A command that ties up your terminal while it is running is said to be running in the foreground. There is no prompt displayed, and you cannot enter other commands. Contrast with *background*.

FORTRAN
A programming language designed for scientific and mathematical applications. FORTRAN is short for *FORmula TRANslation*. The FORTRAN language used on the workstation is Fortran 77. See *f77(1)* in the *UTek Command Reference* manual.

gateway node
A computer that is physically connected to more than one *local network*. Such a computer acts as a gateway for communications between local networks.

grep
A command that finds patterns in files. See *grep(1)* in the *UTek Command Reference* manual.

hard disk
On the 6000 Series system, a small disk encased in its own drive on which the UTek operating system and your files and data are stored. Also called a Winchester disk. See also *disk* and *diskette*.

hard disk drive
A direct-access mass storage device that uses magnetic disks (often called Winchester disks) to store data. See *disk*.

history
A feature of the C-Shell that lets you reenter commands you entered earlier by typing a short string of characters. See *csh(1)* for more information.

history list
The list of commands you have entered, which you can reenter with the **history** feature of the C-Shell. See *csh(1)* for more information.

home directory
The directory UTek puts you in when you first log in. All users have a home directory, which is specified in the *password file*.

job control
A feature of a *shell* that lets you control commands (called jobs) that you enter. Job control lets you move jobs between the *background* and the *foreground* and lets you temporarily stop running jobs.

kbaud
See *kilobaud*.

kbyte
See *kilobyte*.

kernel
The portion of the operating system that always resides in random access memory (RAM).

kilobaud
A data transmission line that operates at 1 kbaud transfers 1024 bits per second.

kilobyte
A kilobyte is 1024 bytes.

LAN
See *local area network*.

LED

Short for *Light-Emitting Diode*. LEDs are used as indicator lights on the back panel of the workstation.

local area network

A physical linking together of workstations and *gateway nodes* that allows any machine so linked to talk to any other machine that is also connected. Also referred to as a *LAN*. See also *local network*.

local network

Different from a *local area network* in that it refers to the machines that a workstation can communicate with without going through a *gateway node*. There can be many local networks connected to a single gateway node, but they are all part of the same LAN.

.login

A file that contains commands executed every time you log in (assuming you are using the C-Shell as your login shell) before you are given control of the terminal. The *.login* file, which is stored in your home directory, is usually used to set up your terminal and to set C-Shell and environment variables.

login

Also *logon*. The procedure of making a connection to UTek is called *logging in* or *logging on*.

login shell

Your login shell is the shell that UTek starts on your terminal when you log in. If the Bourne shell is your login shell, it executes commands from a file in your home directory, named *.profile*, before giving you control of the terminal. The C-Shell executes commands from files in your home directory named *.login* and *.cshrc* when it is invoked, and executes commands from a file named *.logout* when you log out.

.logout

A file that contains commands executed every time you log out (assuming you are using the C-Shell as your login shell). The *.logout* file is stored in your home directory.

mail alias
A name you use as an abbreviation on the To: and Cc: lines of a mail message. Mail aliases are frequently used to abbreviate a group of users and to abbreviate long paths to users on other machines you access over a network. You define mail aliases in the *.aliases* file in your home directory.

.manrc
A file that resides in your home directory and controls how the online manual command man operates.

Mbyte
See *megabyte*.

media storage device
A device that stores data. Common media storage devices are hard disks, diskettes and magnetic tape.

megabyte
A megabyte is one million bytes, or characters, of storage.

metacharacter
Metacharacters can be substituted for actual characters or groups of characters. For example, a question mark (?) replaces any single character, and an asterisk (*) replaces a number of characters.

.mh_profile
A file that resides in your home directory and controls how the commands in the MH Mail System operate.

modem
A device that passes data between two computers, or between a terminal and a computer, over telephone lines.

mouse
A pointing or graphical input device that rolls or slides on a desktop. The cursor on the screen reflects the mouse's motions. Press the buttons on the mouse to make selections. display screen to select objects and menu items.

multibus
A bus protocol developed by Intel that is widely used by mass storage peripherals, such as 9-track tape drives and hard disk drives.

multiprocessing
The capability of a computer system to execute more than one program at a time.

multipurpose
Describes a large computer that is used for a wide range of applications.

multiuser system
A computer system designed to be used by more than one user at a time. Contrast this with *single-user system* or *personal computer.*

network
Any group of workstations or computers connected to share data and information. Usually used to refer to a *local area network.*

Network File System
Sun's software running on the workstation that lets you access files and commands on other workstations as if they were on your workstation.

network transceiver
A device through which a workstation, computer, or terminal transmits data to and receives data from a *local area network.* Every *node* (workstation, computer, terminal) on a local area network must be connected to a network transceiver.

node
(1) A workstation, computer, or terminal that is connected to a *local area network.* (2) A directory in the hierarchical UTek file system.

normal mode
The normal operating mode of the workstation. The opposite of normal mode is *service mode.* The operating mode of the workstation is selected by the configuration switches on the back panel of the workstation.

operating system
The group of programs running on the workstation that control the workstation's computing resources.

option
(1) An optional argument to a command that alters its action. An option consist of one or more characters preceded by a minus (–) or a plus (+) sign. See *argument.* (2) A component you can substitute for a component of (or add to) the *base configuration* of your workstation.

Pascal
A highly structured programming language derived from ALGOL. The Pascal compiler used on the workstation is named **pc.**

password
A secret code associated with each *username*. Usually the only person who should know your password is you. To add or change a password, see the *passwd(1)* command.

password file
The password file, */etc/passwd*, contains information about the accounts of every user on your system. Each entry in the password file consists of seven colon-separated fields. The fields are, in order: login name (no uppercase), password (encrypted so no one can read it), user identification number (must be unique for each user), group identification number (must be unique for each group), real name and phone numbers (separated by semicolons), home directory, and login shell (blank indicates the Bourne shell, */bin/sh*).

path
A list of directories where the shell finds commands you enter. Both the Bourne shell and the C-Shell have different ways to add directory names to your path.

peripheral
Also *peripheral device*. Any device on your system that is used for input/output with the CPU. Peripheral devices include tape drives, disk drives, diskette drives, terminals, and printers.

peripheral server
A system that collects data from one or more computers to send to a peripheral device at a later time. A peripheral server can collect data from your system at a high rate of speed and store it until the peripheral it serves is free to accept the data. This frees your system from having to wait for a slow data transfer (to the peripheral) to occur before you can continue with other work.

personal computer
A small desk-top computer designed to be used by one person.

.plan
A file in your home directory that is displayed by the **finger** command when someone types **finger** *yourname*.

.profile
A file that contains commands executed every time you log in (assuming you are using the Bourne shell as your login shell) before you are given control of the terminal. The *.profile* file, which is stored in your home directory, is usually used to set up your terminal and to set environment variables.

.project
A file in your home directory that is displayed by the **finger** command when someone types **finger** *yourname*.

prompt

A message that the shell prints on your terminal to indicate that it is ready to receive your input. The default prompt (the one you get if you don't change it) for the Bourne shell is the dollar sign ($), while the C-Shell default prompt is the percent sign (%).

rcp

The **rcp** (remote copy) command copies files between your workstation and other workstations (and computers) connected to your *local area network.*

.rhosts

A file you create in your home directory that lets you or other users you specify log into your your account on your workstation without entering a password. See *rlogin.*

rlogin

The **rlogin** (remote login) command logs you into another workstation connected to your *local area network.*

rsh

The **rsh** (remote shell) command executes commands on another workstation connected to your *local area network.*

service mode

Bringing the workstation up in service mode lets you run diagnostic tests selected by the configuration switches to check the hardware of your workstation.

shell
A program that reads input from your terminal or a file, interprets that input, and invokes the appropriate commands. Shells have control structures, variables, commands, and comments that allow them to be used as a programming language.

shell variable
A string-valued variable that changes the way a shell operates. Bourne shell variables are called *shell variables*, and variables used by the C-Shell are called *C-Shell variables*.

single-user system
A system designed to be used by one person at a time. Contrast this with *multiuser system*.

single-wide board

stty
The **stty** command sets many terminal port parameters. For example, you can use **stty** to set the baud rate, parity, character echoing, and other parameters.

subsystem
A component of the workstation that only operates when used with the workstation, yet has its own computing resources (microprocessor and memory). Subsystems use their own computing resources to help reduce the load on the workstation's CPU.

sysadmin
The program that helps you perform system administration tasks, such as adding a user to your system and adding peripherals to your workstation.

system enclosure
The cabinet that houses the workstation. The system enclosure includes mounting locations for mass storage devices, a card cage for mounting circuit boards, a power supply, and connecting cables.

termcap file
The terminal capabilities database. This file, which is stored in */etc/termcap*, contains codes and abbreviations that tell the operating system how to communicate with various types of terminals.

tset

The tset command tells UTek what type of terminal you are using and how to communicate with that type of terminal.

tty

An abbreviation for *teletype* that is used on UNIX-like systems to mean any type of terminal.

UNIX

A multiuser, multiprocessing operating system developed by AT&T Bell Laboratories.

upwardly compatible

Describes computer programs that run on one range of computers and can be run (with little or no modification) on more advanced computers produced by the same manufacturer. For example, programs that run on the 6130 workstations also run on the 6200 Series workstations, usually with no modification.

username

A username (also called a *userid* or *user account* or a *login name*) is a unique identifier assigned to each user of the system.

UTek

The operating system of the workstation. UTek is based on UNIX.

vi

A screen-oriented text editor on your workstation. Vi, which is short for *visual*, is based on a line editor named ex.

INDEX

Manual Change Information

Affected Manual Part # 070-5307-00 **Change Reference #** C5/189

Manual Name 6130 System Users Guide

THIS IS A REPLACEMENT PACKAGE

1. Remove the appropriate pages from your manual and insert the attached pages.
2. Keep this cover sheet in the Change Information section at the very back of your manual for a permanent record.

Replace: Table of Contents, Sections 8 and 9, Appendices A and B, and Index, for UTek 3.0 upgrade and new GPIB driver.

Replace – 3.0 UTek 3.0 & new GPIB driver

Tektronix
COMMITTED TO EXCELLENCE