Asynchronous Serial Communications Programmer's Reference Manual

HP 3000 MPE/iX Computer Systems

Edition 8

32022-90052
E1098

Printed in: U.S.A. October 1998

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Acknowledgments

UNIX is a registered trademark of The Open Group..

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Contents

Contents

Contents

Contents

Contents

Figures

Figures

Tables

Preface

This manual documents functionality for the MPE/iX 6.0 release, as well as later releases, for HP 3000 Series 900 systems.

This manual documents a subset of the system-supplied intrinsics available through the native mode of operation for MPE/iX on HP 3000 Series 900 computers. The intrinsics described here are those that are particularly useful for the programmatic control of asynchronous devices. An asynchronous device is a device which can be configured as a terminal or serial printer and communicates with the system in an asynchronous, serial manner, as described in Chapter 1, “Introduction.”

NOTE

MPE/iX, Multiprogramming Executive with Integrated POSIX, is the latest in a series of forward-compatible operating systems for the HP 3000 line of computers.

In HP documentation and in talking with HP 3000 users, you will encounter references to MPE XL, the direct predecessor of MPE/iX. MPE/iX is a superset of MPE XL. All programs written for MPE XL will run without change under MPE/iX. You can continue to use MPE XL system documentation, although it may not refer to features added to the operating system to support POSIX (for example, hierarchical directories).

Finally, you may encounter references to MPE V, which is the operating system for HP 3000s, not based on the PA-RISC architecture. MPE V software can be run on the PA-RISC HP 3000s (Series 900) in what is known as compatibility mode.

Audience

This manual will be of greatest value to applications programmers who need to access and manipulate asynchronous devices during execution of a process. While some general and conceptual information is provided in the first two chapters of the manual, the intrinsic descriptions assume that the reader has a good basic understanding of system intrinsics and their use.

Related Manuals

DTC Planning Guide.

HP 2345A DTC Installation and Service Manual

HP 2340A DTC Installation and Service Manual

HP J2070A DTC Installation and Service Manual

Configuring Systems for Terminals, Printers, and Other Serial Devices

Troubleshooting Terminal, Printer, and Serial Device Connections

Configuring and Managing Host-Based X.25 Links

Using the OpenView DTC Manager

DTC Technical Reference Manual

MPE/iX Intrinsics Reference Manual

Accessing Files Programmer's Guide, Trap Handling Programmer's Guide, and Data Types Programmer's Guide

MPE/iX Commands Reference Manual Kit

Guide to This Manual

This manual is divided into the following chapters and appendices:

- Chapter 1, "Introduction," introduces Asynchronous Serial Communications (ASC) and the organization of the hardware and software used by the Distributed Terminal Subsystem (DTS). Includes a review of basic data communications concepts relevant to asynchronous devices connected to HP 3000 Series 900 computer systems.
- Chapter 2, "Controlling Asynchronous Devices Programmatically," summarizes the programmatic control of asynchronous devices. Provides an overview of the file system and device operation, along with a general description of using intrinsics in a program.
- Chapter 3, "Common Device Control Functions," contains additional information needed to use intrinsics successfully in a user program, and provides code samples to demonstrate how intrinsics can be used to perform some basic device control functions.
- Chapter 4, "Using FDEVICECONTROL," provides information on how the FDEVICECONTROL intrinsic may be used to perform terminal control functions. Code fragments demonstrate several of the functions performed by this intrinsic.
- Chapter 5, "Using Subsystem Break," explains the use of subsystem break and summarizes the intrinsics involved. Includes example code illustrating a subsystem break handler and a procedure to enable subsystem break.

- Chapter 6, “Typeahead Mode,” describes the programmatic controls involved in the use of typeahead mode. Code fragments illustrate the use of these controls.
- Chapter 7, “Programming for PAD, DTC Telnet, Telnet/iX Server, and VT Devices,” describes how PAD and DTC Telnet connections are implemented through DTCs. For PAD connection, a DTC must be equipped with a DTC/X.25 Network Access Card, and for DTC Telnet connections a DTC must be equipped with a Telnet Access Card. Both pieces of hardware require the appropriate OV DTC Manager network management software. This chapter explains differences involved with the programmatic control of these devices. This chapter also describes how connections are implemented via the Telnet/iX Server and via Virtual Terminal (VT) services.
- Chapter 8, “Intrinsics Reference,” contains a detailed description of the MPE/iX Native Mode intrinsics that are most useful for the programmatic control of asynchronous devices. Describes the syntax, parameters, and condition codes or status return for each. When applicable, functional returns, special considerations, and references to additional sources of information are included.
- Appendix A, “Comparing MPE/iX Systems to MPE V Systems,” provides a comparison between asynchronous devices connected to an MPE V system and those connected to an MPE/iX system.

1 Introduction

The HP 3000 Series 900 computer supports a wide range of useful peripheral devices. This manual describes how devices which are connected to a DTC and communicate asynchronously (such as terminals and serial printers) interact with the MPE/iX operating system. It also describes how asynchronous devices can be controlled programmatically through the use of system intrinsics. Only the subset of intrinsics useful for control of asynchronous devices is described here. For a complete listing of all the system intrinsics available with MPE/iX see the *MPE/iX Intrinsics Reference Manual*.

When you are programmatically controlling devices connected to your system, it is vital that you have a basic understanding of the way each specific device interacts with the computer. How this interaction occurs depends on such diverse factors as baud rate, transmission method and the purpose for which the device is being used. In order for successful communications to take place, the computer and the peripheral device must agree on the method of communication being used and must know exactly what to expect from each other.

This chapter provides an overview of how communications take place between an MPE/iX system and the asynchronous serial devices connected to it. The following topics are discussed:

- The types of devices that communicate in an asynchronous serial fashion.
- The physical components of the Datacommunications and Terminal Subsystem.
- The software that governs Asynchronous Serial Communications.
- The basic data communications concepts involved.

Asynchronous Serial Communications

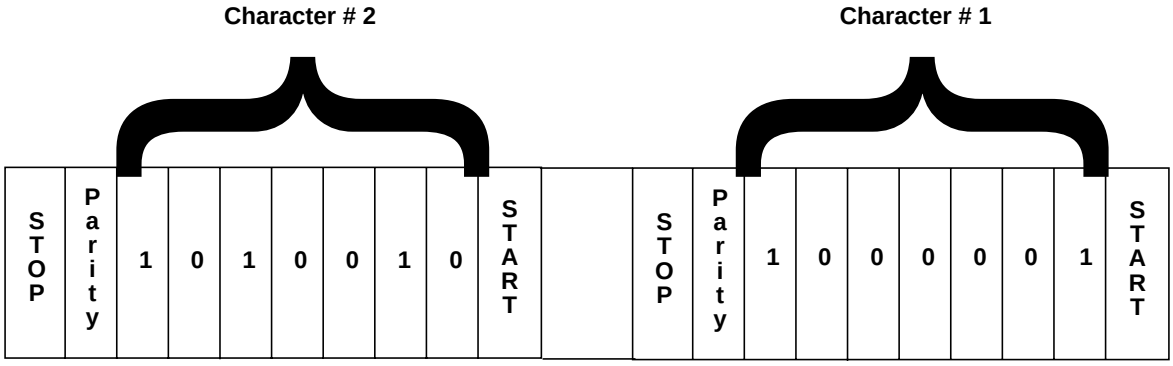
Asynchronous Serial Communications (ASC) is the term used to describe the specific manner in which communications take place between an MPE/iX computer and its associated terminals and serial printers.

Asynchronous refers to the scheme used to let the receiving end of a communication link know when data is being sent and when transmission of the data has completed. Each character is preceded by a special bit, called a start bit, which signals its arrival. The character is then followed by at least one stop bit, a second special bit which signals that transmission of that character is complete. Because this scheme allows data to be sent intermittently and does not require elaborate timing mechanisms, the asynchronous method is particularly useful for reading characters entered through a keyboard, where the time interval between the entry of two characters can vary greatly.

Serial transmission refers to the way each character is transferred. Characters transmitted serially are sent one bit at a time and received one bit at a time in the order of transmission. This is in contrast to parallel transmission schemes which transmit more than one bit at a time.

Figure 1-1 depicts transmission of data characters in an asynchronous serial mode. The arrow indicates the direction of the transmission.

Figure 1-1 Asynchronous Serial Transmission



Devices

The devices which communicate with MPE/iX asynchronously include a number of supported terminals and serial printers. Modems may also be a part of the communications link. The characteristics of each device are determined by the MPE/iX operating system as well as by the device itself.

Terminals

A **terminal** is a hardware device that enables an end user to communicate with the computer, and both send data to and receive data from the system. All terminals supported for use on MPE/iX systems include typewriter-like keyboards and CRT screens. Some personal computers capable of running in terminal emulation mode are also supported for asynchronous connection.

Terminal types are assigned to each terminal port configured for the system and serve to help define the terminal's features. For a complete discussion of terminal types see Configuring Systems for Terminals, Printers, and Other Serial Devices.

Serial Printers

Like terminals, **serial printers** can be connected to the system using asynchronous serial communications. Unlike terminals, serial printers are used only to receive information and to transfer that information to paper in a permanent, readable format. **Printer types** are used to help define the characteristics of each serial printer port configured for the system. A complete discussion of printer types can be found in the *Configuring Systems for Terminals, Printers, and Other Serial Devices* manual.

Plotters

Plotters are special graphic devices which allow you to transfer computer generated drawings to paper by plotting their X and Y coordinates and moving pens to corresponding locations on the output surface of the plotter.

Modems

In some cases asynchronous communications may take place over telecommunication lines. Because the signal used by computers (digital signal) is different from the signal transmitted over telephone lines (analog signal), a device called a **modem** is required on each end of such a link. The word “modem” comes from “modulator/demodulator”, which describes the function of the device.

System Console

The **system console** is a terminal used by the system operator to execute specific tasks for the purpose of managing sessions, jobs, and system resources. A system console is connected to each HP 3000 Series 900 computer through a special access port, and system console functions are controlled through the Console Management software. Some of the console functionality can be temporarily transferred to another terminal, or **logical console**. This can be useful if it is necessary to receive console messages on another device. Most system control functions, however, cannot be executed from a logical console.

While the system console is technically an asynchronous serial device, its functionality, as well as its physical and logical interfaces, are very different from those discussed in this manual. For more information on the system console see *Managing Peripherals* manual.

Non-Hewlett-Packard Devices

Theoretically, any device that conforms to Hewlett-Packard's asynchronous protocols and is capable of being physically connected to the system can be controlled through Asynchronous Serial Communications. This could include test instruments, data collection devices, etc. Such devices must be capable of performing XON/XOFF flow control, and should be configured as terminal type 18 or printer type 18. Extreme care must be taken when connecting any non-supported device.

NOTE Non-HP devices are not supported. HP is not responsible for troubleshooting customer problems that involve the use of non-supported devices

PAD Device Connections

MPE/iX systems are also capable of supporting asynchronous devices connected through a private or public PAD. While any device normally supported as an asynchronous device can be connected in this manner, there are certain differences in how they are configured, and in how they react to various programmatic controls. For information on configuring PAD connections or systems managed by an OpenView DTC workstation, see *Using the OpenView DTC Manager* manual. For information on configuring PAD connections on host-based systems, see *Configuring and Managing Host-Based X.25 Links* manual. Refer to Chapter 7, "Programming for PAD, DTC Telnet, Telnet/iX Server, and VT Devices," of this manual for more information on PAD connections and programmatic control.

Telnet

The Telnet protocol provides a standard virtual terminal connection between the HP 3000 and other HP and non-HP systems. MPE/iX Telnet connections can be made using one of two solutions: 1) via a direct HP 3000 implementation called "Telnet/iX" which enables users to have direct access to, or from an HP 3000 using the Telnet protocol. This product provides both *inbound* and *outbound* Telnet connections using only HP 3000 resident code. 2) via a Datacommunications and Terminal Controller (DTC) with a Telnet Access Card (herein referred to as "DTC Telnet") which provides only *inbound* Telnet access to an HP 3000 through a DTC.

Telnet/iX

Telnet/iX provides the Telnet service over Transmission Control Protocol/Internet Protocol (TCP/IP) on MPE/iX. The **Telnet/iX Client** enables users on an HP 3000 to have direct access to HP 9000, HP 3000, and non-HP systems that support Telnet and TCP/IP. The **Telnet/iX Server** enables users on a remote system running standard Telnet services to logon and run most applications on the HP 3000. Many MPE/iX file system intrinsics are supported over Telnet/iX Server connections. Please see the chapter “Programming for PAD, DTC Telnet, Telnet/iX Server and VT Devices” in this manual for more details.

DTC Telnet

A DTC with a Telnet Access Card (TAC) provides Telnet access to an MPE/iX system. Only one TAC is needed per network. The terminal data is transmitted to the Telnet Access Card in the DTC. The DTC then transmits the data to the MPE/iX system. For the purposes of asynchronous serial communications, the terminals connected through DTC Telnet are non-nailed devices with special control restrictions. Those restrictions are noted in this manual where appropriate. Otherwise, DTC Telnet is transparent to your program.

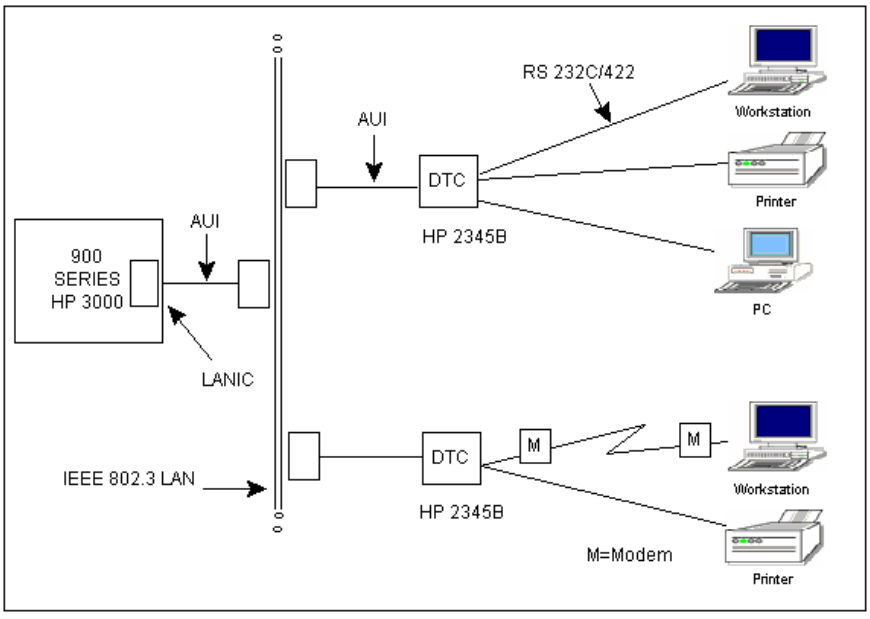
Datacommunications and Terminal Subsystem (DTS)

With the exception of the system console which has its own access port, all of the asynchronous devices listed here can be connected to HP 3000 Series 900 computers via the **Datacommunications and Terminal Subsystem** (DTS). DTS is made up of all of the datacommunications and terminal controllers connected and configured on the LAN, a LAN interface card that connects each host system to the LAN, the LAN cable, and the software that controls all related DTS hardware. Optionally, there may also be an OpenView workstation on the LAN, which is a personal computer that functions as a manager for the network. Each of these components are briefly described.

DTC

The **HP Datacommunications and Terminal Controller** (DTC) provides the connection between asynchronous devices and the HP 3000 Series 900 family of computers. Each DTC is housed in its own box and connected to the LAN. There are five types of DTCs currently supported, the DTC 72MX (HP J2070A), the DTC 48 (HP 2345B), the DTC 16 (HP 2340A), the DTC 16iX (HP J2062A) and the DTC 16MX (HP J2063A). Figure 1-2 represents the location of the DTCs in relation to the host systems, the LAN, and the terminals/printers connected to the network via the DTCs. (Note that the network represented in this figure includes a single MPE/iX system, two DTC 48s, and does not include an OpenView workstation.)

Figure 1-2 HP Distributed Terminal Controllers



If used for direct connections and remote modem connections only, each HP 2345B DTC (DTC 48) can provide connections for up to 48 directly connected devices, 36 remote devices using modems, or a combination of both.

For HP 2345B DTCs, up to three of the DTC's slots can contain a **DTC/X.25 Network Access Card**. This card provides access to X.25 networks, and can be used to connect asynchronous devices through PAD connections. Alternatively, a single Telnet Access Card per DTC can be used to access MPE/iX systems from ARPA systems.

Each HP 2345B DTC provides six slots, each of which can contain one of the following connector cards:

- Eight 3-pin RS-232-C local connections.
- Eight 5-pin RS-422 local connections.
- Six 25-pin RS-232-C modem connections.
- DTC/X.25 Network Access card (up to three per DTC, slots 1 through 5 only).
- Telnet Access Card (only 1 per DTC, slots 1 through 5 only).

Each HP J2070A (DTC 72MX) can provide connections for up to 72 local and modem devices, if used for direct and remote modem connections only. It provides four slots, with one slot used for the LAN card, and three slots each of which can contain one of the following cards:

- Asynchronous processor board with three 62-pin connectors, for a total of 24 RS-232 local or modem connections.
- Asynchronous processor board with three 62-pin connectors, for a total of 24 RS-423A local connections.
- DTC/X.25 Network Access card (up to three per DTC).
- Telnet Access Card (only 1 per DTC).

The DTC 16 contains three slots. The first two slots and only the first two slots (card #0 and card #1) can contain direct connect or modem cards. The third slot (card #2) can contain only a DTC/X.25 Network Access Card.

Each DTC 16iX and DTC 16MX can provide RS-232 or RS-423 port connections for up to 16 asynchronous devices. They both have two port connectors built directly into their backplanes, each of which can be connected to a distribution panel which allows either all direct connections or all modem connections. The DTC 16iX does not support ARPA Telnet or X.25 connections. The DTC 16MX provides AFCP connections for the HP 3000 environment and Telnet connections for the ARPA environment; it does not support X.25 connections for wide area networking. The DTC 16MX managed by an MPE/iX host is limited to identical functionality as the DTC 16iX.

NOTE

Direct and remote connections may be mixed on the same DTC, since its design is modular.

A Telnet Access Card and an X.25 Network Access Card *cannot* be on the same DTC 48. (Note that neither DTC/X.25 Network Access Cards nor a Telnet Access Card may be mounted in slot 0 for DTC 48s.

On networks that are not managed by the OpenView DTC Manager, devices connected through the DTC function as if the connection were made through a single cable between the device and its associated MPE/iX host system. Such DTCs (and the attached devices) can access only their host system, and their operations are controlled by software that is downloaded from the MPE/iX host when the DTC is powered up or reset. The downloaded information includes configuration files that define the devices connected to the system, and operating code to control the functions of the DTC.

If, on the other hand, the OpenView DTC Manager is used for network management, configuration information and controlling code for the DTCs on the LAN are managed and downloaded by the OpenView workstation. Offloading network management functions from the host systems to the OpenView DTC Manager allows for much greater flexibility in how connections can be made through a DTC. For example, it becomes possible for a terminal connected to a single DTC to access more than one MPE/iX or UNIX system (or both MPE/iX and UNIX systems) on the LAN, since configuration files need no longer explicitly define connections to a single host system.

Regardless of which type of network management is used, once the controlling code and configuration information has been downloaded, the DTC is ready for operation. Devices physically connected to its ports and properly configured can then establish communication links with a host computer.

A DTC Self-Test and an offline Self Diagnostic program are available to diagnose any problems encountered when a DTC is powered-on or reset.

When an MPE/iX host manages your DTCs, an online diagnostic program, TermDSM, is available to help diagnose any problems which may occur during DTS operation. Use of the online diagnostics as described in detail in the *Troubleshooting Terminal, Printer, and Serial Device Connections* manual.

If an OpenView workstation is part of the network, the diagnostic functions otherwise performed through TermDSM are provided instead by the diagnostic facilities that are a part of the OpenView DTC Manager software. See *Using the OpenView DTC Manager* manual for more information on using these facilities.

Local Area Network

HP 2345B Datacommunications and Terminal Controllers are connected to the system via a **Local Area Network** (LAN) connection, over coaxial cable which conforms to the IEEE 802.3 standard. Hewlett-Packard provides two versions of this cable, **ThinLAN** and **ThickLAN**, each of which provides certain specific advantages to the user.

Each DTC has a unique **LAN address**, also referred to as the **node address** or the **station address**, which serves to identify the DTC as a node on the LAN. You should keep in mind, however, that while DTCs are treated as nodes for purposes of configuration, they are not true nodes in the sense that nodes are normally defined. DTCs are dedicated communication devices; they are not self-contained, general-purpose computer systems.

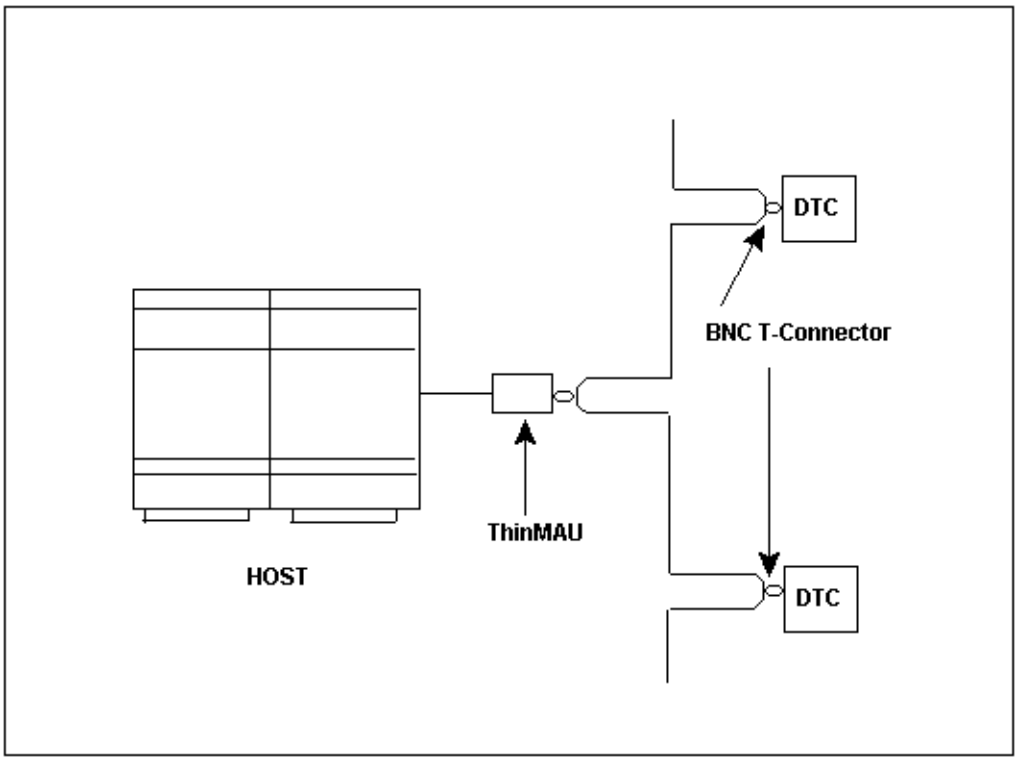
LAN Interface Card (LANIC)

Every computer in the HP 3000 Series 900 family of computers includes a **LAN Interface Card** (LANIC) as part of its basic hardware. This card is located in the System Processing Unit (SPU) and enables the system to communicate with asynchronous devices via the LAN by providing the basic connection between the computer and the LAN.

ThinLAN Connection

Two versions of LAN cable are available. The first version, **ThinLAN** (Type 10BASE2), is available in connection lengths of up to 185 meters (555 feet), and supports up to 30 nodes. ThinLAN provides an economical solution where DTCs are located fairly close to their associated computer systems. ThinLAN cable is very flexible and easy to install, and can be bent around objects to accommodate its use in confined areas. The ThinLAN is connected to each computer on the LAN via a **ThinMau** (Medium Attachment Unit), with a small **BNC T-connector** used to connect each DTC to the LAN. (Note that the ThinMAU for the DTC itself is built into the DTC box.) You can connect up to three segments of the ThinLAN cable with two repeaters, thus extending the length of the LAN to 555 meters. Figure 1-3 depicts a configuration using ThinLAN.

Figure 1-3 ThinLAN Configuration

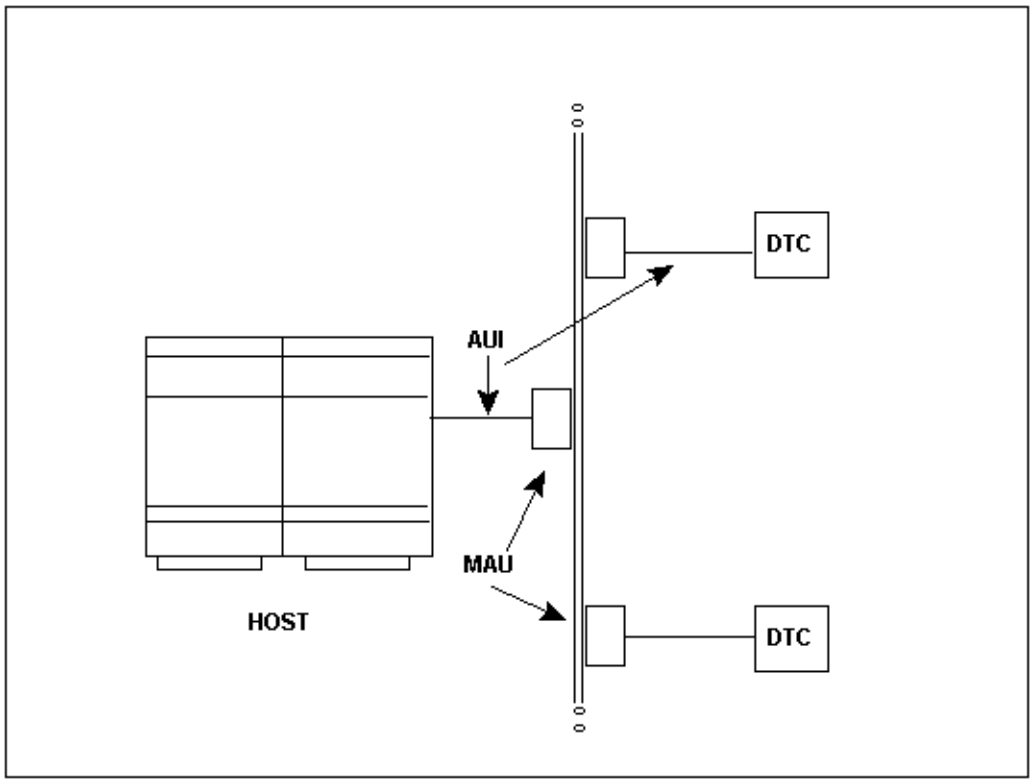


ThickLAN Connection

The second version of the LAN cable, **ThickLAN** (Type 10BASE5), offers maximum connection lengths of up to 500 meters (1,600 feet.), and supports up to 100 nodes. Each node on the LAN is connected via a **Medium Attachment Unit** (MAU). An **Attachment Unit Interface** (AUI) cable runs between each node and its MAU, as well as between the host computer and its MAU. You can connect up to three segments of the ThickLAN cable with two repeaters, thus extending the length of the LAN to 1,500 meters. A ThickLAN configuration is depicted in Figure 1-4.

More information on cabling and connection options can be found in the *LAN Cable and Accessories Installation Manual*.

Figure 1-4 ThickLAN Configuration



OpenView Workstation

As of MPE/iX version A.30.00, an OpenView workstation may also be a part of the Datacommunications and Terminal Subsystem. The OpenView workstation is a personal computer which is properly configured as a part of the LAN, and runs the OpenView DTC Manager software.

The OpenView DTC Manager is an application based on OpenView Windows. It provides an easy to use network management facility that allows you to configure, monitor, and diagnose the network and its components.

The OpenView DTC Manager assumes many of the network management functions that would otherwise have to be performed on an MPE/iX host system, including configuring the devices connected through the DTCs on the LAN and uploading and downloading the code that controls DTC operations.

The functionality of the DTC is expanded when network management is performed by the OpenView DTC Manager. It becomes possible for the DTC to act as a true data communications server, providing access to X.25 networks through the DTC/X.25 Network Access. Network access can include asynchronous connections through a private or

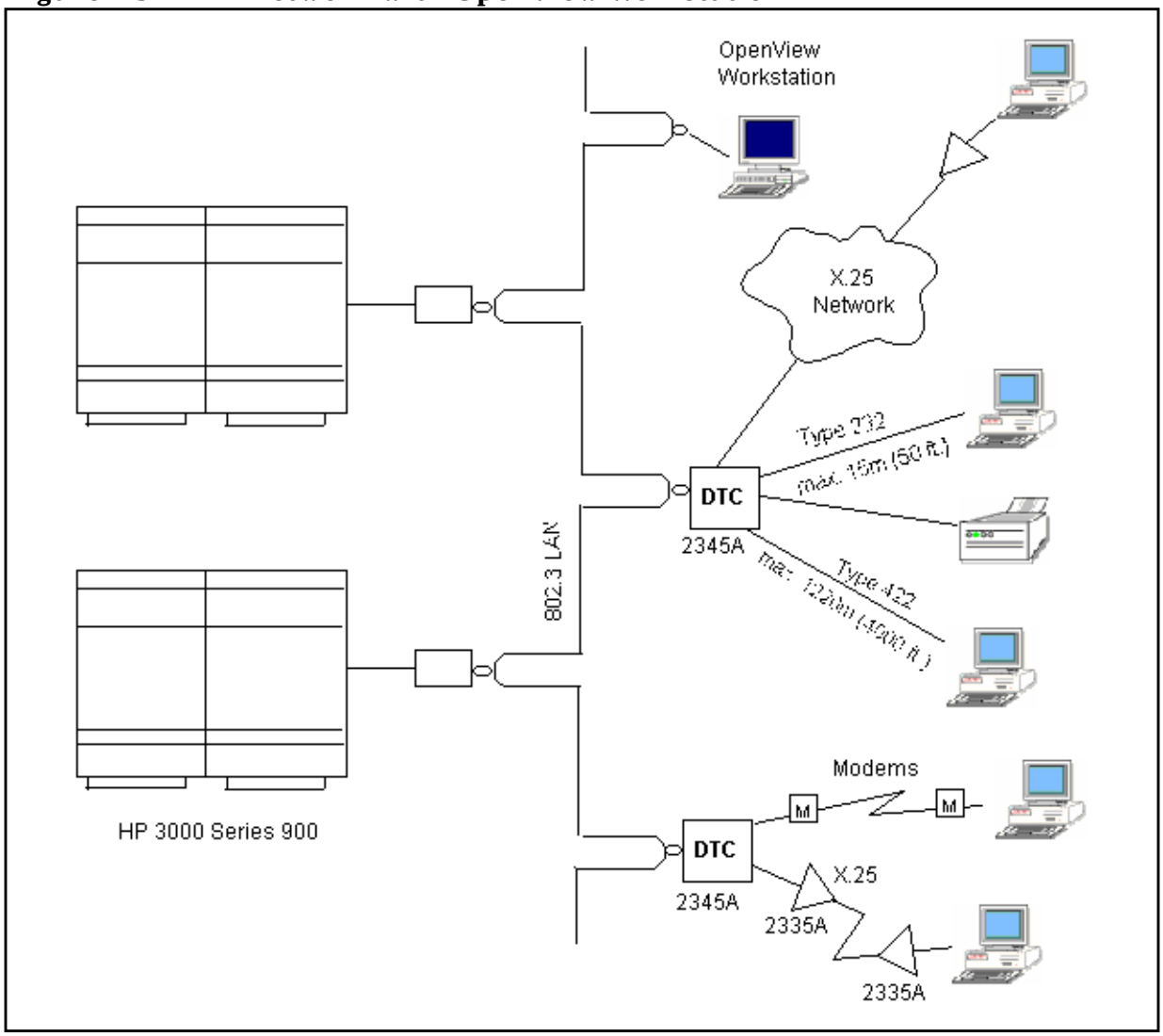
public PAD, as detailed in Chapter 7, “Programming for PAD, DTC Telnet, Telnet/iX Server, and VT Devices,” of this manual. DTC Telnet enables Telnet users to access MPE/iX hosts.

Additionally, terminals connected through DTCs on LANs managed by the OpenView workstation may be configured so that they are capable of establishing connections with more than a single MPE/iX system on the LAN. This ability is referred to as **DTC switching**, and is possible because the software that manages terminal connections resides on the OpenView workstation, and not on a specific MPE/iX host system.

When switching is allowed the user at a terminal can select the MPE/iX system to which they want a connection to be established through a special **DTC User Interfacer**. See *DTC Planning Guide* for more information on the DTC Services.

Figure 1-5 depicts a LAN managed by an OpenView workstation.

Figure 1-5 Network with OpenView Workstation



ASC Software Overview

The Asynchronous Serial Communications software provides the interface between the IEEE 802.3 LAN and the MPE/iX system.

Control

The software is included with the MPE/iX Fundamental Operating System (FOS). MPE/iX system software mirrors its related hardware in the sense that it is structured in modules, with each module handling the control of a specific type of hardware device. These modules, or managers, know which devices of their type are configured, or currently connected through a DTC connection, and handle all functions related to those devices. They manage terminal and printer I/O and are responsible for detecting errors that take place during communications and reporting them to the MPE/iX system. On LANs that do not include an OpenView Workstation, the ASC managers also manage the downloading and uploading of information for the DTCs configured on the system.

The ASC managers define the characteristics of the devices they control. They determine, among other things, the type of communications protocol being used. The way in which the ASC software expects devices to act is defined in part at configuration time, but can be altered programmatically through use of the system intrinsics described in this manual.

The ASC managers provide the interface between asynchronous devices and the File System. The File System is the part of the MPE/iX operating system that manages information being transferred or stored with peripheral devices. More information on the File System and how intrinsics are used to control devices can be found in Chapter 2, “Controlling Asynchronous Devices Programmatically.”

Configuration

The MPE/iX system keeps track of all devices connected to it by means of **configuration files**. The network configuration information necessary to initialize and control the Datacommunications and Terminal Subsystem (DTS) resides in the `NMCONFIG.PUB.SYS` configuration file. This file is created through use of the **Node Management Configuration Manager** (NMMGR) utility.

NMMGR is described in *Using the Node Management Services (NMS) Utilities*, while *Configuring Systems for Terminals, Printers, and Other Serial Devices* provides a detailed discussion of the configuration process.

Configuration with OpenView DTC Manager

Effective with version A.30.00 of MPE/iX, HP 3000 Series 900 and HP 9000 Series 800 computers may be part of a network managed by the OpenView DTC Manager. Devices connected to the DTCs on such a network are configured through a PC-based network management workstation which is also attached to the LAN, and the DTC control software is subsequently downloaded from the PC.

The connection between each host computer and the LAN is configured using NMMGR, along with profiles of the asynchronous device types that will be accepted for connection to the host, and the ldev numbers of devices that require a permanently associated (nailed) connection. However, many of the other parameters usually associated with a connection between a host system and each individual asynchronous device no longer need to be configured through NMMGR.

Nailed and Non-Nailed Connections

Asynchronous devices attached to DTCs managed by an OpenView workstation can be configured as either nailed or non-nailed devices on the MPE/iX host systems to which they have access.

A **nailed** device is one that is permanently associated with an ldev number through the NMMGR configuration of an MPE/iX system. Any time a connection exists between a system and one of its nailed devices the same ldev number will belong to that device. Since only nailed devices can be accessed programmatically, all printers must be nailed, as must any devices that will be accessed as programmatic devices at any time.

A **non-nailed** device is a device that is able to establish a connection to an MPE/iX host system but has no permanently assigned ldev number in the NMMGR configuration file of that system. The user of a terminal connected as a non-nailed device can establish a session provided a connection is available and the MPE/iX host configuration specifies a device profile matching the characteristics of the device requesting the connection. When the user logs on to the system, an ldev number is assigned from a pool of available ldev numbers defined through the host system's NMMGR configuration.

When the connection is ended (the user logs off) the associated ldev number is returned to the pool of ldev numbers and becomes available for use by a different device connection.

The use of non-nailed device connections provides several major advantages. Non-nailed connections simplify the configuration process that must be done on each MPE/iX system, since configuration values do not have to be entered separately. It is possible, if you specify a sufficient number of non-nailed connections on each host, for you to add

asynchronous connection capabilities by using the OpenView DTC Manager workstation instead of the host. This prevents you from having to reconfigure your HP 3000 Series 900 computer.

NOTE

The concept of nailed and non-nailed devices deals only with the association of a device to an ldev number on a specific MPE/iX system. It is independent of the concept of switching, which deals with the ability of a single device to establish a connection to any of several MPE/iX systems on the LAN.

A terminal connected through a DTC on a LAN with multiple MPE/iX or UNIX systems may be configured as a nailed device on more than one host system. Alternatively, it may be nailed on one host system and not nailed on another, or it may be configured as a non-nailed device on all the systems to which it has access.

For more information on configuration of asynchronous devices on networks managed by an OpenView network management workstation see the *Using the OpenView DTC Manager*, as well as *Configuring Systems for Terminals, Printers, and Other Serial Devices*.

Data Communications Concepts Reviewed

The hardware and software described in this chapter conforms to specific rules designed to make data communications between devices possible. To use this manual as a programming guide, you need a good understanding of these rules and of data communications concepts in general. If you are unfamiliar with data communications concepts, it would be helpful for you to take a general course in data communications before you use this manual extensively. You may also want to review other Hewlett-Packard documentation, as listed below:

- *DTC Installation and Service Manual*, the hardware reference for the DTC.
- The manual for the peripheral device.
- *DTC Planning Guide*.

The material in the rest of this chapter is designed to serve as a review of several concepts that are central to successfully connecting terminals and printers to a HP 3000 Series 900 computer.

Protocols

In order for electronic devices to communicate with each other they must conform to well defined sets of rules which govern factors ranging from hardware configurations to the actual process of data transfer. Such a set of rules is called a protocol. Data communications protocols are conventions set up to assure the orderly exchange of information between two or more data processing entities.

Hardware Protocols

The lowest level protocols deal with the hardware connection between two pieces of data processing equipment. The cables used to connect asynchronous devices to the Datacommunications and Terminal Controller conform to protocols defined by the **Electronic Industries Association** (EIA). These protocols, **RS-232-C** and **RS-422**, were developed and updated in conjunction with the Bell System and independent equipment manufacturers.

The RS-232-C standard defines the electrical circuit functions for 25 connector pins. This standard is implemented by Hewlett-Packard in two versions. The first version makes use of only three of the defined transmission circuits, `Data-In`, `Data-Out` and `Signal Ground`. This special implementation provides an inexpensive interface, since the cable consists of only three wires with the connector on one end replaced by a three-pin connector. However, it is limited to local (direct) connections, in distances up to 15 meters (50 feet).

The second Hewlett-Packard RS-232-C implementation makes use of additional electrical circuits defined by the standard. The cable and connectors more closely resemble conventional RS-232-C 25-wire cable and connectors. This implementation can be used for direct connections, subject to the same system limitations as the three-pin version. It must be used when modems are included (remote connection), because additional signals are generated by modems, and required for modem control. See the *DTC Installation and Service Manual* for a pin-out diagram of the signals employed by this implementation.

The RS-422 standard is implemented by HP in a five pin version. The connecting cable is composed of one twisted pair of wires for `Data-In`, one twisted pair of wires for `Data-Out`, and a single wire for `Signal Ground`. Devices can be connected via this interface at distances up to 1,500 meters (4,000 feet).

The coaxial cable used for the Local Area Network complies with the 802.3 standard developed by the **Institute of Electrical and Electronics Engineers (IEEE)**. This standard describes a transmission access method called Carrier Sense Multiple Access with Collision Detect (CSMA/CD).

Flow Control Protocols

Flow Control is the means by which the transfer of data between the system and an asynchronous device is regulated. It protects both the system and devices from data overruns. A data overrun occurs when the sender of data transmits that data faster than the receiver can accept it. Because the receiver cannot accept all the data being sent to it, it is said to be *overrun* with data. Overruns inevitably result in lost data.

With asynchronous devices on MPE/iX systems, flow control actually takes place between the device and the DTC. One main method of flow control is used: **XON/XOFF protocol**. Its fundamental purpose is to protect devices from overruns.

XON/XOFF protocol is controlled by the recipient of the data being transferred. The recipient sends an XOFF character (ASCII DC3) to the sender of the data if, and only if, it is unable to continue to receive data. The sender then suspends transmission and waits for the receiver to send an XON character (ASCII DC1), thereby signaling that it is once again ready to receive data.

The flow control method being used is sometimes referred to as the **pacing method**. For instance, the terminal settings for XON/XOFF protocol are called transmit pacing (`XmitPace`) and receive pacing (`RecvPace`).

The DTC is protected from overruns through an additional flow control mechanism referred to as the **read trigger**. The read trigger is also the ASCII DC1 character, and is generated by the ASC software when the system is ready to accept data.

There is also a flow control mechanism used between a system and its DTCs called the Transport Flow Control Protocol. Its fundamental purpose is to protect DTCs from being overrun by data from the system. It also handles error checking of the packets passed to and from the system through the LAN. This flow control mechanism does not directly affect the way devices act and react.

NOTE The characters XON (ASCII DC1) and XOFF (ASCII DC3) are defined as *special characters* and are reserved for the purpose of protocol. These characters may not be used as data except in binary mode, if binary mode is supported.

Local Device Settings

Each device has certain specific characteristics which define the way in which it physically interacts with the system. For HP terminals these characteristics are generally set by changing configuration parameters from the keyboard via terminal configuration menus. Terminal settings include such things as baud rate (transmission speed), parity settings, and terminal block mode functions. Terminal settings can usually be altered programmatically, through the use of appropriate escape sequences, as defined for each type of terminal.

You may hear the term **strap setting** used synonymously with device configuration setting. This is left over from a time when changing a device's characteristics meant physically opening its cover and moving a strap from one set of contacts to another. A number of settings are still referred to by their original alphabetic strap designations. For example, how you set the D, G and H straps (`Line/Page, InhHndShk` and `Inh DC2` settings respectively) determines how your terminal will respond to block mode processing.

Certain printer characteristics are also set locally at the device and may also be altered by transmission of appropriate escape sequences. Though printers have no need for all of the control mechanisms that are related to user input, such as echo, or block mode, they do have their own special set of control needs to determine how output appears on the printed page, and to define ways to prevent a loss of data.

An important distinction must be made between the physical setting of the device (what the device does and expects) and the programmatic setting of the ASC software (what the system does and expects). The two settings *must agree* with each other in order for data transmission to take place. If you programmatically alter the system setting through an intrinsic call, you must also make sure the device's physical setting

is changed, either through programmatic transmission of the proper escape sequences, or by sending a message to the terminal operator as a reminder to properly set the device.

For more information on the proper settings for terminals and printers, refer to the reference manuals for each device.

2 Controlling Asynchronous Devices Programmatically

Programmatically Once an asynchronous device is properly connected and configured on a HP 3000 Series 900 computer, it can be accessed programmatically through the MPE/iX File System. The **File System** interfaces with the ASC software, which communicates with the device based on its port configuration. Your program has only to treat the device as a file a **devicefile**, and the computer will handle the actual communication with the device, including use of the proper protocol. This chapter discusses how the File System, system intrinsics, and your program can work together to control an asynchronous device. The following topics are discussed:

- An overview of File System functions.
- Device operation modes.
- Use of intrinsics.

File System Overview

File System Overview All input/output operations on MPE/iX systems are done through the mechanism of files. All files are considered the same (part of the system file domain), and handled in the same way. Because of this, the way that input or output data is accessed and treated is the same no matter what medium it resides on. This feature is called **device independence**, and allows for a great deal of flexibility in handling data.

There are two basic types of files:

- **Disk files**, which are files residing on disk, immediately accessible to the system and potentially sharable by several sessions/jobs at the same time.
- **Devicefiles**, which are files currently being input to or from any peripheral device except a disk. A devicefile is accessed exclusively by the session or job that acquires it and is owned by that session/job until that session/job releases it or terminates.

File Equations

Because of device independence, files are not restricted to the same type of device every time a program is run. **File equations** can be used to specify the device you want your program to access at run time. File equations associate the formal file designator of a file to a specific device, or type of device. The formal file designator is the name by which your program recognizes the file. It is the file name coded in your program along with the program's specifications for the file.

As an example of changing device assignment at run time, consider a program designed to write information contained in a file with the formal file designator of *outfile* to a line printer. You may want to direct the output to a terminal on a one time only basis. You could do this by using a file equation to direct the file to the terminal's ldev. For a terminal with an ldev of 124 the file equation would be:

```
:FILE OUTFILE;DEV=124
```

where **FILE** is the command, OUTFILE is the formal file designator, and DEV=124 specifies the device configured as ldev 124.

File equations must be issued before a file is accessed, and they take effect when the file is accessed. They remain in effect until the job or session ends unless they are canceled with a **:RESET** command or overridden by another command for the same file.

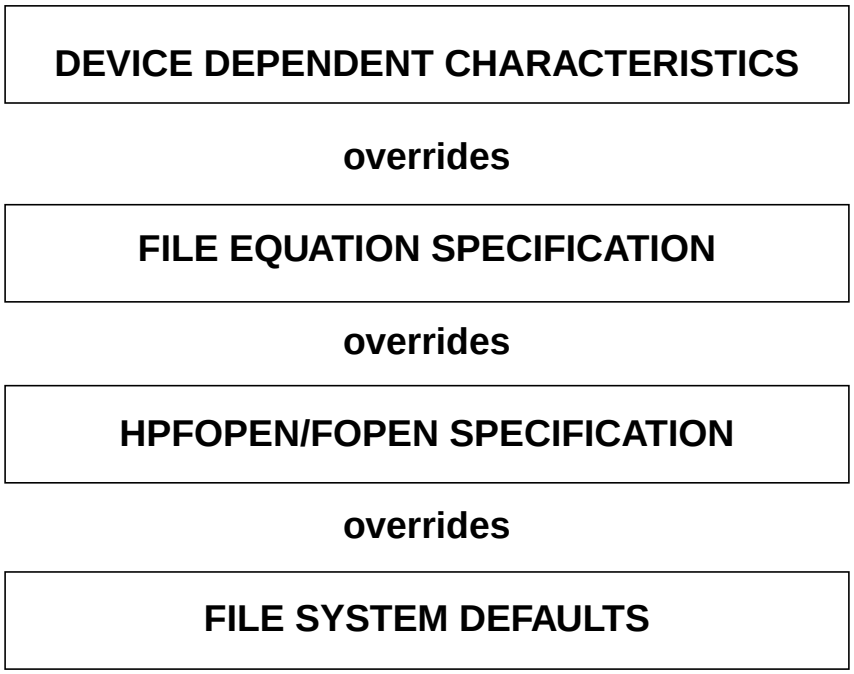
The **:FILE** command can be used to modify many programmatic or system default file specifications. There are, of course, certain restrictions on what can be modified by a file equation. The device must

be suited to the characteristics being specified. Data directed to terminals or printers, for example, must be in ASCII format, and have a blocking factor of 1. These are **device-dependent** characteristics, and if you try to specify something different it will be overridden. See the *Accessing Files* manual for a summary of device-dependent characteristics.

File System Hierarchy

There is a basic hierarchy to the way the File System overrides file specifications. Device dependence is at the top of the hierarchy and will override any attempts to modify file characteristics in a manner inappropriate to the device being used. Figure 2-1 shows which methods of specifying file characteristics take precedence in the File System hierarchy.

Figure 2-1 File System Hierarchy



The FFILEINFO intrinsic can be used to determine what specifications are currently in place for a given file. FFILEINFO is described in more detail in Chapter 8, “Intrinsics Reference,” of this manual.

See the *Accessing Files* manual for detailed information on the MPE/iX File System.

Device Operation Modes

An asynchronous device may be operated in one of five **device operation modes**. Each mode has a specific set of capabilities and is intended for a specific purpose. The five device operation modes are:

- Session-Accepting Devices.
- Programmatic Devices.
- System Console.
- Spooled Devices.
- Slaved Devices.

A device's current operation mode has a direct bearing on how a program may or may not exert control over the device's characteristics, as summarized below. For a complete description of the features and capabilities of each device operation mode, see *Configuring Systems for Terminals, Printers, and Other Serial Devices*.

Session-Accepting Devices

Session-accepting devices are opened with the MPE/iX **:HELLO** command and a session is initiated. The session remains active until the user logs off by using the **:BYE** command or initiates a different session by using the **:HELLO** command, or in the case of remote sessions, modem signals are dropped.Some error conditions will also cause sessions to end, as will the **:ABORTJOB** command, if entered by someone with the proper capabilities. (Note that neither **:ABORTJOB** nor **:ABORTIO** have any effect if issued against a terminal that is currently accessing the switching user interface.)

Session-accepting devices, also referred to as **log-on devices**, are normally terminals or personal computers running in terminal emulation mode. They can be configured as either nailed or non-nailed devices. During a session, the terminal is used for interactive (conversational) communication with the computer.

Session-accepting devices frequently come under the control of a system or user program, but only when the user who is logged-on to the device takes some action to allow this, such as issuing a valid MPE/iX **:RUN** command.

The device is then under the control of the program, and the program can read from and write to the device and programmatically control the interaction between the device and the computer through the use of intrinsics and the proper escape sequences.

At the completion of the program, control of the terminal is returned to the active session. For this reason it is critical that any device characteristics altered by the program be reset to normal or default settings before the session regains control. If a program fails to reset altered characteristics, the session may not be able to communicate successfully with the MPE/iX system.

NOTE

For similar reasons, it is strongly recommended that you disable system break through use of the FCONTROL system intrinsic in any program which significantly alters the operating characteristics of a device. If system break is not disabled and the **[BREAK]** key is pressed during execution of such a program (one that uses block mode, for example), the device may not be able to resume normal operations. Both of these recommendations are discussed in greater detail in later chapters of this manual.

Programmatic Devices

Programmatic devices are those which run under the control of a program running on the computer. The program accesses the device through the File System by treating the device as a file. The File System passes the information to the ASC software, which communicates with the device itself.

A programmatic device is considered to be **program captive**, because access to the device is through the program only. For instance, no MPE/iX commands can be executed from a terminal under the control of a program unless the controlling program initiates the command.

Programmatic devices can be used for input, output, or both, depending on the device and how it was opened by the controlling program.

There are many situations in which devices that are operating in one of the other device operation modes are also operating programmatically. As explained in the discussion above, session-accepting devices become programmatic devices when their users execute a **:RUN** command.

The system console can be a programmatic device in the same way. Slaved and spooled devices can be considered to be programmatic in the sense that they must be controlled through an application.

System Console

The CIO system console is the only terminal (or PC) that is not connected to the system through a DTC. The system is aware of its existence from system initialization. It is connected to a special interface card in the system cabinet known as the **access port**.

The system console has a unique set of functionality not available to any other device. It can receive console messages, and execute special commands through use of console attention characters that are not recognized by any other device.

A subset of console functionality may temporarily be assigned to another interactive device through use of the **:CONSOLE** command. This is called moving the **logical console** to the device. Any commands requiring use of the special console attention characters must continue to be executed from the system console. Note that the logical console may not be moved to a terminal connected via PAD or to a terminal with switching enabled.

The system console is also a session-accepting device, and therefore can be operated as a programmatic device as well. However, because the console serves a special purpose, some of the features of other session-accepting devices are not provided:

- The console may not operate at 19200 bps. Supported speeds are 300, 1200, 2400, 4800 and 9600 bps.
- No user block mode applications are supported. Except for NMMGR which is used to modify configuration files, no VPLUS block mode applications are supported.

CAUTION The console will continue to display messages it receives even if being operated in block mode. This may cause data on the block mode screen to be overwritten or lost.

Spooled Devices

MPE/iX is equipped with a spooling facility to assist the operation of certain nonsharable devices. When a spooler process controls a nonsharable device the device gives the appearance of being shared among several users.

The spooler intercepts data being output by various processes and temporarily stores it on disk instead of sending it directly to the device. The disk essentially becomes a staging area, while the spooler manages the selection of output spool files destined for the spooled device.

Printers are the only type of asynchronous device that can be spooled. Because it makes no sense for printers to be session-accepting devices (they can be used for output only), spooled devices are always programmatic devices.

Slaved Devices

If two devices share the same port in the DTC (one device is connected to the port and the other device is connected by an additional cable to the first device), only one device is recognized by the ASC software. The device that is recognized is referred to as the **master** device, while the other device is called a **slave** of the master. A wide variety of devices can be connected as slaves, including terminals, printers, card readers, flexible-disk drives, cartridge tape units and plotters. Slaved devices are considered programmatic because the device is controlled through an application. They never accept sessions and are not spooled.

MPE/iX System Intrinsics

The MPE/iX system provides a set of subroutines, callable by your program, that help you interface with the file system to accomplish various tasks. These subroutines are an integral, or intrinsic, part of the operating system, and are therefore referred to as system intrinsics. Intrinsics exist which allow you to perform a variety of recurring tasks, ranging from manipulating individual file attributes to manipulating processes, if you have the needed capabilities.

Intrinsics are program segments, no different from the code you write yourself except that they are invisible to you. They are usually coded in Pascal/iX (system language for HP 3000 Series 900 computers), but are generally callable from any supported language.

This manual discusses the subset of system intrinsics that are most useful for dealing with communications between the system and an asynchronous device. It is intended to provide the reader with a high level of detail regarding the use of intrinsics for controlling such devices. A complete description of all the system intrinsics available to you can be found in the *MPE/iX Intrinsics Reference Manual*, along with details regarding their use.

With one exception, the intrinsics described in this manual can be used in both native mode and compatibility mode programs. The exception is the HPFOPEN intrinsic, which can be called from native mode programs only. (There are other MPE/iX intrinsics which can only be called from native mode programs, but HPFOPEN is the only one discussed in this manual.)

Intrinsics and Asynchronous Serial Communications

Certain system intrinsics are particularly useful to you for providing an interface between the ASC managers that handle serial device connections and the MPE/iX File System. These are primarily the intrinsics that provide simple mechanisms for accessing and altering files, handling I/O operations, and checking file status. The following list provides a summary of these intrinsics and their functions. For a more complete description of the syntax and use of these intrinsics see Chapter 8, “Intrinsics Reference.” Additional information about the practical use of the intrinsics listed here can be found throughout this manual.

Summary of Intrinsics

INTRINSICS	PURPOSE
FCHECK	Requests details about file I/O errors.
FCLOSE	Closes a file.
FCONTROL	Performs a number of control operations on a file or device.
FDEVICECONTROL	Provides control operations to a printer, terminal, or spooled device.
FERRMSG	Obtains textual error message corresponding to an FCHECK error code.
FFILEINFO	Provides information about the characteristics of a file.
FGETINFO	Provides access and status information about a file.
FOPEN	Opens a file, defines its physical characteristics, and provides access to the file.
FREAD	Transfers a record of data from an input device to an array.
FSETMODE	Controls the output of linefeed.
FWRITE	Transfers a record of data to a file on any device.
HPFOPEN	Opens a file, defines its physical characteristics, and sets up access to the file. Provides access to mapped files.
IODONTWAIT	Initiates completion of a no-wait I/O request. Allows program processing to continue before an I/O operation completes.
IOWAIT	Initiates completion of a no-wait I/O request. Delays program processing until an I/O operation completes.
PRINT	Prints a character string on job/session list device.
PRINTFILEINFO	Prints a file information display on a job/session list device.
READ	Transfers a record of data from the file `$STDIN` into an array.
READX	Transfers a record of data from the file `$STDINX` into an array.
RESETCONTROL	Allows another subsystem break to occur.
XCONTRAP	Arms or disarms the user-written **[CTRL]Y** trap handling procedure.

You should note that the functions described here are specific to the way these intrinsics interact with and provide control for asynchronous communications ports. Many are useful for additional functions not described in this manual. See the *MPE/iX Intrinsics Reference Manual* for information about additional uses of these intrinsics.

Other Types of Intrinsics

The discussion in this manual pertains exclusively to system intrinsics. Certain application programs and system utilities also provide subroutines that are referred to as intrinsics. For example, VPLUS/3000 iX provides a set of intrinsics for handling page block mode programming. For a description of VPLUS/3000 iX intrinsics see the *VPLUS Reference Manual*. For descriptions of intrinsics provided by any other application or utility, see the documentation for that program.

Using Intrinsics in your Program

You can call any system intrinsic from any supported language, but the details of the calling procedure vary according to the language being used. In general, you should follow the steps as summarized:

1. Refer to the intrinsic description in Chapter 8, “Intrinsics Reference,” to determine the parameter types and their positions in the parameter list.
2. Declare variables to be passed as parameters, using types appropriate to the language you are using to write your program.
3. Include the name of the intrinsic in a declaration statement, if appropriate for your programming language.
4. Issue the intrinsic call at the appropriate place in your program.

Depending on the intrinsic call, error checking is provided either through condition codes or through a special status parameter. Because the condition code or status return value can provide you with useful information about what happened during execution of an intrinsic, you should routinely check for error conditions at the completion of each intrinsic call.

Condition Codes

Most of the intrinsics discussed in this manual use condition codes for error checking. Condition codes are status flags stored in two bits of the status register. Their values and general meanings are shown in Table 2-1. Because the condition code status flags are affected by almost every instruction, you should check condition codes immediately after execution of an intrinsic to make sure that the information being returned pertains to the intrinsic and not to some subsequent activity.

The method of accessing condition codes is language specific. Pascal/iX, for example, provides the function ccode, which returns the value 0, 1, or 2, depending on the current condition code stored in the status register. See the reference manual for the language you are using to determine how to access condition code information.

If the condition code indicates that an error condition occurred during execution of an intrinsic, you may want to obtain more specific information about what took place. You can do so by calling FCHECK to obtain details such as the error code associated with the problem that occurred, and FERRMSG to display the message associated with that error code. See the discussions of these intrinsics in the Intrinsics Reference section of this manual.

Table 2-1 Condition Codes

Condition Code	Value	General Meaning
CCE (=)	2	**Condition Code Equal.** This generally indicates that the request was granted.
CCG (>)	0	**Condition Code Greater Than.** A special condition occurred but may not have affected the execution of the request. (For example, the request was executed, but default values were assumed as intrinsic call parameters.)
CCL (<)	1	**Condition Code Less Than.** The request was not granted, but the error condition may be recoverable.

Status Parameter

While many intrinsics return an error parameter, only MPE/iX native mode intrinsics, such as the HPFOPEN intrinsic described in this manual, use the **status** parameter in place of condition codes as their error checking convention. The **status** parameter is a 32-bit signed integer which returns error information in one 16-bit field and subsystem information in its other 16-bit field.

The information returned by the **status** parameter is more explicit than that available through condition codes. The intrinsics that use the **status** parameter do not return meaningful condition code values.

The **status** parameter is optional, but it is strongly advised that you use this feature whenever it is provided. If an error or warning condition is encountered during execution of a native mode intrinsic and you did not specify the **status** parameter, the intrinsic causes the calling process to abort.

See the discussion of HPFOPEN in this manual, as well as the *MPE/iX Intrinsics Reference Manual* for more information on using the **status** parameter.

Abort Errors

The errors that return condition code or status information to your program are generally recoverable and do not cause your program to abort. Another type of error occurs if a calling program attempts to pass

illegal parameters to an intrinsic or does not have the capabilities needed to execute an intrinsic. Such an error is called an abort error and results in termination of the process.

Optional Capabilities

A number of system intrinsics can be used in a program only if you have optional capabilities assigned to you. Optional capabilities are assigned when your account is created, but can be altered by the System Manager or Account Manager through use of appropriate MPE/iX commands. See the *MPE/iX Commands Reference Manual* for more information on how to alter the capabilities assigned to an account, group or user.

If your program calls intrinsics which require optional capabilities you must prepare the program with these capabilities specified. Such a program can be run by users without the specific capabilities as long as it resides in a group with the proper capabilities assigned.

None of the intrinsics discussed in this manual require optional capabilities; however, if you intend to use NOWAIT I/O, you must have specified the NOWAIT option in the FOPEN or HPFOPEN call used to open the file. Specification of NOWAIT I/O requires that you have Privileged Mode (PM) capability. Once NOWAIT I/O has been specified you can control completion operations for I/O requests by calling the IOWAIT or IODONTWAIT intrinsics, as described in Chapter 8, “Intrinsics Reference.”

CAUTION The normal checks and limitations that apply to users with standard (default) capabilities are bypassed in Privileged Mode. It is possible for a Privileged Mode program to destroy file integrity, including the MPE/iX operating system software itself. Hewlett-Packard will, upon request, investigate and attempt to resolve problems resulting from the use of Privileged Mode code. This service, which is not provided under the standard Service Contract, is available on a time and materials billing basis. Hewlett-Packard will not support, correct, or attend to any modification of the MPE/iX operating system software.

Obtaining Information Using Intrinsics

Some intrinsics return information to the calling process, either as a value returned in one of the positional parameters, or through the special device used to designate functional return information.

Status Information

A number of intrinsics include output parameters in their parameter lists and can therefore be used to determine information about current device settings or devicefile specifications. The values returned in this way are a report of current status, rather than a result of intrinsic activity.

FGETINFO is an example of such an intrinsic. FGETINFO is called with the file number of the file about which you are requesting information specified in its first positional parameter. (The file number is obtained through an FOPEN or HPFOPEN call initiated by the calling process.) Access and status information about the file is returned in the other FGETINFO positional parameters. Information that can be obtained through a call to FGETINFO includes which file characteristics are in effect, the logical size of records in the file, the configured device type, and more. See Chapter 8, “Intrinsics Reference,” for a complete list of the information that can be obtained through a call to FGETINFO.

Some intrinsics contain a mixture of input and output parameters in their parameter lists. For example, some intrinsics return a file system error number in one of the parameters if an error message occurs during processing.

A special case exists for the FCONTROL intrinsic. FCONTROL provides a variety of control functions for files or devices, based on the value of a control code specified by the second parameter in its parameter list. The value of the second parameter determines the content of the third parameter, and whether it is used for input, for output, or both. In some cases the third parameter is used to set a value, while in other cases it returns a current status. In a few cases

this parameter passes a new value and returns the previous value to the program in the same call. FCONTROL functions are discussed throughout this manual.

Functional Returns

Some intrinsics return a value to the calling program through a functional return. Functional return values result from or are altered by the intrinsics of which they are a part. In examples and calling syntax depicted in this manual, they are shown in assignment statements, with the symbol `:=` used as the operator to designate that the functional return parameter is *assigned the value of* the result of the intrinsic call. The functional return parameter is shown on the left of the operator, with the call to the intrinsic on the right, as in the following example:

```
lgth:=FREAD(filenum,buffer,length);
```

Keep in mind that while functional returns are always demonstrated in assignment statements in this manual they need not be used exclusively in this way. They can also be used in output statements, as one of the parameters in a parameter list, and in conditional statements.

Unlike the values returned in positional parameters, which are a report of current status, functional return values are the result of the activity of the intrinsic. The FREAD intrinsic reports back the length of the message read. Likewise, the FOPEN (and HPFOPEN) intrinsic reports back a file number, which can then be used by other intrinsics called by the same program.

The intrinsics discussed in this manual which yield functional return values are FOPEN, FREAD, IOWAIT, IODONTWAIT, READ and READX. See Chapter 8, “Intrinsics Reference,” for their usage and syntax descriptions.

3 Common Device Control Functions

This chapter describes some of the more common ways that intrinsics can be used to control asynchronous devices and includes sample code segments to demonstrate these uses. The segments are coded in Pascal/iX, but should be helpful regardless of the language you are using. While details of declaring and calling intrinsics vary between languages, the principles of their application do not. For more information on using intrinsics with FORTRAN 77/iX or COBOL II/iX you should see the *MPE/iX Intrinsics Reference Manual* as well as the reference manual for the individual language.

The following device control functions are described in this chapter:

- Setting up access to a file and defining its characteristics through FOPEN.
- Reading from an asynchronous device using FREAD.
- Writing to an asynchronous device using FWRITE.
- Manipulating device settings through FCONTROL calls.
- Closing devicefiles using FCLOSE.

Opening Asynchronous Devicefiles

Devicefiles Before an asynchronous device can be used by your program, you must obtain access to the device and define the characteristics you want to be associated with the device for your application. You do this by using the FOPEN or HPFOPEN intrinsic to open a file on the device, passing the file specifications required by your program in the parameters of the call. See Chapter 8, “Intrinsics Reference,” for a description of the syntax of the FOPEN and HPFOPEN intrinsics.

When the FOPEN or HPFOPEN call is executed, MPE/iX will open the file as specified by the parameters of the call, establish communications between your program and the file, allocate needed resources (including the device on which the file is to reside), and return a file number to your program. If the file is not opened successfully, a 0 is returned as the file number. See Chapter 2, “Controlling Asynchronous Devices Programmatically,” for more information on condition codes.

The file number assigned by the open call is unique. Because every file has its own number, even if multiple files are opened on the same device, it is possible to use the file number to access a specific file in subsequent intrinsic calls. The file number remains associated with the file as long as it is opened; that is, until your program either issues an FCLOSE against that file or terminates.

The parameter values used for the FOPEN or HPFOPEN call determine what features will be associated with the file. System default values are assumed for any optional parameters not specifically designated.

Default values will also be assumed if you attempt to specify a value that does not make sense for the device on which the file is being opened. For example, the **userlabel** parameter of the FOPEN intrinsic has no meaning for files opened on terminals and, if specified, any value will be ignored and the default of 0 assumed.

The fact that parameters are ignored if not meaningful for a specific device provides an extra degree of flexibility when coding a program. If you foresee an instance when you might want to temporarily redirect a file normally resident on one type of device to another type of device, you can code the specifications that will allow you to do so. For example, you can specify the tape label option (*foption* bit (6:1) of the FOPEN intrinsic) for a file your program normally opens on a terminal even though this option has no meaning for files on asynchronous devices.

If you later run the same program, but issue a **:FILE** command to redirect that file to tape, there is no need to specify the tape label option.

File Open Intrinsics

The way in which file characteristics are specified differs depending on whether you are calling FOPEN or HPFOPEN. The parameters of the FOPEN intrinsic are positional. The file system expects to see them in a certain order and will not accept any specifications that fail to conform. An FOPEN call to open $STDIN for reading might be coded as:

```
fileid_in:=FOPEN(,octal('444'),0,-80);
```

HPFOPEN provides a superset of the features provided by FOPEN and uses combinations of *itemnum/item* pairs to specify file characteristics. An HPFOPEN call performing the same function as the FOPEN call above (and returning any error information in a variable called "status") might be coded as:

```
HPFOPEN(fileid_in,status, 53,1,
                           5,4,
                           6,2,
                           7,1,
                          11,0,
                          19,-80);
```

HPFOPEN can be used in native mode only. None of the extended features provided by this intrinsic apply specifically to asynchronous devicefiles. For consistency, the examples that follow will show the use of FOPEN exclusively. Be aware, however, that, in native mode, HPFOPEN calls can be used instead.

Your program may open multiple files on the same device, each with its own set of characteristics, logically separate access, and unique file number. Such files opened on a device are often referred to as ports.

The code fragments shown in Figure 3-1 open a read port and a write port on a terminal device. The parameters specify 80 byte ASCII records and carriage control. Condition codes are checked in the example through the Pascal/iX `ccode` function, which will return a value of CCE if no errors occur during the opens, or of CCG or CCL if errors do occur.

Figure 3-1 Opening a Read Port and a Write Port

```
{ Demonstrate the use of the FOPEN intrinsic }

program test;

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

function FOPEN : shortint;  intrinsic;


{-------------------------------------------------------------------------}
{ The OPENPORT function will try to open $STDIN for read access and $STDLIST  }
{ for write access.  If successful, it will return TRUE and the file numbers  }
{ for $STDIN & $STDLIST will be returned in input_filenum & output_filenum.    }
{ If an error occurs, it will return FALSE.                                    }
{-------------------------------------------------------------------------}
function OPENPORT (var input_filenum  : shortint;  {File number for read port}
                   var output_filenum : shortint   {File number for write port}
                  ) : boolean;                     {Returns TRUE if successful}
begin

openport := TRUE;                          {Assume success for now}

{-------------------------------}
{ Open $STDIN for read access... }
{-------------------------------}
input_filenum := FOPEN(               ,    {Formal designator not specified}
                       octal('444') ,    {Foptions = ASCII, $STDIN, CCTL}
                       0              ,    {Aoptions = Read access}
                       -80                {Recsize = 80 bytes}
                      );

if (ccode <> CCE) then
  openport := FALSE;                       {Failed to open read port}

{---------------------------------}
{ Open $STDLIST for write access... }
{---------------------------------}
output_filenum := FOPEN(                ,  {Formal designator not specified}
                        octal('414') ,  {Foptions = ASCII, $STDLIST, CCTL}
                        1              ,  {Aoptions = Write access}
                        -80                {Recsize = 80 bytes}
                       );

if (ccode <> CCE) then
  openport := FALSE;                       {Failed to open write port}

end;


begin
end.
```

NOTE

The `ccode` function used in the examples is a Pascal/iX intrinsic function that returns the condition code bits from the status register. If you are using a language other than Pascal/iX, you should see the reference manual for that language for information on how to access the condition code bits.

After the FOPEN calls are successfully executed, the variable *`fileid_in`* will contain the file number of a file that can be read from, and the variable *`fileid_out`* will contain the file number of a file that can be written to.

It is possible to issue a single FOPEN call that will allow you to read and write from a single file. However, to do so you must include some additional parameters in the FOPEN call. You must specify a formal file designator in the **formaldesignator** parameter of the FOPEN call (HPFOPEN item number 2) instead of using one of the system defined files (`$STDIN`, `$STDINX` or `$STDLIST`). You must also specify the logical device number of the terminal device in the **device** parameter (HPFOPEN item number 20). You can use the WHO intrinsic to determine the logical device numbers of the devices being used as the input and output devices for the current job or session.

Reading From Asynchronous Devices

Data is read from a terminal by means of the FREAD, READ or READX intrinsic. FREAD is the most versatile of the three, but requires the use of a file number. READ allows reading from `$STDIN` only; READX allows reading from `$STDINX` only. Neither READ nor READX requires the use of a file number, but their utility is somewhat limited.

The manner in which a read from a terminal device takes place is determined by the device control settings in effect for that device at the time the read is issued. You can modify many of these settings by using specific device control intrinsics prior to issuing a read.

The following paragraphs summarize the settings that are relevant when reading input from a terminal device and the effect these settings have on how a read will occur. See Chapter 8, “Intrinsics Reference,” of this manual for details on using specific device control intrinsics.

Input Modes

When devices are connected asynchronously, data is always sent one character at a time. The transmission of data may occur in either character mode or block mode, depending on the physical settings of the device and the settings of the ASC software controlling the device.

Character Mode

Under default conditions, transmission of data occurs in character mode, with characters transmitted to the DTC individually as they are typed in. When the read is terminated, the DTC transmits the accumulated data to the host. If DTC echo is enabled, characters are echoed back to the terminal screen as they are received. Under default conditions, character mode reads are terminated by a carriage return. Character mode is available with all supported terminal types and is the mode in which all terminals are opened, whether by a session or a program.

Block Mode

When a terminal is operating in block mode, characters are held in the terminal’s internal memory (buffer) as they are typed in. They are not transmitted to the DTC until after you have taken a specific action (normally, pressing the **[Enter]** key).

Data transmission may occur a line at a time (line block mode) or a page at a time (page block mode) depending on the settings of the terminal and the type of block mode being used.

Block mode is enabled programmatically by a block mode application when it is executed at a terminal. The terminal's configuration settings need to be changed also, in order for the terminal to function properly in block mode. This can be done by prompting the user to change the terminal's configuration settings or by programmatically sending escape sequences to the terminal.

Block mode allows a terminal user to see data on the screen as it is being typed in and to use the terminal's local editing and cursor control features to alter the data before it is transmitted to the DTC. It is available only on terminals that have block mode capabilities; all of the terminals supported for use on MPE/iX systems are capable of handling block mode data transmission. When using a PC, however, you should verify that the terminal emulation program you are using is capable of handling block mode transmission.

User block mode can be invoked through a call to FCONTROL(29) and makes it possible for your program to control the way in which a block mode transaction will take place. See the discussion of FCONTROL(28,29) in chapter 8 for more information on user block mode.

NOTE

A convenient way to handle block mode processing is provided by the VPLUS intrinsics.VPLUS intrinsics automatically perform many of the block mode terminal control operations that you are otherwise responsible for. It is recommended that you use this method whenever possible. See the *VPLUS Reference Manual* for more information on using the VPLUS intrinsics.

Field Mode

Enhancements have been added to provide the performance of block mode and the flexibility of character mode, called field mode.

Field mode was created to help application programmers control certain keys on the keyboard. Some applications require certain keys to be controlled by the application and not processed locally in the terminal. Block mode applications do not recognize any of these special keys except in some cases where they are processed locally by the terminal. To process the special characters would require reading characters one at a time. This would result in performance degradation.

Field mode allows the users to collect normal user data input without having to interrupt the CPU for each character, yet still have many ways to return control to the application when the user enters command input. The basic building blocks for field mode are summarized:

- Inter-byte timer. Provides a mechanism by which the DTC can terminate a read and forward data to the host. It is useful for collecting data where there is no guarantee that an end-of-record or

alternative end-of-record character will be received, or that a specified number of bytes will be received. It is implemented in FDEVICECONTROL 192, parm1 = 65.

- Multiple alternative end-of-record (AEOR) characters. Allows an application or user to specify up to 16 AEOR characters. When the DTC receives one of these characters, it completes the current pending read request with the received AEOR character as the last byte of data. This functionality is implemented in FDEVICECONTROL 192, parm1 = 66.

  It is also implemented in TTUTIL.PUB.SYS, in the Special Characters Screen. Refer to the *Customizing Terminal and Printer Type Files Using the Workstation Configurator* manual for more information on TTUTIL.

- Delete to backspace mapping. Allows the delete character (the **[DEL]** key) to be processed as a backspace character, in order to support ANSI mode terminals. These terminals send the delete character when the backspace key is pressed. This is implemented in FDEVICECONTROL 192, parm1 = 67.

- Escape sequence read termination. Enables an application to have a read terminated by an ESC sequence and receive the entire ESC sequence in the data. It is implemented in FDEVICECONTROL 192, parm1 = 68. An application cannot enable both the escape sequence read termination and the inter-byte timer method.

- Suppress echo of read termination character. Allows applications to determine whether or not to include the read terminators with the data that is echoed back to the device. It is implemented in FDEVICECONTROL 192, parm1 = 69.

Refer to FDEVICECONTROL in Chapter 8, “Intrinsics Reference,” for more information on these functionalities.

Data Editing Modes

When data is entered from a terminal, a number of control characters, referred to as special characters, signal the DTC or the ASC software to perform specific functions. The functions vary from deleting a character from the input stream to interrupting a program or subsystem. All special characters are stripped from the data after the system (or the DTC) responds to them. The specific characters that will be treated as special characters vary depending on the data editing mode being employed. Standard editing, transparent editing, and binary editing modes are available.

Standard Mode

Standard editing mode is enabled by default for asynchronous terminals. All special characters recognized by MPE/iX are available in standard editing mode. See *Configuring Systems for Terminals, Printers and Other Serial Devices* for a complete description of all the standard special characters.

Transparent Mode

In transparent editing mode, only a limited number of special characters retain their meanings and are acted on by the system. Also referred to as unedited mode, this facility allows many of the characters that would otherwise be treated as special characters to be passed through as input, without causing any control action to be performed or being stripped from the input data stream. Table 8-6 lists the special characters that retain their meaning in transparent mode. FCONTROL(41) can be used to enable transparent mode. The FDEVICECONTROL intrinsic can also be used to enable transparent editing mode. You will find examples of invoking transparent editing mode later in this chapter and in Chapter 4, “Using FDEVICECONTROL.”

Binary Mode

In binary editing mode no special characters are recognized. All characters are treated as data and passed through without any terminal control actions being taken. Only character mode processing is possible in binary mode, since there are no special characters available to control block mode processing. Any carriage control information contained in output data transmitted in binary mode will be stripped. Use FCONTROL(26,27) to enable and disable binary mode transfers.

Triggering Reads

With some exceptions, every time a read is issued on an asynchronous device a read trigger character is sent to the device, indicating that the system is ready to receive input. The read trigger is not transmitted over PAD connections and may not be issued if typeahead mode is enabled at a terminal and the typeahead feature is being used.

The normal read trigger character is the ASCII DC1 character (the same as XON). All terminal devices supported for use on HP 3000 Series 900 computers are capable of recognizing and responding to the DC1 read trigger character.

The read trigger is generated at the beginning of each read and signals that the system is ready to receive data. If the device is operating in character mode, the device can begin transmitting without any further exchange of protocol characters. If, however, the data is to be sent through block mode, the device must inform the DTC by sending an ASCII DC2 character in response to the read trigger. The DTC, now

informed that the data will be transmitted as a block, sends another read trigger when it is ready to accept the block of data. The second read trigger is referred to as a block mode read trigger and is the same character as the read trigger.

It is possible to specify an alternate character for use as the read trigger (and therefore also the block mode read trigger) through the FDEVICECONTROL intrinsic. You must take great care if you do so, however, to assure that no data is lost.

Terminating Reads

Reads can be terminated in a number of ways, depending on the programmatic controls issued by your program prior to posting the read. Both the input mode (character or block mode) and the data editing mode (binary, transparent, or standard editing mode) affect the termination of reads.

Binary mode reads cannot be terminated by the recognition of any character. Binary mode reads are terminated only by the byte count being reached or by the expiration of the read limit timer. Only byte count termination results in a successful binary mode read.

The following summarizes the events that will cause reads other than binary mode reads to terminate

- An EOR character is sent from the terminal. This includes the block mode read termination character (usually RS) if you are in block mode and any EOR character specified in an FCONTROL(41) call or through FDEVICECONTROL(192) with a **controlcode** value of 15 or 39. Reads terminated in this way terminate normally.
- An AEOR character defined in a preceding call to FCONTROL(25) is sent from the terminal. The AEOR character is included in the data and the byte count and the read terminates with an error condition (FSERR 31). An AEOR character will be recognized in either standard or transparent editing mode.
- The input byte count is reached. This count is set by a parameter in the read intrinsic used for the read (FREAD, READ, or READX). Reaching the input byte count causes all reads to terminate, regardless of any other condition.
- The read limit timer set by a preceding call to FCONTROL(4) expires. Setting this timeout value prevents a device from hanging because of an incomplete read. The read data is transferred to the user buffer and the read length is returned. The length is the number of characters entered at the time of the timeout, and the read terminates with SOFTWARE TIMEOUT (FSERR 22). Reaching the timeout limit causes all reads to terminate, regardless of any other condition. (During block mode reads, the timer halts when the DC2 character is received.)

- In block mode, the block mode read timer expires. This timer is active from the time that the second DC1 is sent until the RS character is received. The timer expires with a VPLUS intrinsic error if VPLUS block mode is being used. The timer expires with a file system error (FSERR 27) if any other type of block mode is being used.
- System break is sent from the terminal (and system break is not disabled). All read data is lost, but the read is reposted when the user resumes the application.
- The subsystem break character is sent from the terminal while subsystem break is enabled. All read data is returned to the sender. Subsystem break is not recognized in binary mode.
- The DTC encounters a parity error while parity is enabled. Parity is not recognized in binary mode.

End of File Indicators

If a program is reading from one of the standard input files, certain conditions will cause an end-of-file to occur and the read will terminate with a condition code of CCG (>>). On $STDIN an end-of-file occurs if a colon (:) is encountered in column 1 of the record being read. On $STDINX the end-of-file occurs only if the first four characters of the record are **:EOF**. Once your program encounters an end-of-file condition on a standard file, any subsequent read requests issued by the program against the same file will also encounter an end-of-file. For this reason, your program should check for the end-of-file condition as the first step in reading input from a device. If one is found, the current file must be closed, because a read will not succeed.

Using FREAD

Once your program has successfully opened a file on a terminal device, with read access specified, and a valid file number is obtained as a result of the file open call, you can use the FREAD intrinsic to read data from the device. Make sure to issue any calls to affect the way the read will take place (input mode, data editing mode, etc.) before posting the read. The syntax of the FREAD call is as follows:

```
     I16                  I16V     UDS      I16V
transfercount:=FREAD(filenum,buffer,length);
```

FREAD accesses the devicefile through its file number (**filenum**) and the data input from the device is transferred to **buffer** when one of the read termination conditions occurs.

The byte count is specified in the **length** parameter of the FREAD call. The intrinsic allows you to specify length as either a positive value representing halfwords or a negative value representing bytes.

However, for asynchronous devices, the actual data transfer always occurs on a byte-by-byte basis. If the number of bytes input reaches the value specified, the read will terminate on byte count.

For asynchronous terminals, the maximum supported read length is four kilobytes (4 KB) in standard or transparent editing mode and 128 bytes in binary editing mode.

If *transfercount* is used, the actual number of bytes transferred as a result of the read will be returned as its value. If you specified NOWAIT I/O in the FOPEN or HPFOPEN call, however, this value will be 0 and the actual byte count will be returned in the IOWAIT or IODONTWAIT call. See Chapter 8, “Intrinsics Reference,” of this manual for more information on IOWAIT and IODONTWAIT.

The program fragment shown in Figure 3-2 illustrates the use of the FREAD intrinsic. NOWAIT I/O is not used in this example. A file is opened as `$STDIN` with 80 byte ASCII records specified and the file number is returned in the *fileid_in* variable. A read is then posted against this file using the FREAD intrinsic to read up to 80 bytes and store the data as the value of `read_buffer`. Note that `read_buffer` is declared as a string large enough to hold up to the maximum amount of data that could be transferred in a single FREAD call.

Figure 3-2 Illustration of the FREAD Intrinsic

```
{ Demonstrate the use of the FREAD intrinsic }

program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  input_filenum     : shortint;
  num_bytes_read    : shortint;
  read_buffer       : string[80];

function FOPEN : shortint;  intrinsic;
function FREAD : shortint;  intrinsic;


begin

{--------------------------------}
{ Open $STDIN for read access... }
{--------------------------------}
input_filenum := FOPEN(              ,     {Formal designator not specified}
                       octal('444') ,     {Foptions = ASCII, $STDIN, CCTL}
                       0            ,     {Aoptions = Read access}
                       -80               {Recsize = 80 bytes}
                      );

if (ccode <> CCE) then
  writeln('FOPEN failed.');

{---------------------------------------}
{ Read up to 80 bytes from the terminal. }
{ Store read data in read_buffer and     }
{ store number of bytes read in          }
{ num_bytes_read.                        }
{---------------------------------------}
num_bytes_read := FREAD( input_filenum ,   {Filenum for $STDIN}
                         read_buffer   ,   {Local buffer for read data}
                         -80               {Up to 80 bytes}
                        );

if (ccode <> CCE) then
  writeln('FREAD failed.');

end.
```

Timing a Read

On MPE/iX systems every read that occurs is timed; there is never a need to turn a timer on to obtain information about the length of a read. However, the result of the timer is not actually returned to your program unless you explicitly issue an FCONTROL call, using 22 as the value of the **controlcode** parameter. This FCONTROL call returns the time used for the immediately preceding read as the value of **param**.

The time of the read is returned in hundredths of a second, up to a limit of 655.35 seconds. This limit is imposed by the 16-bit capacity of **param**. If the read being timed took longer than 655.35 seconds to complete, the FCONTROL call returns a CCG condition. A call to FCHECK would return an `FSERR 98, Read Timer Overflow`.

The code fragment shown in Figure 3-3 posts a read against *`fileid_in`* and issues a call to FCONTROL(22) immediately after the read to obtain the results of the read timer. After execution of the FCONTROL call, `read_time` will contain the results of the timer and the program will write this value to the standard output device. The code sample also checks for a CCE condition, which would indicate that the FCONTROL had succeeded.

Figure 3-3 Obtaining the Result of the Read Timer

```
{ Demonstrate how to check the time of a read }

program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  input_filenum     : shortint;
  num_bytes_read    : shortint;
  read_buffer       : string[80];
  read_time         : shortint;

procedure FCONTROL;          intrinsic;
function  FOPEN : shortint;  intrinsic;
function  FREAD : shortint;  intrinsic;


begin

input_filenum := FOPEN(              ,     {Formal designator not specified}
                       octal('444') ,     {Foptions = ASCII, $STDIN, CCTL}
                       0             ,     {Aoptions = Read access}
                       -80                 {Recsize = 80 bytes}
                      );

if (ccode <> CCE) then
  writeln('FOPEN call failed.');

num_bytes_read := FREAD( input_filenum ,   {Filenum for $STDIN}
                         read_buffer   ,   {Local buffer for read data}
                         -80               {Up to 80 bytes}
                        );

if (ccode <> CCE) then
  writeln('FREAD call failed.');

{---------------------------------------}
{ Use FCONTROL 22 to obtain time of last }
{ read.                                 }
{---------------------------------------}
FCONTROL( input_filenum ,                  {Filenum for $STDIN}
          22            ,                  {Get read time}
          read_time                        {Put read time here}
        );

if (ccode <> CCE) then
  writeln(output, 'FCONTROL failed.')
else
  writeln(output, 'Time of last read was ', read_time, ' centiseconds.');

end.
```

Setting a Read Time Limit

FCONTROL can also be used to set a time limit on a succeeding read, by specifying 4 as the value of **controlcode** and specifying a time limit, in seconds, through **param** (up to 65535 seconds). If the read immediately following the call takes more than the time specified, the read will terminate in error and any data already received will be flushed. The read limit remains in effect only for the next read. You must issue a separate FCONTROL(4) call prior to every read on which you want to impose a time limit.

There are many circumstances in which it is advisable to set a timeout value. Timeouts should always be used with devices that operate without a dedicated attendant, to prevent hangs that could occur if the device were to fail in some way.

Timeouts are effective in all data editing modes. This makes it possible to terminate even a binary mode read should it exceed the specified time limit. During block mode reads, however, the timer is halted upon receipt of the DC2 character and the block mode read timer takes over.

When a read terminates as the result of exceeding the timeout set by FCONTROL(4), the read fails with a CCL condition and all data is lost. A call to FCHECK returns an `FSERR 22, Software Timeout`.

The code segment shown in Figure 3-4 sets a 30-second time limit for the next read posted against *`fileid_in`*. The sample includes an error checking routine which verifies whether or not the read terminated normally. If not, the program calls FCHECK. If FCHECK returns an `FSERR 22`, the program will inform the user that a read timeout occurred.

Figure 3-4 Opening a Read Port and a Write Port

```
{ Demonstrate how to post a timed read }

program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  errornum          : shortint;
  input_filenum     : shortint;
  num_bytes_read    : shortint;
  read_buffer       : string[80];
  timeout           : shortint;

procedure FCONTROL;          intrinsic;
procedure FCHECK;            intrinsic;
function  FOPEN : shortint;  intrinsic;
function  FREAD : shortint;  intrinsic;


begin

input_filenum := FOPEN(              ,     {Formal designator not specified}
                       octal('444') ,     {Foptions = ASCII, $STDIN, CTL}
                       0             ,     {Aoptions = Read access}
                       -80                 {Recsize = 80 bytes}
                      );

if (ccode <> CCE) then
  writeln('FOPEN call failed.');

timeout := 30;                            {Read time limit will be 30 seconds}
FCONTROL( input_filenum ,                 {Filenum for $STDIN}
          4             ,                 {4 = Set time limit for next read}
          timeout                         {Buffer containing read time limit}
        );

if (ccode <> CCE) then
  writeln('FCONTROL call failed.');

num_bytes_read := FREAD( input_filenum ,  {Filenum for $STDIN}
                         read_buffer   ,  {Local buffer for read data}
                         -80              {Up to 80 bytes}
                        );

if (ccode = CCE) then
  writeln('Read completed normally.')
else
  begin

  FCHECK (input_filenum, errornum);

  if (errornum = 22) then
    writeln('Read completed on timeout.')
  else
    writeln('FREAD failed.');

  end;

end.
```

Using READ or READX

The READ and READX intrinsics provide a simple method for reading from the standard input device. You do not need to know the file number when you use these intrinsics. Except for the intrinsic name, the syntax is the same for READ and READX. The syntax of the READ intrinsic is shown below:

```
    I16              CA          I16V
transfercount:=READ(message,expectedlength);
```

The maximum length of the read is passed in **expectedlength** and the call returns the actual length of the read in *transfercount*, if *transfercount* is used. The read data is returned in message.

Both READ and READX transfer an ASCII string into a character array in your program. However, READ transfers data from $STDIN, while READX transfers data from $STDINX. READ also differs from READX in the way an end-of-file is interpreted. READ recognizes any colon (:) found in column 1 as an end-of-file, while READX interprets only **:EOD** as an end-of-file.

Because there is no file number associated with READ or READX, it is not possible to use the FCHECK intrinsic to determine what took place in the event of an error. Nor can you use the **:FILE** command to redirect input, since the input always comes from the standard input files opened by the session at logon time.

Because of the limitations inherent in the use of READ and READX, it is recommended that you use these intrinsics only for temporary programs as a quick way to obtain input from a terminal. A better programming practice for permanent programs would be to use FOPEN or HPFOPEN to open $STDIN or $STDINX and then issue FREAD calls against these files.

Writing to Asynchronous Devices

The most common ways to write information to an asynchronous devicefile are provided by the FWRITE and PRINT intrinsics. FWRITE is the most versatile of these and can be used to write information to any device. PRINT is more limited in its usage, but does not require the use of a file number.

Using FWRITE

FWRITE transfers a record of data to a file on any device. You must first open a file on the device with write access specified and obtain a valid file number. The file number is then used by the FWRITE call to access the file. The syntax of FWRITE is as follows:

```
        I16V    UDS   I16V      U16V
FWRITE(filenum,buffer,length,controlcode);
```

The **filenum** parameter passes the file number of the file to which data will be written. The record to be written is passed in the **buffer** parameter, with the length of the data specified in the **length** parameter. A positive or negative value is used to indicate whether **length** is halfwords or bytes. You should be aware, however, that for asynchronous devices, physical data transfer will always occur on a byte-by-byte basis. Specifying halfwords will result in an even number of bytes being output.

If **length** is 0, no data transfer will occur. You might use FWRITE with **length** specified as 0 to set line spacing (carriage control) at the device before any data is output.

Carriage control is passed through the **controlcode** parameter. The carriage control specification can be 0, 1, or one of the values designated as **controlcode** values for MPE/iX. Table 8-1, “Intrinsics and Device Control,” on page 152 lists many of the carriage control directive values most commonly used with asynchronous devices. For a complete list of carriage control directive codes that can be used with the FWRITE intrinsic you should see the *MPE/iX Intrinsics Reference Manual*.

If **controlcode** is 0, the full record of data will be transferred to the devicefile, up to a maximum of 132 characters per printed line, and single spacing will be applied.

If **controlcode** is 1, the first character of the data will be interpreted as space control and will be suppressed when the data is printed. Any of the values supported as carriage control directives can be used as the first character when this method is used.

The carriage control directive values can also be applied directly through the **controlcode** parameter. Simply use one of these values instead of a 0 or 1.

When writing to a terminal, a **controlcode** value of %320 is often used. This value specifies that no carriage control will be sent to the terminal and the next character printed by the next write will physically follow the data written by the current write. You might use a **controlcode** of %320 when prompting for user input at a terminal. No carriage return or linefeed will follow the write and the cursor will be positioned next to the prompt.

Figure 3-5 illustrates the use of the FWRITE intrinsic. This code fragment opens a terminal file as $STDLIST with write access specified. The prompt to be written to the terminal is placed into buffer and the variable cctl is specified as %320. FWRITE is then called using these values. The result will be that the prompt “Enter Your Name” will be written to the device and the cursor will be placed next to the prompt.

Your program should check condition codes after each FWRITE; however, you should be aware that this intrinsic is completed logically. This means that once data is passed to an output buffer the write is considered to have completed successfully. If the output device has suspended operation, due to flow control or some other reason, the actual write may not have occurred. There is no programmatic way to verify the physical output of data to a device.

Figure 3-5 Illustration of the FWRITE Intrinsic

```
{ Demonstrate the use of the FWRITE intrinsic }

program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  output_filenum    : shortint;
  write_buffer      : string[80];

function  FOPEN : shortint;  intrinsic;
procedure FWRITE;            intrinsic;


begin

output_filenum := FOPEN(              ,    {Formal designator not specified}
                        octal('414') ,    {Foptions = ASCII, $STDLIST, CCTL}
                        1             ,    {Aoptions = Write access}
                        -80                {Recsize = 80 bytes}
                       );

if (ccode <> CCE) then
  writeln('FOPEN call failed.');

write_buffer := 'Enter Your Name:';        {Fill the write buffer}

FWRITE( output_filenum         ,           {File number for $STDLIST}
        write_buffer           ,           {Buffer containing write data}
        -strlen( write_buffer) ,           {Length of buffer in BYTES}
        octal('320')                       {Carriage control (%320 = nospace)}
      );

if (ccode <> CCE) then
  writeln('FWRITE call failed.');

{---------------------------------------}
{ The carriage control code of %320 has   }
{ left the cursor on the same line as the }
{ write data, allowing the user to enter  }
{ data following the prompt (if a read is }
{ posted).                                }
{---------------------------------------}

end.
```

Sending Escape Sequences

You can also use FWRITE to send escape sequences to an asynchronous device. Escape sequences can be used to alter the physical settings of a device. For example, on most supported terminals you can use escape sequences to set terminal straps, control the placement of the cursor on a terminal screen, and set margins. You should consult the manual for your device for a summary of useful escape sequences.

In the following example the variable *esc_seq* contains the string *'ESC&s1D'*. This escape sequence selects page block mode at the terminal (opens the "D" strap). The escape sequence can be sent to the device with the following FWRITE call:

```
FWRITE(file_num, esc_seq, -5, octal('320'));
```

The **length** parameter is specified as 5 bytes, the length of the escape sequence (ESC is one byte). Carriage control is set to %320, so that no carriage return or linefeed are sent. As a result of this call, the terminal on which the file specified by `file_num` is opened will be strapped for page block mode.

Using PRINT

In much the same way that READ and READX provide a quick method for reading from the standard input device, PRINT provides a quick method for writing to the standard output device. PRINT sends a string of ASCII characters to `$STDLIST`. The syntax of the call is as follows:

```
       CAI     16V     I16V
PRINT(message,length,controlcode);
```

The **message** parameter is a character array that holds the string to be written to the device. It must begin on a halfword boundary, as must all character arrays.

The **length** parameter specifies the length of the character string. If the length of the message exceeds the configured record length of the terminal, successive records will be written.

The **controlcode** parameter specifies the carriage control to be applied when the message is output. Either a 0, 1, or one of the valid carriage control directives should be passed in this parameter. See the discussion of the FWRITE intrinsic for more information on **controlcode** values.

Like READ and READX, PRINT has a number of limitations on the way it can be used. Because no file number is available, **`:FILE`** commands are not useful, nor is it possible to use the FCHECK intrinsic to check error conditions. For permanent programs it is recommended that FWRITE be used instead.

Altering Terminal Deficefiles

Many of the asynchronous device control functions performed most commonly are accomplished through the use of the FCONTROL intrinsic. FCONTROL has often been referred to as the workhorse of the file system intrinsics. The examples that follow illustrate how FCONTROL can be used to alter the characteristics of data transmission to and from a terminal, or to alter device control settings. The syntax of FCONTROL is shown below:

```
          I16V      I16V      *
FCONTROL(filenum,controlcode,param);
```

FCONTROL accesses specific devicefiles through their file numbers, as specified in the **filenum** parameter. The function performed by a particular call to FCONTROL is determined by the **controlcode** parameter, as is the meaning and use of **param**. For some **controlcode** values **param** has no meaning; in these cases **param** must be coded as a dummy value to satisfy the requirements of the intrinsic.

See Chapter 8, “Intrinsics Reference,” for a list of the **controlcode** values most useful for the control of asynchronous devices and for information on how **param** is used for each of the **controlcode** values described.

Completion status for each FCONTROL call is indicated through condition codes. See Chapter 2, “Controlling Asynchronous Devices Programmatically,” for more information on condition codes.

FCONTROL must be called before any data transfer activity in order to have any effect on how that activity occurs. In many cases you will need to use a series of FCONTROL calls to set all of the characteristics desired for a particular transaction. The code fragments that follow illustrate the use of a number of FCONTROL functions.

NOTE

An alternative to the FCONTROL intrinsic is provided by the FDEVICECONTROL intrinsic, which is described in this manual. The code segments shown in this chapter, however, demonstrate only the use of FCONTROL.

Specifying an AEOR Character

Specifying an AEOR Character Figure 3-6 shows how FCONTROL may be used to alter the way in which a read issued against a terminal can be terminated. A **controlcode** value of 25 is used, which designates that a character, specified through the call, is to be accepted as an Additional End-of-Record (AEOR) character. In this case, an asterisk is designated as the AEOR character.

Once an AEOR character is designated, any non-binary mode read issued for that devicefile can be terminated by entry of the AEOR character, as well as in the normal ways, until the device is closed, or until the AEOR character is disabled. You can disable the AEOR character by calling FCONTROL(25) again, using a 0 as the new value for AEOR.

A read terminated by means of an AEOR does not end normally. A file system error is returned, indicating that the AEOR character was used. The AEOR character is transmitted with the rest of the read data and is added to the byte count. No carriage return or linefeed is sent to the terminal. See the discussion of FCONTROL(25) in Chapter 8, “Intrinsics Reference,” for additional details on this use of the FCONTROL intrinsic.

Figure 3-6 Obtaining the Result of the Read Timer

```
program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  aeor_char         : shortint;
  input_filenum     : shortint;

procedure FCONTROL;          intrinsic;
function  FOPEN : shortint;  intrinsic;


begin

input_filenum := FOPEN(              ,     {Formal designator not specified}
                       octal('444') ,     {Foptions = ASCII, $STDIN, CCTL}
                       0            ,     {Aoptions = Read access}
                       -80                {Recsize = 80 bytes}
                      );

if (ccode <> CCE) then
  writeln('FOPEN failed.');


{---------------------------------------}
{ Enable AEOR character (set it to '*')  }
{---------------------------------------}
aeor_char := ord('*');                    {AEOR char will be '*'}

FCONTROL( input_filenum ,                 {Filenum for $STDIN}
          25            ,                 {25 = Define AEOR character}
          aeor_char                       {Buffer containing character num}
        );

if (ccode <> CCE) then
  writeln('FCONTROL failed.');


{---------------------------------------}
{ Disable AEOR by setting it to 0...     }
{---------------------------------------}
aeor_char := 0;                           {AEOR will be turned off}

FCONTROL( input_filenum ,                 {Filenum for $STDIN}
          25            ,                 {25 = Define AEOR character}
          aeor_char                       {Buffer containing character num}
        );

if (ccode <> CCE) then
  writeln('FCONTROL failed.');

end.
```

Altering Device Control Settings

The code fragment presented in Figure 3-7 shows how a series of FCONTROL calls may be used to perform various device control functions. The end result of such a sequence of FCONTROL calls should be a devicefile whose characteristics are as required for a specific application.

The first call in the example, to FCONTROL(40), returns device information to the calling program as the value of the *line_speed* variable. This call is used to determine the current line speed of the device. Examples of when such a call may be used are to verify a particular setting for an application, or to obtain and store a current value so that a setting can be altered and later reset to its original value.

The next call, to FCONTROL(11), changes the line speed of the device to a new value. The value used in the call must be a supported line speed, entered as characters per second (240 characters per second is 2400 bps).

The next two FCONTROL calls work together to set a desired parity and enable parity checking. FCONTROL(36) is used in this case to define odd parity as the parity to be used if parity is enabled. Note that setting parity has no effect until parity is enabled.

Parity is enabled through the subsequent call to FCONTROL(24). In the FCONTROL(24) call, the third parameter has no meaning. A dummy value must be specified, however, to satisfy the needs of the FCONTROL intrinsic, as well as the needs of the Pascal compiler.

Figure 3-7 Opening a Read Port and a Write Port

```
{ Use FCONTROL to alter baud rate and parity settings }

program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  dummy             : shortint;
  input_filenum     : shortint;
  new_line_speed    : shortint;
  old_line_speed    : shortint;
  output_filenum    : shortint;
  parity_setting    : shortint;

procedure FCONTROL;          intrinsic;


begin

{---------------------------------------}
{ Input & output files should be opened  }
{ before executing the following code... }
{---------------------------------------}

{-------------------------------------------------------------}
{ Use FCONTROL(40) to read the current line speed setting... }
{-------------------------------------------------------------}
FCONTROL( output_filenum ,                {Filenum for $STDLIST}
          40             ,                {40 = Determine speed setting}
          old_line_speed                  {Current speed is returned here}
        );

if (ccode <> CCE) then
  writeln('FCONTROL(40) failed.');


{-------------------------------------------------------------}
{ Use FCONTROL(11) to set new line speed...                   }
{-------------------------------------------------------------}
new_line_speed := 240;                    {240 characters per second}

FCONTROL( output_filenum ,                {Filenum for $STDLIST}
          11             ,                {11 = Change device line speed}
          new_line_speed                  {Buffer containing new line speed}
        );

if (ccode <> CCE) then
  writeln('FCONTROL(11) failed.');


{-------------------------------------------------------------}
{ Use FCONTROL(36) to set parity to ODD...                    }
{-------------------------------------------------------------}
parity_setting := 3;                      {3 = Odd parity}

FCONTROL( input_filenum  ,                {Filenum for $STDIN}
          36             ,                {36 = Define parity setting}
          parity_setting                  {Buffer containing parity setting}
        );

if (ccode <> CCE) then
  writeln('FCONTROL(36) failed.');


{-------------------------------------------------------------}
{ Use FCONTROL(24) to enable parity generation & checking... }
{-------------------------------------------------------------}
FCONTROL( input_filenum  ,                {Filenum for $STDIN}
          24             ,                {24 = Enable parity checking}
          dummy                           {Not used}
        );

if (ccode <> CCE) then
  writeln('FCONTROL(24) failed.');
```

Setting Transparent Editing Mode

The FCONTROL intrinsic is also commonly used to set the data editing mode at a terminal. As explained earlier in this chapter, the data editing mode controls which characters are considered to have special meaning when they are encountered in the data stream.

In standard editing mode, all special characters recognized by MPE/iX apply and are acted on when sent from a terminal. However, there are a number of reasons why you might want to suspend the special processing associated with some of these characters. Your application may be using one or more of these characters for a different purpose. Or you may be using an unsupported device which transmits signals that cause problems in standard editing mode.

One solution is to enable binary editing mode, which performs no special character processing and allows all characters to pass through as data. However, the use of binary mode severely limits the transmission and terminal control options available to you.

Transparent editing mode provides a data editing level between standard and binary editing modes. Because a limited number of special characters retain their meaning in transparent mode, there is much more flexibility in how a data transfer may take place than there is when binary mode is used. For example, you can perform block mode reads while in transparent editing mode, because the block mode control characters (DC1 and DC2) retain their special meanings. (DC2 only has special meaning when received as the first byte of data, however.) Transparent mode also allows you to use parity checking and recognizes and responds to the AEOR character, if one has been designated.

If transparent editing mode is enabled for any file, it is also in effect for any other files opened on the same device, regardless of whether or not it was specifically enabled for those files. Also, any FCLOSE issued against the device will return the device to standard editing mode.

Subsystem break processing is also available in transparent editing mode. The character that will be recognized as the subsystem break character is passed in the intrinsic call. Any character not otherwise recognized as a special character can be used, including the normal subsystem break character, **[CTRL]Y**. If a 0 is specified, subsystem break will be disabled. You should note, however, that simply designating a subsystem break character does not enable subsystem break processing.

You must still enable subsystem break response and arm the **[CTRL]Y** trap before the subsystem break function can be invoked.

Transparent editing mode will be overridden if you should enable binary editing mode at the same device. However, when you leave binary mode, the device will still be operating in transparent mode. You

must explicitly disable transparent editing, either by issuing an FCLOSE against the device, or by calling the intrinsic used to set transparent mode again with both the EOR and subsystem break characters set to null.

The code fragment shown in Figure 3-8 provides an example of enabling transparent editing mode through FCONTROL(41). This FCONTROL uses **param** to set new values for the EOR character (in the low order byte) and the subsystem break character (in the high order byte). In the example, the *bkeor* variable is set to 6433, to specify that an exclamation point (!) is the new EOR character and **[CTRL]Y** is the subsystem break character. Any subsequent read will recognize and treat these characters accordingly and strip them from the input stream.

A similar code fragment, setting *bkeor* to 0, could be used to disable transparent editing mode.

See Chapter 4, “Using FDEVICECONTROL,” for an example of using FDEVICECONTROL to enable transparent editing mode.

Figure 3-8 Opening a Read Port and a Write Port

```
{ Use FCONTROL to set transparent editing mode }

program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  input_filenum      : shortint;
  output_filenum     : shortint;
  transparent_bk_eor : shortint;

procedure FCONTROL;          intrinsic;


begin

{---------------------------------------}
{ Input & output files should be opened  }
{ before executing the following code... }
{---------------------------------------}

{-------------------------------------------------------------}
{ Use FCONTROL(41) to set transparent editing mode...         }
{ EOR character will be '!', and Subsystem Break key will    }
{ be ^Y.                                                      }
{-------------------------------------------------------------}
transparent_bk_eor := ord('!') +          {'!' is EOR in low byte}
                      25 * 256;           {^Y is SSBK in high byte}

FCONTROL( input_filenum      ,            {Filenum for $STDIN}
          41                 ,            {41 = Set transparent mode}
          transparent_bk_eor              {Buffer passing SSBK & EOR chars}
        );

if (ccode <> CCE) then
  writeln('FCONTROL(41) failed.');


{-------------------------------------------------------------}
{ If FCONTROL worked, device is now in transparent mode...    }
{-------------------------------------------------------------}

end.
```

Setting Binary Editing Mode

In some cases you may need to disable all special character processing and pass everything through as data. Binary editing mode allows you to transfer information in this manner.

Binary editing mode treats all characters as data and passes all eight bits of every character through without performing any special operations. Fairly obviously, this restricts the way in which data transfers can take place. You cannot use parity checking in binary mode, since all eight bits of each character are treated as data. Block mode transfers are also impossible, since no special handshake characters are recognized.

The only way that a binary read will terminate without returning an error is by reaching the byte count specified in the FREAD, READ, or READX call that initiated the read. Read timeouts will terminate a binary read in error and no data will be transferred. It is a good idea to specify a read timeout value when posting binary reads to avoid problems that can occur if the read does not reach the specified byte count.

Binary mode is enabled by calling FCONTROL with a **controlcode** value of 27 and disabled through FCONTROL with a **controlcode** value of 26. With either of these FCONTROL calls the third parameter has no meaning and should be invoked with a dummy value of 0.

Figure 3-9 provides an example of enabling binary editing mode in a program. You should note several features of this code fragment, as explained below, to understand how to use binary mode for your own programming needs.

Note first that a call to FCONTROL(13) precedes any call to enable binary mode. FCONTROL(13) disables character echo at the terminal. If character echo is not disabled prior to transferring data in binary mode, any ASCII DC3 (XOFF) character in the data (CONTROL S typed at the terminal) would be echoed by the DTC back to the terminal and it would appear to the terminal that the DTC had been XOFFed. This would cause the terminal to hang, since the terminal would suspend transmission to await on XON signal that would never arrive.

An alternative to disabling character echo is to turn XON/XOFF flow control off at the terminal. This is a dangerous alternative, however, since disabling flow control may result in write data being lost at the terminal.

The next feature of this program you should note is that separate calls to FCONTROL(27) are posted against a read port and a write port on the device. FCONTROL(27) is one of several terminal controls that affect the data transfer characteristics of a specific file, rather than of the device itself. This makes it necessary to explicitly specify binary mode for both the read port and the write port opened on the device.

Finally, note that in the example an FREAD call specifying a read of 0 bytes is posted against the device immediately following the FCONTROL calls. Only after the 0 byte read completes successfully does the program notify the user that the port is in binary mode. It is actually the read, not the FCONTROL call, that causes the controlling software to place the device driver in binary mode. (This is only true when you are setting binary mode. Most FCONTROLs take effect at the time the FCONTROL call is issued.)

If you are setting binary mode for reads only this is not a consideration, since the first read posted will cause binary mode to be set. If, however, you are setting binary mode for both a read port and a write port, you need to post a read of 0 bytes before assuming that binary mode is set.

Figure 3-9 Opening a Read Port and a Write Port

```
{ Use FCONTROL to set binary mode }

program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  bytes_read         : shortint;
  dummy              : shortint;
  input_filenum      : shortint;
  output_filenum     : shortint;
  read_buffer        : packed array [1..80] of char;

procedure FCONTROL;          intrinsic;
function  FREAD : shortint;  intrinsic;


begin

{---------------------------------------}
{ Input & output files should be opened  }
{ before executing the following code... }
{---------------------------------------}

{-------------------------------------------------------------}
{ Disable echo from DTC...                                    }
{-------------------------------------------------------------}
FCONTROL( input_filenum      ,            {Filenum for $STDIN}
          13                 ,            {13 = Turn echo off}
          dummy
        );

if (ccode <> CCE) then
  writeln('FCONTROL(13) failed.');

{-------------------------------------------------------------}
{ Enable binary mode on input file (for reads)...             }
{-------------------------------------------------------------}
FCONTROL( input_filenum      ,            {Filenum for $STDIN}
          27                 ,            {27 = Enable binary mode}
          dummy
        );

if (ccode <> CCE) then
  writeln('FCONTROL(27) failed.');

{-------------------------------------------------------------}
{ Enable binary mode on output file (for writes)...           }
{-------------------------------------------------------------}
FCONTROL( output_filenum     ,            {Filenum for $STDLIST}
          27                 ,            {27 = Enable binary mode}
          dummy
        );

if (ccode <> CCE) then
  writeln('FCONTROL(27) failed.');

{-------------------------------------------------------------}
{ Send a zero-byte read to force driver software to flush     }
{ binary mode FCONTROLs out to the DTC immediately...         }
{-------------------------------------------------------------}
bytes_read := FREAD( input_filenum ,       {Filenum for $STDIN}
                     read_buffer   ,       {Buffer for read data}
                     0                     {Max read length is zero}
                   );

if (ccode <> CCE) then
  writeln('FREAD failed.');


{-------------------------------------------------------------}
{ If all FCONTROLs worked, device is now in binary mode...    }
{-------------------------------------------------------------}

end.
```

Closing Files

The FCLOSE intrinsic is used to terminate access to a file. It applies to files on all devices and frees any resources that were assigned at the time the file was opened. If more than one file is opened against a device, the device itself is not deallocated until an FCLOSE is issued against every file opened on that device by your program. The syntax of the FCLOSE call is as follows:

```
        I16V       I16V         I16V
FCLOSE(filenum,disposition,securitycode);
```

The **filenum** parameter contains the file number of the file to be closed. The **disposition** and **securitycode** parameters provide facilities for determining file disposition and security levels for files on devices other than terminals and printers. They are not meaningful for asynchronous devices and should be set to 0.

Some of the device settings that may have been altered by FCONTROL (or FDEVICECONTROL) calls issued by your program will be reset to their default settings at FCLOSE. However, this is not true for all device settings that may have been altered. You are responsible for making sure that your program does not adversely affect the way the device will act after your program terminates. It is good programming practice to reset any device characteristics your program alters and to issue an FCLOSE for every file your program opens.

Should your program fail to issue an FCLOSE for a file it has opened, the file system will automatically issue an FCLOSE for that file when the program terminates. When the last file opened against a device is closed, all device characteristics are returned to their configured settings. If there is a session associated with the device, the last file is not closed until the session is terminated.

In Figure 3-10, two FCLOSE calls are issued to close ports previously opened for reading and writing to a terminal. Condition codes are checked after each FCLOSE call through the Pascal ccode function, which will return a value of CCE if no errors occur during the call, or of CCL or CCG if errors do occur.

Figure 3-10 Opening a Read Port and a Write Port

```
{ Demonstrate the use of the FCLOSE intrinsic }

program test;

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

procedure FCLOSE;  intrinsic;


{------------------------------------------------------------------------------}
{ The CLOSEPORT function tries to close the two device files passed to it.     }
{ If successful, it returns TRUE.  If an error occurs, it will return FALSE.  }
{------------------------------------------------------------------------------}
function CLOSEPORT (input_filenum  : shortint;   {File number for read port}
                    output_filenum : shortint    {File number for write port}
                   ) : boolean;                  {Returns TRUE if successful}
begin

closeport := TRUE;                          {Assume success for now}

{---------------------------------}
{ Close read port...              }
{---------------------------------}
FCLOSE( input_filenum ,                     {Filenum for read port}
        0             ,                     {Disposition}
        0                                   {Security}
      );

if (ccode <> CCE) then
  closeport := FALSE;                       {Failed to close read port}

{---------------------------------}
{ Close write port...             }
{---------------------------------}
FCLOSE( output_filenum ,                    {Filenum for write port}
        0              ,                    {Disposition}
        0                                   {Security}
      );

if (ccode <> CCE) then
  closeport := FALSE;                       {Failed to close write port}

end;


begin
end.
```

4 Using FDEVICECONTROL

Many of the asynchronous device control functions that can be performed through FCONTROL can be performed through FDEVICECONTROL as well. In addition, FDEVICECONTROL allows you to perform a number of device control functions that are not available through use of FCONTROL. Chapter 3, “Common Device Control Functions,” in this manual provides some practical examples of how you can use FCONTROL in a program to control asynchronous devices.

This chapter describes how you might use FDEVICECONTROL to perform additional device control functions. It includes program fragments to illustrate the use of the intrinsic. The text which accompanies the examples will contrast the use of the FDEVICECONTROL and FCONTROL.

Examples in this chapter illustrate using FDEVICECONTROL to perform the following functions:

- Enable/disable transparent editing mode
- Specify a new end-of-record character.
- Change the backspace character and backspace response at a terminal.
- Enable/Disable device XON/XOFF.
- Combine enable/disable escape sequence read termination function with the enable/disable suppress echo of read termination characters function.
- Specify the number of stop bits.

Refer to Chapter 8, “Intrinsics Reference,” for a complete summary of the functions available through FDEVICECONTROL and for full explanations of its syntax and parameter values.

Syntax Description

While FDEVICECONTROL performs some of the same device control functions performed by FCONTROL, the two intrinsics look very different from each other when you use them in a program. FCONTROL uses just three parameters, through which you specify a devicefile, define the control operation to be performed, and, depending on the control operation, pass or receive specific information that may be required for that operation.

FDEVICECONTROL requires the use of a greater number of parameter values. For operations that can be performed by either FCONTROL or FDEVICECONTROL, it is easier to use FCONTROL. The following is the syntax for FDEVICECONTROL:

```
                I16V     UDS      I16V     I16V
FDEVICECONTROL(filenum,buffer,length,controlcode,
                I16V   U16V    U16
                 parm1,parm2,fserrorcode);
```

FDEVICECONTROL is used for a variety of file control operations, each of which is appropriate for controlling a specific type of devicefile.

The **controlcode** parameter determines the control operation that will be performed. The only control operation which effects asynchronous devicefiles is **controlcode** value 192. It sends device control directives to an asynchronous device.

Refer to Table 8-7 for a list of other valid **controlcode** values and the operations they perform. For more information on these operations refer to the *MPE/iX Intrinsics Reference Manuals*.

When **controlcode** is 192, **parm1** determines the specific control directive to be sent to the devicefile. **parm1** must contain one of the control directive values specified in Chapter 4, “Using FDEVICECONTROL.”

parm2 determines the Read/Write access to be imposed on the directive. The Read/Write access setting must be valid for the control directive specified in **parm1**. Set **parm2** to 1 for Read access, 2 for Write access, or 3 for both Read and Write access.

For example, if you were to call FDEVICECONTROL with **controlcode** set to 192 and **parm1** set to 57, which is the control directive used to obtain the subsystem break character, you would need to set **parm2** to 1, since only Read access is allowed for that control directive.

If, however, **parm1** were set to 36, which is the control directive used to define the backspace character, you could set **parm2** to 1, 2, or 3. In this case, setting **parm2** to 1 would return the current backspace character

to your program, while setting it to 2 would actually change the backspace character to whatever new character you pass in the call (through the **buffer** parameter). Setting **parm2** to 3 would both return the previous value and change the backspace character to the new value passed in the call.

Some control directives do not permit all possible values of **parm2** to be used with them. Table 8-8 lists the allowable **parm2** access values for each of the valid control directives that may be specified in **parm1**.

The **filenum** parameter specifies the file number, as obtained through a previous call to FOPEN or HPFOPEN, of the devicefile on which the control operation is to be performed.

The **buffer** parameter passes and returns the data associated with the control directive being performed. For example, if the control directive specified in **parm1** is 3, which sets the line speed for a device, **buffer** must contain the new linespeed setting (and will return the previous speed setting if **parm2** is set to Read/Write access).

The **length** parameter sets the length of **buffer**. Positive values indicate the length in half words. Negative values indicate the length in bytes. Usually **length** will be set to 1, indicating that **buffer** is 1 half word long.

The **fserrorcode** parameter returns error information to your program. If no error occurs during execution, **fserrorcode** contains a 0. If an error occurs, **fserrorcode** returns the file system error code corresponding to that error. You will need to call FCHECK for more information about the error that occurred.

Examples

The examples that follow illustrate using FDEVICECONTROL to perform a number of control operations on asynchronous devicefiles. The first example demonstrates how FDEVICECONTROL is used to set transparent editing mode at a terminal. A similar example, using FCONTROL for the same purpose, is included in Chapter 3, "Common Device Control Functions." These two examples allow you to contrast the use of the two intrinsics.

The remaining examples, illustrated by the following figures, demonstrate the use of FDEVICECONTROL to perform functions that are not available through any other intrinsic.

Setting Transparent Editing Mode

In Figure 4-1 transparent editing mode is enabled at a terminal through the FDEVICECONTROL intrinsic. Transparent editing mode is also referred to as unedited mode. The parameters of the call specify how the control action is to take place.

A file number, obtained through a previous call to FOPEN or HPFOPEN issued against the device to be controlled, is contained in the **port** parameter. (The file open call is not shown in the example.) The **controlcode** is set to 192, indicating that this is a device control operation.

The **parm1** value is set to 15, which is the control directive used to set transparent editing mode. The **parm2** value is 2, specifying that only Write access is needed. This means that a value will be passed in the **buffer** parameter, but the previous value will not be returned.

When you use FCONTROL to enable transparent editing, you specify new subsystem break and EOR characters through one of the parameters of the call. You must also specify these characters when you enable transparent mode through FDEVICECONTROL. You pass the value corresponding to the characters you want to use through the **buffer** parameter. Any character not otherwise defined as a special character in transparent mode can be used.

The example shown here uses the Pascal/iX `ord` function to set the subsystem break and EOR characters through a variable called **value**. The high order bit of **value** contains a slash (/), which will be the new subsystem break character, and the low order bit contains an asterisk (*), which will be the new end-of-record character. This variable value is passed through the **buffer** parameter.

One other notable feature of the example is the way errors are handled. The example uses the **fserrorcode** parameter to provide information to the user in the event of an error by writing the file system error number to the standard output device.

Figure 4-1 FDEVICECONTROL to Enable Transparent Mode

```
{ Use FDEVICECONTROL to enable transparent editing mode }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 15;                                {Set/Clear Unedited Mode}
parm2 := 2;                                 {Write only access}
value := ord('/') * 256 +                   {Subsystem break = '/' (bits 0:8)}
         ord('*');                          {End of record   = '*' (bits 8:8)}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Buffer containing SSBK & EOR}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
               );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 15 failed with fserror ',fserror);

end.
```

You also call FDEVICECONTROL with a control directive of 15 to disable transparent editing mode. To return to standard editing, call FDEVICECONTROL with **parm1** set to 15, but specify 0 for both the subsystem break and EOR characters. Figure 4-2 provides an example of disabling transparent mode. Note that, in this example, **parm2** specifies both Read and Write access for the call. The subsystem break and EOR characters in effect before the call executed can therefore be identified and reported after the intrinsic completes successfully.

Figure 4-2 FDEVICECONTROL to Disable Transparent Mode

```
{ Use FDEVICECONTROL to clear transparent editing mode }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  old_eor        : char;
  old_ssbk       : char;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{                                          }
{ Also, this code assumes read port has    }
{ already been put into transparent mode... }
{-----------------------------------------}

parm1 := 15;                                {Set/Clear Unedited Mode}
parm2 := 3;                                 {Read & Write access}
value := 0;                                 {Disable Unedited mode}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Buffer containing SSBK & EOR}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (read & write)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 15 failed with fserror ', fserror)
else
  begin

  old_eor  := chr(value MOD 256);
  old_ssbk := chr(value DIV 256);
  writeln('Previous EOR  character was ', old_eor);
  writeln('Previous SSBK character was ', old_ssbk);

  end;

end.
```

Specifying an End-of-Record Character

You can change the character that will be recognized as the end-of-record character for a device by specifying a control directive of 39 and passing the desired character in **buffer**. This FDEVICECONTROL is effective in either standard and transparent editing mode. Note that the call does not alter the editing mode in effect for the device; it simply changes the character that will signal the end of a record for that device. This capability is not available through any FCONTROL function.

You might want to use this capability, for example, to make it possible for carriage returns to be passed in your data without enabling binary editing mode and without modifying the processing of any other special character. Only the end-of-record character is affected by this control directive.

A notable characteristic of this use of FDEVICECONTROL is that the value of the new end-of-record character must be passed in the high order byte of the **buffer** parameter. The low order byte is not used with this control directive. (Two other FDEVICECONTROL control directives, 40 and 41, also require that the value be passed in the high order byte.)

In Figure 4-3, FDEVICECONTROL is used to change the end-of-record character to a bracket (]). Again, the value of the character is specified through the Pascal/iX `ord` function and is then multiplied by 256 to place it in the high order byte. Read/Write access is set to Write only for the call, so the previous EOR character will not be returned to the program.

To return to normal end-of-record processing, call this FDEVICECONTROL again, specifying a null value as the EOR character. The EOR character will be reset to the default character, which is the carriage return. If you were processing in transparent mode prior to changing the EOR character, and want to return to using the EOR character that you specified in the call that established transparent mode, you need to explicitly reset the EOR to that character.

Figure 4-3 FDEVICECONTROL to Specify EOR Character

```
{ Use FDEVICECONTROL to set EOR character to ']' }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 39;                                {Define Type 1 EOR}
parm2 := 2;                                 {Write only access}
value := ord(']') * 256;                    {EOR = ']' (bits 0:8)}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Buffer containing EOR}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 39 failed with fserror ',fserror);

end.
```

Controlling Backspace Processing

Among the capabilities provided by FDEVICECONTROL which are not available through other intrinsics is the ability to manipulate how backspace processing takes place at a device. You can specify the character that the DTC will recognize as a backspace when it is sent from a specific device. You can also, to a limited extent, control the action that the DTC will take as a result of the backspace character being entered at a specific device. These controls are independent of each other, enabling you to change either the backspace character, the backspace response, or both, depending on the needs of your application. They are only useful in standard editing mode, since backspace is not recognized as a special character in either transparent or binary editing modes.

You use FDEVICECONTROL with a control directive of 36 to change the character that the DTC will recognize and act on as a backspace character. In the program fragment shown in Figure 4-4, the backspace character in effect for the devicefile whose file number is contained in `port` is set to a dollar sign ($), using the Pascal/iX `ord` function to specify its value. The program then notifies the user that the backspace character has been changed. Read/Write access is set to 2, which means that the backspace character previously in effect will not be returned.

If a dollar sign is entered at the device after successful execution of the program, it will be treated as a backspace, and the DTC will perform backspace processing. The dollar sign will not be echoed to the terminal, nor will it be passed in the data stream as a character of data.

In addition, the special processing characteristics normally associated with the backspace character itself (**[CTRL]H**) will be disabled, and, if a backspace is entered from the device, it will be passed in the data stream.

Figure 4-4 Specifying a New Backspace Character

```
{ Use FDEVICECONTROL to set backspace character to '$' }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 36;                                {Define backspace character}
parm2 := 2;                                 {Write only access}
value := ord('$');                          {Backspace = '$' (bits 8:8)}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Buffer containing backspace char}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 36 failed with fserror ',fserror);

end.
```

The code fragment shown in Figure 4-5 illustrates another aspect of backspace processing that can be manipulated through the FDEVICECONTROL intrinsic. In this example, a control directive of 55 is used to select a backspace response action. The backspace response action is the action that will be taken by the DTC as a result of a backspace (or the designated backspace character) being entered from a terminal.

There are two valid backspace response actions for MPE/iX systems. The default response is for the DTC to discard the character that was to the left of the terminal's cursor when the backspace was entered and to move the cursor back one space. The discarded character remains on the terminal screen until the user types a new character.

The second valid backspace response action is for the discarded character to be erased from the screen. To accomplish this, the DTC backspaces the terminal cursor as usual, but then transmits a space to erase the character from the screen and a second backspace to place the cursor at the correct position for the next character to be entered.

Either of these backspace response actions can be set through FDEVICECONTROL using a control directive of 55. To specify that the character on the screen should be erased, set the value of **buffer** to 5. To return to normal backspace response, set the value of **buffer** to 1.

The example shown in Figure 4-5 sets the backspace response to erase by specifying a 5 as the value of **buffer**.

Figure 4-5 Setting Backspace Response

```
{ Use FDEVICECONTROL to set backspace response to erase }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 55;                                {Set backspace response}
parm2 := 2;                                 {Write only access}
value := 5;                                 {Backspace response = erase}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Backspace response}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 55 failed with fserror ',fserror);

end.
```

Controlling Device XON/XOFF Processing

Another device control feature provided by the FDEVICECONTROL intrinsic is the ability to disable (and subsequently reenable) XON/XOFF flow control between a device and the DTC.

XON/XOFF protocol provides the main method of flow control used by MPE/iX systems. The protocol controls the flow of data transmitted between the DTC and a device to protect against data overruns.

XON/XOFF flow control allows the device receiving a transmission to stop the flow of data coming in should the device become temporarily unable to receive the data. It does this by sending an XOFF character to the DTC. XOFF is the ASCII DC3 character (**[CTRL]S**).

When the device is once again able to receive data, it transmits an XON character to signal that the DTC should resume data transmission. XON is the ASCII DC1 character (**[CTRL]Q**).

When XON/XOFF flow control is enabled, the DC1 and DC3 characters are not transmitted to the host. XON/XOFF flow control is enabled by default. All devices supported for use on MPE/iX systems are capable of using XON/XOFF protocol.

The only time XON/XOFF flow control is normally not active is during binary mode reads. This is because the DC1 and DC3 characters are not recognized as special characters in binary editing mode. However, no other special character processing occurs in this mode, either.

By using FDEVICECONTROL with a control directive of 26, it is possible to disable XON/XOFF flow control without altering any other aspect of how the device operates. It then becomes possible to transmit DC1 and DC3 characters to the host without causing the DTC to suspend transmission. All other special characters will still be recognized, and the associated special character functions will be performed. You should only use this capability when it is absolutely necessary to do so, however, and always with extreme care. When XON/XOFF flow control is disabled, it is possible for data overruns to occur and for data to be lost as a result.

The program fragment shown in Figure 4-6 provides an example of using FDEVICECONTROL to disable device XON/XOFF flow control. The **parm1** value of 26 specifies that the call is to set device XON/XOFF. The *value* variable contained in the **buffer** parameter is set to 0, which indicates that XON/XOFF is to be disabled by the call.

To re-enable device XON/XOFF you would call FDEVICECONTROL with a control directive of 26 again, but this time you would set the value of **buffer** to 1.

Figure 4-6 Disable Device XON/XOFF Processing

```
{ Use FDEVICECONTROL to disable XON/XOFF flow control }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 26;                                {Set/clear device XON/XOFF}
parm2 := 2;                                 {Write only access}
value := 0;                                 {Clear XON/XOFF}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Clear XON/XOFF}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 26 failed with fserror ',fserror);

end.
```

Controlling Host and Device XON/XOFF Processing

Another host and device control feature provided by the FDEVICECONTROL intrinsic is the ability to disable (and subsequently reenable) XON/XOFF flow control between device and the DTC. See Figure 4-7 for enabling host XON/XOFF processing.

XON/XOFF protocol provides the main method of flow control used by MPE/iX systems. The protocol controls the flow of data transmitted between DTC and the device to protect against data overruns.

Device XON/XOFF flow control allows the device receiving a transmission to stop the flow of data coming in should the device become temporarily unable to receive the data. It does this by sending an XOFF character to the DTC. XOFF is the ASCII DC3 character (**[CTRL]S**).

Host XON/XOFF flow control allows the DTC receiving a transmission to stop the flow of data coming in should the DTC become temporarily unable to receive the data. It does this by sending an XOFF character to the device. XOFF is the ASCII DC3 character.

When the device is once again able to receive data, it transmits an XON character to signal that the DTC should resume data transmission. XON is the ASCII DC1 character (**[CTRL]Q**)

When the DTC is once again able to receive data, it transmits an XON character to signal that the device should resume data transmission. XON is the ASCII DC1 character (**[CNTRL]Q**).

By using FDEVICECONTROL with a control directive of 76, it is possible to disable XON/XOFF flow control between device and the DTC.

The functionality for device flow control (FDEVICECONTROL 192,26) is maintained for backward compatibility, even though that can be done through this FDEVICECONTROL.

Figure 4-7 Enable Host XON/XOFF Processing

```
{ Use FDEVICECONTROL to set escape sequence read termination }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 69;                                {Suppress echo of read termination}
parm2 := 2;                                 {Write only access}
value := 1;                                 {Enable suppression of echo}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Enable suppression of echo}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 69 failed with fserror ',fserror);


parm1 := 68;                                {Set/clr escape sequence read term}
parm2 := 2;                                 {Write only access}
value := 1;                                 {Enable escape sequence read term}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Enable suppression of echo}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 69 failed with fserror ',fserror);

end.
```

Setting the Number of Stop Bits

Among the capabilities provided by FDEVICECONTROL is to specify or set the number of stop bits used. By using FDEVICECONTROL with a control directive of 71, it is possible to set the number of stop bits used. There are two possible **buffer** values to specify the number of stop bits used. See Figure 4-8 on how to set the number of stop bits equal to two.

In asynchronous communication, the stop bit informs the receiving device where the character ends. Thus it helps the receiving and sending machines to synchronize the transmission.

Figure 4-8 Set the Number of Stop Bits

```
{ Use FDEVICECONTROL to set stop bits }

program test (input, output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL; intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1  := 71;              {Set number of stop bits to 1 or 2}
parm2  := 2;               {write only access}
value  := 1;               {2 stop bits}

FDEVICECONTROL( input_filenum,      {Filenum for $STDIN}
                value         ,     {2 stop bits}
                1 .           ,     {Length of buffer (16 bit words) }
                192           ,     {192 = Device Control directive}
                parm1         ,     {Control directive for stop bit}
                parm2         ,     {Access directive (write only) }
                fserror             {Buffer to hold fserror code   }
              );

if (ccode <> CCE) then
   writeln ('FDEVICECONTROL 192,71 failed with fserror ',fserror);

end.
```

Enabling Escape Sequence Read Termination

FDEVICECONTROL has the ability to control the interpretation of escape sequences. Escape sequences are usually generated by function keys or cursor control keys. The escape sequence read termination (**parm1** = 68) control directive allows your application to react immediately to a special key when it reads it.

The escape sequence read termination causes a read to terminate when the escape character is encountered in input data. However, the read does not terminate immediately.

The DTC starts a timer immediately after it receives the escape character. The DTC sets the timer to the period of time required to transmit two characters at the current line speed. Characters that are automatically transmitted as part of the escape sequence will be received. When the entire escape sequence is transmitted, the timer will expire, the read terminates, and your application can respond to the special meaning of the escape sequence.

NOTE

The escape sequence may contain any number of characters, but the time between the reception of each character in the sequence cannot exceed the timer.

Enable/Disable escape sequence read termination (**parm1** = 68) should be accompanied with the enable/disable suppress echo of read termination characters function (**parm1** = 69). This prevents the DTC from echoing the escape sequence characters and guarantees that your application always has control of the position of the cursor. For example, you could design a function key to display a help message and then return to the point on the input line where the function key was typed. See Figure 4-9 for escape sequence read termination.

Figure 4-9 Escape Sequence Read Termination

```
{ Use FDEVICECONTROL to set escape sequence read termination }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 69;                                {Suppress echo of read termination}
parm2 := 2;                                 {Write only access}
value := 1;                                 {Enable suppression of echo}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Enable suppression of echo}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 69 failed with fserror ',fserror);


parm1 := 68;                                {Set/clr escape sequence read term}
parm2 := 2;                                 {Write only access}
value := 1;                                 {Enable escape sequence read term}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Enable suppression of echo}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 69 failed with fserror ',fserror);

end.
```

5 Using Subsystem Break

If your application will be run in session mode, you can include a subsystem break routine that allows a user running the program interactively to interrupt a task being performed by the program. To provide this capability you must include calls to a number of system intrinsics and supply a special subroutine that will run when subsystem break is invoked.

This chapter describes the process of implementing a subsystem break handler in your program. It includes discussions of the following:

- Intrinsics required to enable and perform subsystem break processing.
- Intrinsics that control how subsystem break processing occurs.
- Sample procedures demonstrating the use of these intrinsics.
- Additional considerations for allowing subsystem break in a program.

The Break Keys

Two kinds of breaks can be made available to users running in session mode. System break, the so called “hard” break, lets the terminal user suspend the currently running process and access the Command Interpreter to enter MPE/iX commands. The user can then type **:RESUME** to return to the interrupted process or **:ABORT** to terminate the suspended process. System break is enabled by default; you must explicitly disable system break through the FCONTROL (or FDEVICECONTROL) intrinsic.

The second kind of break that can be made available, subsystem break, must be explicitly enabled by your program. Subsystem break allows you to specify the exact way your program will respond to a user who invokes subsystem break while the program is executing.

Subsystem break is often referred to as **[CTRL]Y** break, because the common default for subsystem break is **[CTRL]Y**, activated by pressing the **[CTRL]** key and the **Y** key simultaneously. This sends an end-of-medium (EM) signal when entered from any MPE/iX supported terminal. If subsystem break is enabled for the program running on the terminal, this signal causes the program to pass control to a break handler subroutine.

It is a good idea to include subsystem break processing capabilities in any program that may be used to perform time consuming tasks, such as listing a large file or searching a database for specific information. By including a subsystem break routine you make it possible for the user to change his mind and interrupt a task without aborting the application.

A subsystem break procedure will usually allow the user to interrupt the program at any point to return to the main program prompt or menu. The procedure may also write a line of information to the terminal to indicate that a subsystem break has occurred. You, as the programmer, control exactly what will take place when subsystem break is entered by the way you program the break handler subroutine.

Subsystem break may be invoked in either standard or transparent editing mode, but the program must be running in a session in order for it to have any effect. You should not attempt to enable subsystem break if your program is performing block mode reads. Subsystem break is automatically disabled at FCLOSE.

Subsystem Break Intrinsics

When you want to make subsystem break processing available in a program there are several intrinsics that you must call. These intrinsics enable subsystem break and set up the procedure that will execute if the subsystem break character is entered while the program is running.

You use the XCONTRAP intrinsic to specify the address of the break handler subroutine that will execute when the user presses subsystem break. This subroutine is called the **[CTRL]Y** trap procedure, and the call to XCONTRAP is referred to as “arming the **[CTRL]Y** trap”. No subsystem break processing can occur unless this trap is armed by your program through a call to XCONTRAP. Only one **[CTRL]Y** trap handler can be active at any given time for a process.

You use the FCONTROL intrinsic to enable subsystem break processing. You must issue a call to FCONTROL with a **controlcode** value of 17 for the subsystem break signal to be recognized by the program. This is true even if the **[CTRL]Y** trap is already armed.

You use the RESETCONTROL intrinsic to reset the trap procedure after a subsystem break has occurred. Once subsystem break is invoked, the program will not respond to another subsystem break signal until RESETCONTROL is called.

There are also several other intrinsics that you can use to control how subsystem break processing occurs.

When you enable transparent editing mode, either through a call to FCONTROL with a **controlcode** value of 41, or FDEVICECONTROL using a control directive of 15, you are required to enter a value for the subsystem break character through one of the parameters of the call. You can specify any character not otherwise recognized as a special character in transparent mode as the new subsystem break character, including **[CTRL]Y**. (If you use a value of 0 for the subsystem break character, subsystem break processing will be disabled.)

You can also specify a new character to be used as the subsystem break character in either standard or transparent editing mode by calling FDEVICECONTROL using a control directive of 41.

Finally, you can call FDEVICECONTROL using a control directive of 57 to determine the current value of the subsystem break character. This FDEVICECONTROL function provides Read access only.

Refer to Chapter 8, “Intrinsics Reference,” in this manual, for descriptions of the syntax and parameters of each of these intrinsics, as well as for additional information regarding their use.

Subsystem Break Example

The code samples shown in the following figures, illustrate how you might provide subsystem break capabilities in a program. The figures show two procedures, one to handle the subsystem break itself, and one to set up and enable subsystem break.

The sample procedure in Figure 5-1 is a break handler routine. This is the procedure to which XCONTRAP will pass control when the subsystem break signal is sent from a user's terminal. Control might be passed at any point in the execution of the process.

In this example, when the break handler gets control from XCONTRAP, it writes a message to the user's terminal to verify that a subsystem break has occurred. The handler then takes whatever action is coded as the appropriate subsystem break response for the program. Finally, the break handler calls RESETCONTROL, and checks for successful completion before the procedure ends, (using the Pascal/iX *ccode* function). If the RESETCONTROL intrinsic does not complete successfully, the procedure will notify the user that the attempt to reenable subsystem break failed.

Note that the RESETCONTROL call does not have to be included in the break handling procedure. You can call RESETCONTROL from any place in the application (keeping in mind that it must execute before another subsystem break is to be allowed). However, the break handler is a convenient place to put the call if you want to make sure of its timely execution.

Figure 5-1 Illustrating a Subsystem Break Handler

```
{------------------------------------------------------------------------}
{ XCONTRAP will transfer control into this procedure.  This procedure will }
{ print a message so the user knows that a subsystem break occurred.  It will }
{ then re-enable ^Y and exit.                                            }
{------------------------------------------------------------------------}
procedure controly;

var
  write_buffer  : string [23];

begin

write_buffer := '...Subsystem BREAK!!';

FWRITE( output_filenum ,                          {Filenum for output}
        write_buffer   ,                          {Buffer with output data}
        -20            ,                          {Data length in bytes}
        0                                         {Control code}
      );


{-------------------------------}
{ Code to handle subsystem break }
{ event would go here...         }
{-------------------------------}


RESETCONTROL;

if (ccode <> CCE) then                            {RESETCONTROL failed}
  writeln(output,'Failed to re-enable subsystem break.');

end;
```

The procedure illustrated by Figure 5-2 shows the other half of what you must do to provide subsystem break capabilities in a program. This procedure sets up and enables subsystem break.

The procedure first calls XCONTRAP, which passes the address of the **[CTRL]Y** procedure shown in Figure 5-1. This arms the **[CTRL]Y** trap (enables the subsystem break handler). The Pascal/iX `waddress` function is used to pass the address in this example.

The address of the break handler is passed in the **plabel** parameter of the XCONTRAP intrinsic. The **oldplabel** parameter will return the address of the old trap handler, if one was previously armed. If there was no previous address, this parameter will return a 0.

After XCONTRAP executes, the procedure checks to make sure it completed successfully. Again, the procedure uses the Pascal/iX *ccode* function to check for errors. If XCONTRAP did not complete successfully, the user is informed of the failure, and the procedure ends.

If the call to XCONTRAP is successful, the procedure calls FCONTROL, using a **controlcode** value of 17. This call actually enables subsystem break processing and makes it possible for the application to respond if **[CTRL]Y** is entered. Again, the procedure checks for errors following the call and will inform the user should the attempt to enable subsystem break fail.

Subsystem break can also be disabled at any time, by calling FCONTROL again, using 16 as the **controlcode** value.

Figure 5-2 Procedure to Enable Subsystem Break

```
{------------------------------------------------------------------------}
{ This procedure sets up the control-Y trap by calling XCONTRAP and passing   }
{ it a parameter 'plabel' which is the entry point for the procedure that     }
{ actually handles the subsystem break.  If that succeeds it will use FCONTROL}
{ to enable the subsystem break key.  If either action fails, subsystem break }
{ will not function and an error message will be sent to the user.            }
{------------------------------------------------------------------------}
procedure enable_sub_bk;

var
  dummy      : shortint;
  oldplabel  : integer;
  plabel     : integer;

begin

plabel := waddress( CONTROLY );                {Get address of procedure to be
                                                used for SSBK handler}

XCONTRAP( plabel, oldplabel );                 {Set new SSBK handler entry point,
                                                save old entry point}

if (ccode = CCE) then
  begin

  FCONTROL( input_filenum  ,                   {Filenum for $STDIN}
            17             ,                   {Enable subsystem break}
            dummy                              {Dummy}
          );

  if (ccode <> CCE) then
    writeln(output, 'FCONTROL(17) failed to enable subsystem break.');

  end
else
  writeln(output, 'XCONTRAP failed to enable SSBK handler.');

end;
```

Subsystem Break Processing

A final consideration when building subsystem break processing capabilities into your program has to do with whether your program will run in Native Mode, Compatibility Mode, or both.

Native Mode vs. Compatibility Mode

So far, the discussion in this chapter has assumed that all processing is to take place in Native Mode (NM). There are times, however, when you will want to allow subsystem break processing to take place in programs that will run in Compatibility Mode (CM), or in both CM and NM. It is possible to do so by following some common sense rules when arming the **[CTRL]Y** trap.

If your program is to run in Native Mode only, you should use a Native Mode subsystem break procedure as well. XCONTRAP will execute in Native Mode and pass a 32-bit value as the external address of the subsystem break handler. This situation presents the most straight forward case for subsystem break processing. If the subsystem break character is received during execution of the program, control will simply be passed from the main application, which is running in Native Mode, to the break handler, which will also run in Native Mode. If your program is a Compatibility Mode application, the situation is considerably different. In this case, since XCONTRAP will execute in Compatibility Mode, it is only possible to pass a 16-bit external address in the call. This means that your trap handling routine must be a CM procedure as well.

If your program executes in both NM and CM the situation becomes somewhat more complex. HP recommends that you use a Native Mode trap handler in this case, which would mean that you would have to call XCONTRAP in Native Mode to pass a 32-bit address for the trap procedure. It is possible, however, to use a Compatibility Mode trap handler. (The discussion of the XCONTRAP intrinsic in Chapter 8, “Intrinsics Reference,” includes instructions for specifying a 16-bit address in the 32-bit **plabel** field.)

If you have supplied a Native Mode trap procedure, and the program is running in CM when a subsystem break signal is received, MPE/iX will switch to NM to allow the trap procedure to run. The reverse is also true; that is, a program running in NM with a CM trap handler will switch to CM if a subsystem break occurs.

For more information on handling **[CTRL]Y** traps, see the *Trap Handling* manual.

Subsystem Break Summary

The following summarizes subsystem break processing and the intrinsics involved:

1. A user written procedure must be provided to define how the program should act upon receiving a subsystem break.
2. A call to XCONTRAP must be included in the program, specifying the external address of the procedure written in Step 1 as the value of **plabel**. This arms the **[CTRL]Y** trap.
3. A call to FCONTROL(17) must be included in the program to enable the subsystem break function.
4. If the subsystem break character is received during execution of the program, the procedure specified in the call to XCONTRAP is executed.
5. A call to RESETCONTROL must be executed when the program is ready to receive another subsystem break.
6. A new subsystem break character may be defined through FCONTROL(41), if the terminal is placed in transparent mode, or through FDEVICECONTROL.
7. You can redefine your trap handling procedure at any point during execution of the program through another call to XCONTRAP.
8. If desirable at any time, FCONTROL(16) may be called to disable subsystem break. A call to FCONTROL(17) enables it again.

6 Typeahead Mode

Effective with Release A.20.00, a typeahead capability is supported for use on MPE/iX systems. Typeahead makes it possible for a user at a terminal to enter data before the read which will accept that data is actually posted. This is especially convenient when you need to enter a series of commands with system processing delays in between.

The typeahead facility is controlled programmatically through the FDEVICECONTROL intrinsic, using one of three control directives. This chapter explains each of these typeahead controls, and provides code samples demonstrating how you can do the following:

- Control whether or not typeahead is enabled at a terminal.
- Bypass the typeahead buffer and take data for the next read directly from a terminal device.
- Flush all information from the typeahead buffer before starting the next read.

NOTE

DTC Telnet typeahead is operational only when single echo is enabled. Use FDEVICECONTROL with **controlcode** value 192 and **parm1** value 63, or the TTUTIL program, to enable single echo. PAD typeahead is operational only when `initial echo` is disabled. Use FDEVICECONTROL with **controlcode** value 192 and **parm1** value 4, FCONTROL(13), or the TTUTIL program to disable `initial echo`.

Working in Typeahead Mode

By using the device controls described in this chapter, you can specify whether or not typeahead mode is enabled when your program executes. By allowing typeahead mode you can make it possible for the user of the application to enter data that the user knows will be required before the application actually requests the data.

Suppose, for example, that your program issues a series of questions designed to allow an interactive user with the proper access codes to gain entry to a database program. Without typeahead mode active, the program writes each question on the user's terminal screen and waits for a reply. Each reply may require some additional processing before the next question is written to the screen. In some cases, the delay between questions may be quite noticeable.

If typeahead mode is on, however, a user who is familiar with the series of questions can type replies to all of the questions at once, without regard to processing delays. As long as the replies are entered in the required sequence for the application and are separated from each other by an end-of-record character (such as carriage return), they will be accepted as valid replies.

After all of the replies have been entered, the user will still have to wait for the application to process the replies before being granted access to the database. But now all of the waiting is at the end of the question sequence, and the user is free to perform some other task instead of having to wait for the next question.

Enabling Typeahead Mode

As of version A.30.00 of MPE/iX, a terminal user can enable typeahead mode at the Command Interpreter level for any supported terminal. The command **:SETVAR HPTYPEAHEAD TRUE** enables typeahead, while the command **:SETVAR HPTYPEAHEAD FALSE** disables typeahead. (These commands replace the command file used to set typeahead mode in earlier MPE/iX versions supporting typeahead.)

In order for typeahead to be used in an application, however, the application itself must programmatically enable and manage typeahead through the FDEVICECONTROL functions explained in this chapter.

WARNING No attempt should be made to use typeahead mode at a terminal that is running an application which has not been specifically coded for typeahead processing.

How Typeahead Mode Works

When data is entered from a terminal with typeahead mode active, the data is staged in a special typeahead buffer before being sent through to the process requesting the data. Any read posted against a terminal with typeahead mode active will first access the typeahead buffer, rather than accepting data directly from the device.

The typeahead buffer is limited to 224 bytes; anything in excess of this limit will cause an XOFF to be sent. You should be sure that XON/XOFF transmission pacing is enabled for the terminal if typeahead mode is used. You should avoid using typeahead mode if you anticipate that reads will exceed 224 bytes, to avoid overflowing the typeahead buffer.

If typeahead is on but the terminal user has not typed ahead, the system will act in essentially the same way that it would if typeahead were not enabled. A DC1 read trigger will be sent to the terminal any time the system is ready to receive data, and there is no data in the typeahead buffer. If, however, the user has typed ahead, and there is data in the buffer, the system will simply accept the typed ahead data and will not send a DC1 read trigger to the terminal.

Typeahead Mode and Echo

When typeahead is in use the terminal user will, in most cases, see input echoed to the terminal twice. The first echo is a “convenience echo”, which appears when the data is being typed. This echo allows the user to see the characters that are typed, and to make corrections, if necessary, before entering the end-of-record character.

The second echo appears at the time the data in the typeahead buffer is actually accepted by the system. This echo lets the user know that the read is being processed.

A second type of typeahead mode exists—single echo typeahead mode. With this mode, the device driver withholds the echoing of typeahead data until a read is posted for the data.

There are, however, certain cases in which the first echo will not be seen at the terminal. For example, if the terminal user types ahead while the system is displaying data, such as the output to a **`:LISTF`** command, the convenience echo will not appear.

Also, a user whose terminal is in typeahead mode but who is not typing ahead will see the input echoed only once.

Typeahead Mode and Subsystem Break

When typeahead mode is active, the subsystem break character has additional functionality when typeahead is used, regardless of whether or not subsystem break is enabled. Receipt of the subsystem break character causes all data in the typeahead buffer to be discarded. This provides the user with a final chance to erase a previously typed command. But this feature should be used with caution; subsystem break does, after all, have its own function. Users who depend on this method to flush the typeahead buffer may find themselves processing undesired subsystem breaks if subsystem break is enabled.

Additional Typeahead Considerations

There are a number of actions that should be avoided when data is being typed ahead, such as changing a terminal's speed or parity settings.

If you were to enter the speed command using typeahead, then change the speed at the terminal and attempt to type ahead with the new speed, the terminal would hang. Similarly, if you were to change the parity setting before a typed ahead parity command could execute, data would be lost. Either of these problems can be avoided by simply not typing ahead when changing these settings. Wait for the commands to execute before changing the terminal's settings.

You will also encounter problems if you attempt to use typeahead with VPLUS or user block mode. If your program allows block mode reads, you should make sure that typeahead mode is disabled before the terminal is placed in block mode. Because of the additional terminal controls and status checks involved in processing a read in block mode, it is not possible to use block mode and typeahead at the same time.

Finally, no attempt should be made to enable typeahead from a terminal connected via PAD. Typeahead mode is not supported for this configuration.

Programming for Typeahead Mode

To programmatically set and control typeahead mode you use the FDEVICECONTROL intrinsic, specifying one of three control directives in the **parm1** value.

- A control directive of 51 is used to specify whether or not typeahead mode is on for the device. Use this call with the **buffer** parameter set to 1 to enable typeahead mode, or with the **buffer** parameter set to 0 to disable typeahead mode (**parm2** set to 2 or 3).
- A control directive of 60 allows you to discard any data in the typeahead buffer and read directly from the device.
- A control directive of 61 allows you to bypass the typeahead buffer for the next read while saving the data already in the typeahead buffer.
- A control directive of 63 is used to enable or disable single echo typeahead. Use this call with the **buffer** parameter set to 1 to enable typeahead mode, or with the **buffer** parameter set to 0 to disable single echo typeahead mode (**parm2** set to 2 or 3).

The code fragments that follow illustrate how these calls might look in your program.

NOTE

When typeahead is enabled it is enabled for the device and not specifically for an application. It is a good practice to make sure that any program that enables typeahead mode also restores the terminal's previous typeahead setting before it terminates. To do so, your program should save the value of the device's current typeahead setting. This is obtained either through a separate call to FDEVICECONTROL with a control directive of 51 and **parm2** set to 1 (Read access), or through the call which changes the setting, with **parm2** set to 3 (Read/Write access). When your program is ready to end execution, it will be able to return the device to its original typeahead setting through another FDEVICECONTROL call, using the value of the previous setting obtained in the first call as the value of **buffer**.

Setting Typeahead Mode

The code fragment shown in Figure 6-1 illustrates how you can use the FDEVICECONTROL intrinsic to allow typeahead processing for your program. A control directive of 51, set typeahead mode, is passed in **parm1**, and **parm2** is set to 2 to specify Write access. The **buffer** parameter contains a variable called *value*, set to 1 in this example, which will cause typeahead mode to be enabled. (A value of 0 would disable typeahead mode.)

In this example, the Pascal/iX *ccode* function checks for errors that might occur during execution of the intrinsic. If an error occurs, the program reports the file system error number associated with the error. If no errors occur, the program informs the user that typeahead is now on.

Figure 6-1 Program Fragment Setting Typeahead Mode

```
{ Use FDEVICECONTROL to flush typeahead buffer }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 60;                                {Flush typeahead buffer}
parm2 := 2;                                 {Write only access}
value := 1;                                 {Flush typeahead on next read}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Flush typeahead}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode = CCE) then
  writeln('Typeahead buffer flushed.')
else
  writeln('FDEVICECONTROL 192, 60 failed with fserror ',fserror)

end.
```

Flushing the Typeahead Buffer

Figure 6-2 illustrates how FDEVICECONTROL can be used to flush the typeahead buffer. For example, the buffer can be flushed if an error is detected in the data that was typed ahead. This FDEVICECONTROL is only valid if type ahead is enabled. The default is not to flush the typeahead buffer.

An FDEVICECONTROL call with a control directive of 60 will cause the buffer to be flushed at the time of the next read. The read can then be satisfied by data entered directly from the device. The typeahead buffer is flushed for a single read only; normal typeahead mode processing will resume for any following reads.

In the example, the **parm1** value is set to 60, flush typeahead buffer, and the **buffer** parameter (*value*) passes a value of 1. This specifies that all data currently in the typeahead buffer should be discarded, and the next read should access the device directly. Write access is specified in **parm2**, and the program checks for errors using the Pascal/iX *ccode* function.

Figure 6-2 Program Fragment to Flush Buffer

```
{ Use FDEVICECONTROL to bypass typeahead buffer }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 61;                                {Bypass typeahead buffer}
parm2 := 2;                                 {Write only access}
value := 1;                                 {Bypass typeahead on next read}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Bypass typeahead}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode = CCE) then
  writeln('Next read will bypass typeahead buffer.')
else
  writeln('FDEVICECONTROL 192, 61 failed with fserror ',fserror)

end.
```

Bypassing the Typeahead Buffer

Figure 6-3 is an example of using FDEVICECONTROL to bypass the typeahead buffer. You could use this capability to send a status request and obtain a reply directly from a terminal without affecting the data already in the typeahead buffer. This FDEVICECONTROL is only valid if typeahead is enabled. The default is not to bypass the typeahead buffer.

This FDEVICECONTROL call will cause the typeahead buffer to be bypassed for a single read only; normal typeahead mode processing will resume for any following reads, and data will be taken from the typeahead buffer if data is present.

When you use this request, you call FDEVICECONTROL with **parm1** set to 61, bypass typeahead mode, and pass a value of 1 in the **buffer** parameter (`value :=1`). As in the previous examples, Write access is specified in **parm2**, and the Pascal/iX *ccode* function is used for error checking.

NOTE

In cases where there is no read trigger (read trigger set to null) bypassing the typeahead buffer may have unexpected results. If, for example, your program is swapped out between the FWRITE and FREAD of a terminal status request, the terminal may appear to become “hung”, since there is no read trigger to synchronize terminal response with the posting of the read.

To avoid this problem, instead of bypassing the typeahead buffer to send a status request when no read trigger is present, the following should be the sequence that occurs:

1. = Disable typeahead mode
2. = Post a 0 byte read
3. = Enable typeahead mode
4. = (Lock keyboard) (FWRITE) Send status request
5. = FREAD of status
6. = FWRITE (unlock keyboard)

Figure 6-3 Program Fragment to Bypass Buffer

```
{ Use FDEVICECONTROL to set typeahead }

program test(input,output);

const
  CCG = 0;
  CCL = 1;
  CCE = 2;

var
  fserror        : shortint;
  input_filenum  : shortint;
  parm1          : shortint;
  parm2          : shortint;
  value          : shortint;

procedure FDEVICECONTROL;  intrinsic;


begin

{-----------------------------------------}
{ Input and output files should be opened  }
{ prior to executing this code...          }
{-----------------------------------------}

parm1 := 51;                                {Enable/disable typeahead}
parm2 := 2;                                 {Write only access}
value := 1;                                 {Enable typeahead}

FDEVICECONTROL( input_filenum ,             {Filenum for $STDIN}
                value         ,             {Enable typeahead}
                1             ,             {Length of buffer (16 bit words)}
                192           ,             {192 = Device Control directive}
                parm1         ,             {Control directive for device file}
                parm2         ,             {Access directive (write only)}
                fserror                     {Buffer to hold fserror code}
              );

if (ccode <> CCE) then
  writeln('FDEVICECONTROL 192, 51 failed with fserror ',fserror);

end.
```

7 Programming for PAD, DTC Telnet, Telnet/iX Server, and VT Devices

This chapter describes PAD, DTC Telnet, Telnet/iX Server, and VT access, available on properly equipped and configured MPE/iX systems. It has four parts. The first part details the special considerations involved when programmatically controlling PAD terminals and serial printers. The second part deals with the considerations involved when programmatically controlling DTC Telnet devices. The third part deals with the considerations involved when programmatically controlling Telnet/iX Server connections. The fourth part covers the considerations involved when programmatically controlling VT devices.

PAD Access to MPE/iX Systems

If you have X.25 cards, the programs you write may be used to control devices connected through a packet switched network (PSN) to a PAD device.

A PAD, or Packet Assembler/Disassembler, is a device that converts asynchronous character streams into packets that can be transmitted over a packet switching network. PAD packets follow the CCITT X.25 protocol recommendations.

This part includes discussions of the following:

- The physical components of a network with PAD support.
- Limits and restrictions that apply to PAD-connected devices.
- Security for PAD access.
- Considerations when using terminal control intrinsics with PAD devices.

PAD Access to MPE/iX Systems

The PAD support provided on MPE/iX systems conforms to the 1984 version of CCITT recommendations X.3 and X.29. Packet sizes of 128, 256, and 512 are supported. Access is provided for users of asynchronous devices connected to both public and private PADs.

A public PAD is a service provided by a Public Data Network (PDN), such as TELENET or TRANSPAC. Public PADs can be used with either a dial-up or a leased line.

A private PAD is one that is connected to a PDN as a host node having its own X.25 network address, but behaves as a PAD when connected to another node. Private PADs require leased lines to the network. Hewlett-Packard provides the HP2334A, HP2334A Plus, and HP2335A cluster controllers for use as private PADs.

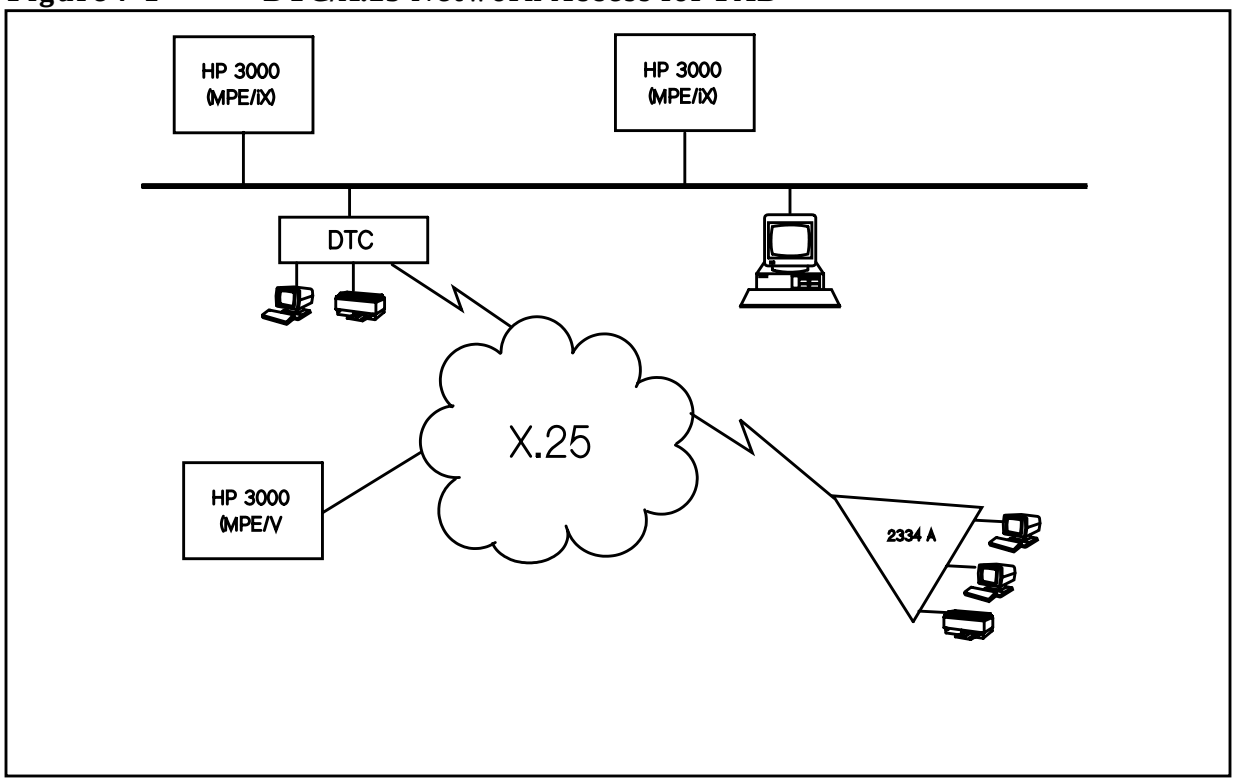
Hardware Requirements

PAD access is provided to MPE/iX networks through a DTC/X.25 Network Access card mounted in the DTC. Three types of DTCs support X.25 connections—the DTC 16, DTC 48, and DTC 72MX. In the DTC 48 and DTC 72MX, each access card takes the place of a terminal connect card in one of the DTC slots, and each DTC may contain up to three network access cards. DTC 48s may not have the access card in slot 0. See the *DTC Planning Guide* for more information on the DTC hardware. In the DTC 16, there are three slots. The first two slots can contain direct connect or modem cards, and the third slot can contain only a network access card.

The DTC is connected to the LAN as usual, with the MPE/iX systems to which it has access also connected to the LAN.

Figure 7-1 depicts PAD access to HP 3000 Series 900 Systems via a DTC equipped with the DTC/X.25 Network Access card.

Figure 7-1 DTC/X.25 Network Access for PAD



Network Configuration Requirements

The parameters for asynchronous devices that can use the PAD support facility provided by the DTC are configured through either the OpenView DTC Manager or through host-based configuration. PAD support information for each MPE/iX host system must be supplied via NMMGR configuration.

PAD Configuration Values. Most of the characteristics that define how asynchronous devices connected through a packet switched network will act are determined by the configuration of the PAD itself. See the documentation for the PAD for more information on how to define the characteristics of its devices.

PAD support for the network is configured, in part, through the OpenView DTC Manager or through host-based configuration. Configuration values are entered to define the types of devices that may access the network. Levels 1 through 3 of X.25 must also be configured.

PAD support parameters for each MPE/iX host system that will allow device connections via PAD must be defined in the NMMGR configuration, along with the device profiles to be associated with PAD devices. PAD devices requiring non-logon (programmatic) access must be configured as nailed PAD devices. Devices that require only logon access can be configured as non-nailed PAD terminals.

Security. Several levels of security are provided to assure that no unauthorized access can be obtained to the MPE/iX systems on your network.

See *Using the OpenView DTC Manager* for more information on configuring and using security for incoming PAD calls.

For host-based security information, see *Configuring and Managing Host-Based X.25 Links*.

PAD Device Access

All devices supported for asynchronous connection to the DTC can be connected via PAD. The devices must support the PAD and modem port configuration.

Hewlett-Packard provides a special terminal type file, terminal type 24, for support of terminals connected via PAD. PAD support is provided for serial printers with printer types 18 or 26.

Log On Access. If a terminal is properly configured for the PAD it is connected to, and PAD connections are allowed for the MPE/iX network you are accessing, you can establish a terminal session on a desired MPE/iX system.

You must first establish a connection from the PAD to the system you want to open the session on. You may choose to access a specific MPE/iX system directly by entering that system's X.25 address in the connection request. Alternatively, you may access the DTC Switching User Interface by entering the X.25 address of the DTC and connect to the system you want by using the connection request provided by the DTC **Switching User Interface**. (Regardless of how you choose to initiate your connection, you will be able to switch between host systems on the LAN once you terminate your session.)

NOTE

When connected via PAD, you must log-off with the MPE **:BYE** command to return to the PAD prompt. It is not possible to escape from data transfer with a session active.

Your virtual connection is not disconnected until you log out from the DTC switching user interface or until the inactivity timer configured via the OpenView DTC Manager expires.

Only character mode and VPLUS block mode applications may be run from PAD devices; no other types of block mode may be used.

Programmatic Access. Terminals and serial printers can be accessed programmatically if they are connected to a private PAD and configured as nailed devices on the MPE/iX host on which the application is running. To open a device programmatically you use the FOPEN or HPFOPEN intrinsic with the *device* option specifying the ldev number or device class that was associated with the remote device through the NMMGR configuration.

General PAD Restrictions

Some basic limitations apply to the way transmission occurs and to the operations that are supported over PAD connections:

- It is necessary to press **[Return]** to forward data from the PAD across the network to the DTC even if the byte count specified in the read is reached.
- Some special character sequences, such as subsystem break (**[CTRL]Y** by default), must also be forwarded with a **[Return]**.
- You cannot use a PAD terminal as a console.
- Use of customized terminal and printer types files is not supported over PAD.
- You cannot use the **:STARTSESS** command with ldevs that allow PAD device connections.
- Typeahead mode is not supported over PAD on versions prior to release 4.5, C.45.00.
- Binary mode transfers are not supported between a PAD terminal and a MPE/iX or HP-UX host on versions prior to release 4.5, C.45.00.
- User Block Mode and HP Block Mode applications are not supported on PAD devices.
- Parity generation and checking is not supported programmatically over PAD.
- The line deletion response (**[CTRL]X** echo characters, ! ! ! by default) cannot be enabled or disabled programmatically over PAD (though some PADs allow you to define line deletion characters locally).
- PAD printers cannot be shared by more than one host.

PAD Programming Considerations

Most file system intrinsics can be used to control devices connected to PADs as if the devices were locally connected. Both the FCONTROL and FDEVICECONTROL intrinsics are supported for PAD device control. In most cases the device control calls work as normal.

There are, however, a number of differences in the way device control functions operate over PAD connections. The individual differences in the operation of each FCONTROL and FDEVICECONTROL call are documented in the Intrinsics Reference chapter of this manual. The following summary is provided here for your convenience.

FCONTROL

Some FCONTROLs are ignored by PAD. These are calls that attempt to control some function that is not supported over PAD, such as changing terminal speed or setting parity. These calls will return a CCE condition code, but no device control action will take place. Any values returned in these calls may not be a true reflection of device control settings. These FCONTROLs include:

- FCONTROL(10), set line speed.
- FCONTROL(11), set line speed
- FCONTROL(23), disable parity checking.
- FCONTROL(24), enable parity checking.
- FCONTROL(34), allow printing of **[CTRL]X** echo.
- FCONTROL(35), disable printing of **[CTRL]X** echo.
- FCONTROL(36), define parity setting.
- FCONTROL(40), determine current speed.

Reads following a call to FCONTROL(27), which enables binary mode, return a CCL condition code with the file system error `FSERR20 (INVALID OPERATION)`, until binary mode is disabled (FCONTROL(26)), if you are using a version prior to release 4.5.

Calls to FCONTROL(37), to set both terminal type and speed, will change the terminal type setting, but will have no impact on how the device actually operates. A CCE condition code will be returned. (Note that this is a difference from how PAD devices behave on MPE V systems.)

Some caution should be taken when using FCONTROL(4), which sets a timeout interval for a read. Keep in mind that some delays may be caused by the network over which the PAD is transmitting. In such cases you may want to add some time to the timeout value you would normally specify for the call.

FDEVICECONTROL

The FDEVICECONTROL intrinsic is also supported on MPE/iX systems with PAD connections. However, there are also several limitations on the device control operations they perform. The following

FDEVICECONTROL **parm1** values, with 192 specified as the **controlcode**, will return a CCE condition code but have no effect on the operation of the device or the transmission of data:

parm1	Action
1	Specify terminal type or printer type file
3	Set line speed
9, 11	Enable/disable parity checking
10, 12	Set parity type
14	Set line deletion response
26	Enable/disable XON/XOFF flow control
27	Set XOFF timer value
29	Define block mode alert character
30	Define block mode trigger character
51	Set typeahead mode
55	Select backspace response action
56	Specify data bits per character
60	Flush typeahead buffer
61	Bypass typeahead buffer
62	Set quiesce I/O
63	Set single echo typeahead mode
64	Ignore parity error on input mode
65	Set inter-byte timer
66	Define multiple type 2 EOR characters (AEOR)
67	Delete (DEL) to backspace (BS) mapping
68	Enable/disable escape sequence read termination
69	Enable/disable suppress echo of read termination characters
73	Set FCLOSE timeout value

FDEVICECONTROL also verifies that a terminal is configured as a PAD device. A **parm1** value of 28 (**controlcode** 192) returns a value corresponding to the type of block mode supported by the driver. A value of 15 is returned by this call when the device is a PAD terminal supporting (VPLUS) page block mode.

DTC Telnet Access to MPE/iX Systems

If your network is managed by the OpenView DTC Manager, and at least one of the DTCs on the LAN are equipped with a Telnet Access Card, the programs you write may be used to control devices connected to the DTC via a Telnet connection to an HP 9000 or other ARPA machine. Collectively, these devices are called DTC Telnet devices. The Telnet Access Card receives asynchronous character streams over the LAN from an ARPA machine.

What's Included

This section includes information on:

- The hardware requirements for DTC Telnet support.
- The network configuration requirements for DTC Telnet support.
- Limits and restrictions that apply to DTC Telnet-connected devices.

Hardware Requirements

DTC Telnet access is provided to MPE/iX networks through a Telnet Access Card mounted in the DTC or with the Telnet Express Box. An access card takes the place of a terminal connect card in one of the DTC slots for DTC 48 and DTC 72MX (the Telnet card cannot be in slot 0 for DTC 48). The Telnet card cannot coexist with an X.25 Network Access Card in the DTC 48. See *DTC Planning Guide* for more information on the DTC hardware.

Both the DTC and the MPE/iX system accessing it must be connected to a LAN. There must also be a network management workstation on the LAN running the OpenView DTC Manager software that controls all DTC network operations.

Network configuration Requirements

MPE/iX host names and IP addresses that can use the DTC Telnet support facility is configured through the OpenView DTC Manager. DTC Telnet support information for each MPE/iX host system must be supplied via the NMMGR configuration software. Also, you must configure a sufficient number of non-nailed TIO ports.

DTC Telent Configuration Values. DTC Telnet support for the network is configured using both the OpenView DTC Manager software on the network management workstation and the NMMGR configuration software on the host.

DTC Telnet support parameters for each MPE/iX host system that will allow device connections via DTC Telnet must be defined in the NMMGR configuration, along with the device profiles to be associated

with DTC Telnet devices. DTC Telnet devices must be configured as non-nailed TIO terminals. The terminal type for these terminals must be set at 10 or 18.

For more information on configuration see both *Configuring Systems for Terminals, Printers, and Other Serial Devices* and *Using the OpenView DTC Manager*.

General DTC Telnet Restrictions

Some basic limitations apply to the way transmission occurs and to the operations that are supported over DTC Telnet connections:

- Use of a DTC Telnet terminal as a console is not supported.
- Use of customized terminal type files is not supported over DTC Telnet.
- Typeahead mode is not supported over DTC Telnet.
- Binary mode transfers are not supported between a DTC Telnet terminal and an MPE/iX host.
- User Block Mode and HP Block Mode applications are not supported on DTC Telnet devices. Only VPLUS block mode applications may be run from DTC Telnet devices; no other types of block mode may be used.
- Parity generation and checking are not supported programmatically over DTC Telnet.

DTC Telnet Programming Considerations

Most file system intrinsics can be used to control devices connected to DTC Telnet as if the devices were locally connected. Both the FCONTROL and FDEVICECONTROL intrinsics are supported for DTC Telnet device control. In most cases the device control calls work as they would with non Telnet connections.

There are, however, a number of differences in the way device control functions operate over DTC Telnet connections. The individual differences in the operation of each FCONTROL and FDEVICECONTROL call are documented in the Intrinsics Reference chapter of this manual. The following summary is provided here for your convenience.

FCONTROL

Some FCONTROLs are ignored by DTC Telnet. These are calls that attempt to control some function that is not supported over DTC Telnet, such as changing terminal speed or setting parity. These calls will return a CCE condition code, but no device control action will take place. Any values returned in these calls may not be a true reflection of device control settings. These FCONTROLs include:

- FCONTROL(10), set line speed.
- FCONTROL(11), set line speed
- FCONTROL(23), disable parity checking.
- FCONTROL(24), enable parity checking.
- FCONTROL(28), disable user block mode.
- FCONTROL(29), enable user block mode.
- FCONTROL(36), define parity setting.
- FCONTROL(40), determine current speed.

Reads following a call to FCONTROL(27), which enables binary mode, return a CCL condition code with the file system error `FSERR20 (INVALID OPERATION)`. All subsequent reads will return an error, until binary mode is disabled (FCONTROL(26)).

Calls to FCONTROL(37), to set both terminal type and speed, will change the terminal type setting, but will have no impact on how the device actually operates. A CCE condition code will be returned.

Some caution should be taken when using FCONTROL(4), which sets a timeout interval for a read. Keep in mind that some delays may be caused by the LAN over which the DTC Telnet is transmitting. In such cases you may want to add some time to the timeout value you would normally specify for the call.

FDEVICECONTROL

The FDEVICECONTROL intrinsic is also supported on MPE/iX systems with DTC Telnet connections. However, there are also several limitations on the device control operations they perform. The following FDEVICECONTROL **parm1** values, with 192 specified as the **controlcode**, will return a CCE condition code but have no effect on the operation of the device or the transmission of data:

parm1	**Action**
1	Specify terminal type file
3	Set line speed
9, 11	Parity checking enable/disable
10, 12	Set parity type
14	Set line deletion response
26	XON/XOFF flow control enable/disable
27	Set XOFF timer value
28	Block mode supported
29	Define block mode alert character

30	Define block mode trigger character
32	Define read trigger character
51	Set typeahead mode
55	Select backspace response action
56	Specify data bits per character
60	Flush typeahead buffer
61	Bypass typeahead buffer
62	Set quiesce I/O
63	Set single echo typeahead mode
64	Ignore parity error on input mode
65	Set inter-byte timer
66	Define multiple type 2 EOR characters (AEOR)
67	Delete (DEL) to backspace (BS) mapping
68	Enable/disable escape sequence read termination
69	Enable/disable suppress echo of read termination characters
73	Set FCLOSE timeout value
74	Suppress the last form feed

Telnet/iX Server Access on MPE/iX Systems

For details on using the Telnet/iX Client and Server on MPE/iX systems, as well as information on the Telnet/iX Client commands, refer to the *HP Telnet/iX User's Guide*. For more information on configuring Telnet/iX, as well as more general information on Internet Services, please refer to *Configuring and Managing MPE/iX Internet Services*.

What's Included

This section includes information on:

- Limits and restrictions that apply to Telnet/iX Server connected devices.
- Recommended programming practices for Telnet/iX Server support.

General Telnet/iX Server Restrictions

- Only sessions, not jobs, can be established on the MPE/iX system by the local Telnet client.
- Although you may use the MPE/iX CONSOLE command to switch the console device to a previously connected Telnet/iX Server Ldev, **[CTRL]-a** and **[CTRL]-b** functionality are not supported on Telnet/iX Server Ldevs.
- Use of customized terminal types is not supported.
- Many FCONTROL and FDEVICECONTROL intrinsics, including all block modes, require the use of a 2392 compatible terminal emulator on the Telnet client side.

Telnet/iX Server Programming Considerations

The following provides details on the limitations of file system intrinsics used to control devices connected to an MPE/iX system via Telnet/iX. For more information about the following intrinsics, refer to Chapter 8, "Intrinsics Reference," of this manual.

FCONTROL

Some FCONTROLs are not supported for use with Telnet/iX Server connections. If a program uses an unsupported FCONTROL for a Telnet/iX connected device, the Telnet/iX driver will return a CCE condition code, but no device control action will take place. Any values returned in these calls may not be a true reflection of device control settings. These FCONTROLs include:

- FCONTROL(10), set line speed.
- FCONTROL(11), set line speed.

- FCONTROL(22), return last read time.
- FCONTROL(23), disable parity generation and checking.
- FCONTROL(24), enable parity generation and checking.
- FCONTROL(36), define parity setting.
- FCONTROL(37), set terminal type and speed.
- FCONTROL(38), set terminal type.
- FCONTROL(40), determine current speed.

The following FCONTROLs are not supported when the local device is an HP-UX client:

- FCONTROL(28), disable user block mode.
- FCONTROL(29), enable user block mode.

Use of FCONTROL(27) (enable binary mode) when the local device is an HP-UX Telnet client, will require the user to set the HP-UX client into binary (using the HP-UX Telnet client toggle command) before the FCONTROL(27) is issued to the Telnet/iX Server. If the HP-UX client is not set into binary before the FCONTROL(27) is issued, the FCONTROL will return with CCL. The user must disable binary on the HP-UX client before binary is disabled on the Telnet/iX Server with FCONTROL(28).

Some caution should be taken when using FCONTROL(4), which sets a timeout interval for a read. Keep in mind that some delays may be caused by the network over which your Telnet connection is operating. In such cases you may want to add some time to the timeout value you would normally specify for the call.

FDEVICECONTROL

Some FDEVICECONTROLs are not supported for use with Telnet/iX Server connections. If a program uses an unsupported FDEVICECONTROL, the Telnet/iX driver will return a CCE condition code, but no device control action will take place. Any values returned in these calls may not be a true reflection of device The following discussion deals with **parm1** values with 192 specified as **controlcode**.

Unsupported FDEVICECONTROLs include:

parm1	**Action**
1	Specify terminal type or printer type file
3	Set line speed
8	Return last input time
9, 11	Set parity generation and checking

10, 12	Set parity type
11	Set parity generation and checking
26	Enable/disable XON/XOFF flow control
27	Set XOFF timer value
56	Specify data bits per character
64	Ignore parity error on input mode
72	Return PAD line speed
73	Set FCLOSE timeout value
74	Suppress the last form feed

FREAD and FWRITE

In general, the Telnet/iX Server will support the same size data buffers for FREAD and FWRITE as the DTC. Currently, Telnet/iX will accept data buffers up to 4096 bytes long. However, communicating with PC clients is limited to buffers no larger than 3500 bytes. When communicating with a PC client, Telnet/iX will not support transfers larger than 3500 bytes. Data loss may occur if you exceed the 3500 byte buffer limitation.

Recommended Programming Practices

Application developers are encouraged to use programming practices that will facilitate successful operation in many different network configurations.

The following should be considered:

1. Because the PC only supports a 3500 byte data buffer, applications should only issue FREADs and FWRITEs of less than 3500 bytes even though the DTC may support larger buffers. If an application attempts to issue an FWRITE larger than the configuration supports, the data will not be processed. The remote host will not detect the buffer size support issue; the application thus receives no error although the data is not processed.
2. Application programs should check for device availability after IO requests. If a terminal logical device becomes unavailable, the application will receive a file system error FSERR24 indicating that the device is no longer available and that the application should terminate. This guideline minimizes unnecessary use of CPU resources.

Virtual Terminal Access on MPE/iX Systems

The Virtual Terminal (VT) service of Network Services (NS) is designed to allow applications on the MPE/iX remote host to access devices on local clients as if they were terminals. The VT service on an MPE/iX host will support connections from MPE/iX, MPE V, PCs, and HP-UX systems.

For details on the required hardware, software and network configurations for VT access on MPE/iX systems, as well as more general information on NS, refer to *Getting Started With NS 3000/iX* and *NS 3000/iX Operations and Maintenance Reference Manual*.

What's Included

This section includes information on:

- Limits and restrictions that apply to VT connected devices.
- Recommended programming practices for VT support.

General VT Restrictions

Only sessions, not jobs, can be established on the remote host by the local client; connections to the remote host under job control will be transparent to the application. This distinction is made for two reasons. First, some applications handle error recovery differently depending on whether the application is run from a session or a job. Second, most FCONTROLs and FDEVICECONTROLs are not supported in the job environment; they will often be ignored although no error is reported.

VT Programming Considerations

The following file system intrinsics can be used to control devices connected to an MPE/iX system via VT. For more information about the following intrinsics, refer to Chapter 8, "Intrinsics Reference," of this manual.

FCONTROL

Some FCONTROLs are not supported for use with VT. If a program uses an unsupported FCONTROL for a VT connected device, the VT driver will return a CCE condition code, but no device control action will take place. Any values returned in these calls may not be a true reflection of device control settings. These FCONTROLs include:

- FCONTROL(10), set line speed.
- FCONTROL(11), set line speed.
- FCONTROL(22), return last read time.

- FCONTROL(36), define parity setting.
- FCONTROL(37), set terminal type and speed.
- FCONTROL(40), determine current speed.

The following FCONTROLs are not supported when the remote device is a PC or an HP-UX client. VT will return a CCE condition code even though no device control will take place:

- FCONTROL(23), disable parity generation and checking.
- FCONTROL(24), enable parity generation and checking.
- FCONTROL(38), set terminal type.

Additionally, for FCONTROL(38), not every terminal type will be supported for use with VT.

The following FCONTROLs are not supported when the remote device is an HP-UX client:

- FCONTROL(26), disable binary mode.
- FCONTROL(27), enable binary mode.
- FCONTROL(28), disable user block mode.
- FCONTROL(29), enable user block mode.
- FCONTROL(34), allow printing of ! ! ! when line deletion character is entered.
- FCONTROL(35), prevent printing of ! ! ! when line deletion character is entered.

The **param** of FCONTROL(41) is a 16-bit unsigned integer by reference. The value passed in bits (0:8) is used as a subsystem break character and the value passed in bits (8:8) is used as a read terminator. HP-UX (VT3K) and MPE V VT do not allow the subsystem break character to be redefined. Altering the read terminator, bits (8:8), is supported; however, bits (0:8) should be **[CTRL]Y** to keep existing subsystem break status unchanged, or set to `0` along with bits (8:8) to return to edited mode.

FDEVICECONTROL

The FDEVICECONTROL intrinsic is also supported on MPE/iX systems with VT connections. However, there are also several limitations on the device control operations they perform. The following discussion deals with **parm1** values with 192 specified as **controlcode**.

Results will vary when an application uses an unsupported FDEVICECONTROL call. If the application host is an MPE/iX system, the VT driver will discard the FDEVICECONTROL and return a CCE condition code even though no device control action will take place. In

situations where the application host supports a specific FDEVICECONTROL but the remote device does not, the VT driver will usually return an error to the application.

Unsupported FDEVICECONTROLs include:

parm1	**Action**
1	Specify terminal type or printer type file
2	Set read timeout value
3	Set line speed
4	Set echo
5	Set system break response
6	Set subsystem break response
7	Set timer
8	Return last input time
9	Set parity genration and checking
10	Set parity type
11	Set parity generation and checking
12	Set parity type
14	Set line deletion response
15	Set transparent editing mode
26	Enable/disable XON/XOFF flow control
27	Set XOFF timer value
30	Define read trigger character
32	Define read trigger character
36	Define backspace character
37	Define cancel line character
39	Define type 1 EOR character
40	Define single type 2 EOR character
41	Define subsystem break character
56	Specify data bits per character
64	Ignore parity error on input mode
72	Return PAD line speed
73	Set FCLOSE timeout value
74	Suppress the last form feed

FDEVICECONTROL(28)(block mode types supported) and FDEVICECONTROL(62) (set quiesce I/O) are not supported for use when the remote device is a PC or an HP-UX client.

FDEVICECONTROL(51) (set typeahead mode) is not supported on MPE V remote devices or PCs without appropriate NS software. Additionally, HP-UX (VT3K) supports single echo typeahead only.

FDEVICECONTROL(60) (flush typeahead buffer) and FDEVICECONTROL(61) (bypass typeahead buffer) only support MPE/iX remote devices or PCs with appropriate NS software.

The following FDEVICECONTROLs are supported with MPE/iX hosts and PC clients running appropriate software only. These FDEVICECONTROLs apply to functionality introduced in MPE/iX release 4.5 and PC code NS 2.1 version B.03.00. The FDEVICECONTROLs affected include:

- FDEVICECONTROL(63), set single echo typeahead mode.
- FDEVICECONTROL(65), set inter-byte timer.
- FDEVICECONTROL(66), define multiple type 2 EOR characters (AEOR).
- FDEVICECONTROL(67), define delete (DEL) to backspace (BS) mapping.
- FDEVICECONTROL(68), enable/disable escape sequence read termination.
- FDEVICECONTROL(69), enable/disable suppress echo of read termination characters.

FREAD and FWRITE

In general, VT will support the same size data buffers for FREAD and FWRITE as the DTC. Currently, VT services will accept data buffers up to 4096 bytes long. However, communicating with PC clients is limited to buffers no larger than 3500 bytes. When communicating with a PC client, VT will not support transfers larger than 3500 bytes. Data loss may occur if you exceed the 3500 byte buffer limitation.

Recommended Programming Practices

Application developers are encouraged to use programming practices that will facilitate successful operation in many different network configurations.

The following should be considered:

1. Because the PC only supports a 3500 byte data buffer, applications should only issue FREADs and FWRITEs of less than 3500 bytes even though the DTC and other VT configurations may support larger buffers. If an application attempts to issue an FWRITE larger

than the configuration supports, the data will not be processed. If the remote host detects a buffer size support issue on the local client, it will return a file system error FSERR32 to the application. However, in the case of data transfer between remote host and PC via a third system, the remote host will not detect the buffer size support issue; the application thus receives no error although the data is not processed.

2. Application programs should check for device availability after IO requests. If a terminal logical device becomes unavailable, the application will receive a file system error FSERR24 indicating that the device is no longer available and that the application should terminate. This guideline minimizes unnecessary use of CPU resources.
3. Use FCONTROL rather than FDEVICECONTROL whenever possible. Since not all FDEVICECONTROLs are supported on MPE V systems, an FDEVICECONTROL may not work properly.
4. Use FFILEINFO with *itemnum* value 60 and FCONTROL(39) if you need to determine the network configuration. FFILEINFO with *itemnum* value 60 will return an *item* value corresponding to the terminal type of the device being accessed.

The *item* values returned from FFILEINFO are of the immediate device. For example, a network configuration involving a PAD connection to an HP 3000 and then a remote VT connection to another HP 3000 will return an *item* value 5 (NS Virtual Terminal). FCONTROL(39) will return the terminal type value 24 corresponding to the PAD device.

8 Intrinsics Reference

This chapter presents reference information pertaining to the intrinsics discussed in this manual. Each is described using the same format as that used in the *MPE/iX Intrinsics Reference Manual*; that is, each intrinsic description does the following:

- Gives the intrinsic name.
- Describes the syntax of a call to the intrinsic.
- Summarizes the use of the intrinsic.
- Defines intrinsic parameters.
- And, where applicable:
  - Explains condition codes
  - Gives information on the functional return.
  - Discusses any special considerations.
  - Points to areas of additional discussion.

NOTE This manual applies explicitly to the use of intrinsics in native mode programs running on MPE/iX systems. Certain terms are therefore given specific connotations. The term *word* is used to designate a 32-bit discrete object, while *halfword* designates a 16-bit discrete object, and byte designates an 8-bit discrete object.

Intrinsics and Asynchronous Device Control

Table 8-1 summarizes asynchronous device characteristics, settings and communication modes that may be affected programmatically through use of the intrinsics described in this chapter. It includes a brief explanation of how initial settings are determined. Unless stated otherwise, these settings remain in effect until the device is closed.

Table 8-1 Intrinsics and Device Control

Device Control/Feature	Related Intrinsic(s)	Notes
System Break	FCONTROL(14, 15) CAUSEBREAK	Enabled by default for terminal with active session.
Subsystem Break	FCONTROL(16, 17) XCONTRAP RESETCONTROL FCONTROL(41)	Initially disabled; reset at FCLOSE.
Carriage Control	FCONTROL(1) FWRITE	See Table 9-3, “Selected Carriage Control Directives,” on page 255 for a list of selected carriage control directives
EOR Characters	FCONTROL(25) FCONTROL(41)	Standard EOR character for HP terminals is **[Return]** for character mode reads.
AEOR Characters	FCONTROL(25) FCONTROL(41)	The default is no AEOR.
Echo	FCONTROL(12, 13) FSETMODE(4)	Initially enabled for terminals at system start-up.
Line Deletion Echo	FCONTROL(34, 35)	Initially enabled; reset a FCLOSE
Editing Mode	FCONTROL(26, 27) FCONTROL(41)	Standard editing enabled by default. Reset at FCLOSE.
Transmission Mode	FCONTROL(28, 29)	Terminals operate in character mode by default.
Parity	FCONTROL(23, 24) FCONTROL(36)	Initial parity type determined at port configuration or sensed at logon. Parity checking disabled by default.
Terminal Type	FCONTROL(37) FCONTROL(38) FCONTROL(39)	Default set through port configuration

Device Control/Feature	Related Intrinsic(s)	Notes
Set Read Timeout	FCONTROL(4)	Reset at FCLOSE
Last Read Time	FCONTROL(22)	Set to 0 at FCLOSE
Typeahead	FDEVICECONTROL (51, 60, 61, 63)	Default is set by termtype.

Intrinsic Descriptions

The descriptions that follow are specific to the use of intrinsics for programmatic control of asynchronous devices. Because of this, the explanations in this manual may vary from the more general explanations provided by the *MPE/iX Intrinsics Reference Manual*. In all cases, however, the intrinsics are described according to the conventions that are explained here.

Intrinsic Name

Intrinsic Name A brief summary of the purpose of the intrinsic is listed next to the intrinsic name at the top of the first page of explanation for each intrinsic.

Syntax

The syntax statement contains the complete intrinsic call description. The intrinsic call descriptions are in the format shown below:

```
            16V        I16V     *
FFILEINFO(filenum [,itemnum,item
                  [,itemnum,item
                  [,itemnum,item
                  [,itemnum,item
                  [,itemnum,item]]]]);
```

Required parameters, such as **filenum**, are shown in **boldface**. Optional parameters, such as *itemnum* and *item*, are shown in *italics*. The mnemonics that appear over the parameters indicate their type and whether they are passed by reference (the default) or by value. The mnemonics applicable to the intrinsics described in this manual are listed below, along with their meanings:

MNEMONIC	MEANING
I16	16-bit signed integer
I32	32-bit signed integer
U16	16-bit signed integer
C	character
A	array
UDS	user-defined structure
@32	32-bit address
*	type varies

NOTE A parameter passed by value is indicated by appending V to the mnemonic. Pass by reference is considered to be the default and, consequently, is not marked. All arrays are assumed to be passed by reference.

In the FFILEINFO intrinsic shown previously, there is one required parameter (**filenum**), and a pair of optional parameters that can be repeated up to five times (*itemnum* and *item*). The * over the *item* parameter indicates that the type of the parameter will change according to an associated factor (in this case, the value associated with *itemnum* determines what the *item* parameter will contain and therefore its type).

Use

This paragraph describes how to use the intrinsic. It provides a more complete description than the brief summary given under the intrinsic name.

Functional Return

If applicable to the intrinsic being described, the functional return value will be explained here. If the intrinsic does not have a functional return, this paragraph of the intrinsic description is omitted.

Parameters

All parameters are described, including their data type, whether they are passed by reference or by value, and whether they are required or optional. Any default values for optional parameters are noted here. The description also includes whether the parameter is used to pass information from the program, return information to the program, or both.

For some parameters certain bit settings have particular meanings; when significant, these bit settings and their meanings are described. Bit groups are denoted using the standard notation (*starting bit number:length*). Thus bit (15:1) indicates bit 15; bits (0:3) indicates bits 0, 1 and 2.

In some cases bits within a word are described as being reserved for MPE/iX. You should always set these bits to zero to be sure your program will remain compatible with future MPE/iX releases.

Notes

A special notes section is included in the explanation of the individual FCONTROL functions, and provides additional information regarding their use.

Condition Codes

Where applicable, condition codes are included in the intrinsic descriptions.

Special Considerations

The special considerations portion of the description is omitted unless the intrinsic requires some special circumstances for proper execution, such as Privileged Mode (PM) capability.

Additional Discussion

This paragraph refers to parts of this or other Hewlett-Packard manuals where you can find additional information on the use of the intrinsic.

FCHECK

Used to request specific details about file input/output errors.

SYNTAX

```
        I16V        I16         I16       I32      I16
FCHECK(filenum,fserrorcode,translog,blocknum,numrecs);
```

Use

You can use the FCHECK intrinsic to obtain specific details about an error that occurred during execution of a file system intrinsic. If an intrinsic returns a condition code of CCL or CCG, FCHECK should be used to determine the error code associated with the problem that occurred. You can then use FERRMSG to display the message associated with that error code.

You can determine the error condition of a failed FOPEN call when no file number was returned by setting the **filenum** parameter to 0. In this case only, **fserrorcode** returns valid information.

Do not use FCHECK to determine error conditions of failed calls to HPFOPEN. Error conditions associated with failed HPFOPEN calls are returned in the HPFOPEN **status** parameter.

Parameters

filenum `16-bit signed integer by value (optional)`

The file number of the file for which error information is to be returned. If you do not specify **filenum**, or if you specify 0, FCHECK assumes you want error information concerning the last failed FOPEN call.

fserrorcode `16-bit signed integer by reference (optional)`

Returns a file system error code indicating the type of error that occurred. A return value of 0 indicates that the previous operation was successful or an EOF was encountered. Default: The error code is not returned.

translog `16-bit signed integer by reference (optional)`

Returns the transmission log value that indicates the number of halfwords that were actually read or written.

Default: The transmission log value is not returned.

blocknum `32-bit signed integer by reference (optional)`

For asynchronous devices, this parameter is used to return the number of records read or written to a device since the last FOPEN. A logical record count is returned for spoolfiles; a physical record count is returned for fixed and undefined record files.

numrecs `16-bit signed integer by reference (optional)`

Returns the number of logical records in the block (blocking factor). Blocking factor is not meaningful for asynchronous devices.

Default: The number of logical records is not returned.

Summary of Error Codes

Table 8-2 lists the file system error codes that you are most likely to encounter during intrinsic calls related to asynchronous devices.

Table 8-2 Error Codes

FS Code (decimal)	Description
0	Successful (no errors) or end-of-file (EOF) reached.
20	Invalid operation requested by program (for example, you tried to set the terminal type to -24, which is an invalid entry).
21	Parity error occurred during read.
22	Read limit timer set by user expired. (Timer set via FCONTROL(4).)
24	Device not ready (not on line, modem's Data Set Ready signal not "high", or line disconnected).
27	Block mode read timer expired. This timer is set by the system, and is not user changeable. If no RS character is received before the time expires, the read is terminated.
28	Timing error—Data arrived at controller faster than it could be serviced (data overrun).
31	Read ended by an Additional End-of-Record character (defined via FCONTROL(25)).
32	Software abort of I/O operation occurred (for example, an **ABORTIO** command was entered.
33	Data was lost, or no buffer was available.
42	Operation inconsistent with device type (for example, attempted to set an invalid line speed via FCONTRO(11)).

FS Code (decimal)	Description
68	Insufficient system resources.
70	I/O error while printing header/trailer. Also appears if an FOPEN or FCLOSE failed
95	The read was halted because the terminal user hit **[Break]**. The file system will automatically restart the read when the user enters the **:RESUME** command. Only the file system see the restart, the application program does not.
98	Read timer overflow. Read exceeded maximum capacity of read timer (655.35 seconds).

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic
CCL	Request denied. The file number passed by **filenum** is invalid, or a bounds violation occurred while processing this request (**fserrorcode** is 73).

Additional Discussion

Refer to the discussion of getting file information in *Accessing Files Programmer's Guide*.

FCLOSE

Closes a file.

Syntax

```
                I16V     I16V          I16V
FCLOSE(filenum,disposition,securitycode);
```

Use

The FCLOSE intrinsic terminates access to a file. You should make sure your program issues an FCLOSE for every file it opens. If you have opened multiple files against the same device, you should issue an FCLOSE for each opened file before your program ends. If you should fail to issue an FCLOSE, however, MPE/iX will automatically issue an FCLOSE for any files left open when your process terminates.

Certain device control actions are taken by the device control software at every call to FCLOSE, as shown in Table 8-3. This means that if you have programmatically altered one of these device settings (through the programmatic control listed in the table), the device will return to the system default setting at FCLOSE.

You should note that Table 8-3 shows only those control actions taken by the device control software. Additional changes may result from actions taken by other software modules. For example, should your program disable the system break feature, break will be reenabled on FCLOSE by the Command Interpreter software.

Table 8-3 Device Control Action at FCLOSE

FCLOSE Action	Related Programmatic Control
Disable Read Timeout	Value set by FCONTROL(4)
Disable Subsystem Break	Enabled through FCONTROL(17)
Enable Printing !!! on the deletion	Printing !!! disabled through FCONTROL(35)
Disable Transparent Editing	Enabled through FCONTROL(41)

For most other device settings that can be programmatically altered, you are responsible for making sure that the altered characteristics are returned to their original settings when your program ends or for assuring that the altered characteristics define the way you want the device to act. For example, if you programmatically alter the speed setting of a device, you should return the terminal to its original setting when your program terminates unless you want the new speed setting to be in effect for other files opened against the same device.

When all files have been closed on a device (referred to as device close), and the device is no longer under the control of a program or a session, all device characteristics are returned to those specified by the device's configuration.

NOTE If a disconnected status is received from a terminal your application has programmatically opened, your application should either FCLOSE the terminal or terminate.

Parameters

filenum `16-bit signed integer by value (required)`

The file number of the file to be closed.

disposition `16-bit signed integer by value (required)`

This parameter has no meaning for terminals or serial printers and should be set to 0.

securitycode `16-bit signed integer by value (required)`

This parameter has no meaning for terminals or serial printers and should be set to 0.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request denied. The file was not closed because an incorrect **filenum** was specified. Any outstanding write I/Os that failed (such as buffered writes that are done in background) also cause the FCLOSE to fail.

Additional Discussion

See the discussion of closing files in *Accessing Files Programmer's Guide*.

FCONTROL

Performs control operations on a file or a device.

Syntax

```
                I16V     I16V       *
FCONTROL(filenum,controlcode,param);
```

Use

The FCONTROL intrinsic performs various control operations on a file or on the device on which the file resides. It applies to files on disk and tape as well as to files on terminals and printers. The **controlcode** parameter determines the action taken by the FCONTROL call.

The following pages describe the use of the FCONTROL intrinsic for control of asynchronous devices. Chapters 3 and 4 also include discussions on FCONTROL. For a complete explanation of FCONTROL functions for disk and tape files, refer to the *MPE/iX Intrinsics Reference Manual*.

The characteristics that define the relationship between a specific asynchronous device and the MPE/iX system are initially set through system configuration or through the configured terminal or printer type of the device. When you issue calls to most of the FCONTROL functions, you are temporarily changing the operating characteristics of the device on which the file (specified in **filenum**) resides.

An exception to this applies to calls to FCONTROL using **controlcode** 25 through 29. These calls affect only the file specified in the **filenum** parameter of the call. They do not affect the operating characteristics of the device itself or any other files that might be opened against the same device.

You will generally need to issue several calls to FCONTROL, each specifying an appropriate **controlcode**, to set up the combination of device characteristics that your program requires. However, for most **controlcode** values, you only need to issue one call for a specific characteristic to be in effect for all files opened against a device. Exceptions to this include the following.

Controlcode 4 sets a read timeout value for the next read (timeout value), and **controlcode** 22 returns the time taken for the last read to your program. Each of these timer functions, if desired, must be issued for every read.

A number of FCONTROL functions return information to your program. Some only return information, while others set a new value and return the old value. You can then use the value returned as input to a call which resets the device to its previous setting when your program ends.

Parameters

filenum `16-bit signed integer by value (required)`

Contains the file number of the target devicefile as returned by FOPEN (or HPFOPEN).

controlcode `16-bit signed integer by value (required)`

A code that specifies the control operation to be performed. The codes used for asynchronous device control are listed below. With the exception of **controlcode = 1**, which sends a carriage control directive to a device, these controls apply only to terminals.

1	Sends a carriage control directive to the device.
4	Sets a read timeout value for the next read.
10	Changes the line speed of a device. Both input and output speed are affected.
11	Changes the line speed of a device. Both input and output speed are affected.
12	Enables character echoing.
13	Disables character echoing.
14	Disables the system break function.
15	Enables the system break function.
16	Disables the subsystem break function.
17	Enables the subsystem break function.
22	Obtains the time required for the last read to complete.
23	Disables parity checking.
24	Enables parity checking.
25	Defines additional end-of-record (AEOR) character for terminal input.
26	Disables Binary Mode.
27	Enables Binary Mode.
28	Disables User Block Mode.

29	Enables User Block Mode.
34	Allows printing of ! ! ! when the line deletion character is entered.
35	Prevents printing of ! ! ! when the line deletion character is entered.
36	Defines parity setting.
37	Sets both terminal type and speed.
38	Sets terminal type.
39	Determines current terminal type.
40	Determines current speed setting.
41	Enables transparent editing.
param	`type varies passed by reference (required)` The meaning of **param** depends on the **controlcode** being used. However, no matter what the **controlcode** is, a value must always be assigned to **param**.

NOTE For a number of the **controlcode** functions **param** has no meaning. In those cases it must be coded as a dummy parameter, with a value of 0, because it is a required parameter.

The following pages describe each **controlcode** value. Each description includes the use of **param**, the meaning of condition codes, and any special considerations or additional discussion.

FCONTROL(1)

Parameters

controlcode 1 — Causes a carriage control or mode control directive to be sent to the device.

param `16-bit unsigned integer by reference (required)`

Passes a value representing a carriage control or mode control directive to a non spooled device.

Returns a value representing the prior mode control setting when a mode control directive is specified.

Notes When FCONTROL is issued with a **controlcode** of 1, **param** specifies either a mode control directive or a carriage control directive to be applied to the file specified in **filenum**.

You can use FCONTROL to pass carriage control directives only if the file being output is not spooled. For any files that are written to disk or spooled prior to output, you must use the FWRITE intrinsic to pass carriage control and mode control directives.

The Carriage Control directives passed through FCONTROL are the same as those that are passed through FWRITE. Table 9-3, “Selected Carriage Control Directives,” on page 255 lists the octal codes used for this purpose, along with their corresponding carriage control actions. You will find this table in the explanation of the FWRITE intrinsic later in this chapter.

Carriage control can be passed through FCONTROL for a non-spooled file regardless of whether the file was opened with `CCTL` or `NOCCTL` specified in the FOPEN or HPFOPEN call.

Mode control can also be specified through FCONTROL for non-spooled files. Mode control specifies whether a carriage control directive takes effect before printing (**prespace movement**) or after printing (**postspace movement**). Mode control is selected through specific octal codes from the Carriage Control table. A value of %100 or %400 sets postspace movement, %101 or %401 sets prespace movement.

When FCONTROL is used to specify a mode control directive, an additional call to FWRITE or FCONTROL is required if you also want to specify a carriage control directive.

If **param** contains one of the mode-control directives, a value of 0 or 1 is returned to **param** to indicate the mode setting of the device prior to the call. A value of 0 indicates postspacing, a value of 1 indicates prespacing.

The default mode and carriage control is postspacing with automatic page eject. This applies to all Hewlett-Packard supported subsystems except FORTRAN77/iX and COBOLII/iX, which have prespacing with automatic page eject.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request denied. An error occurred.

Additional Discussion

See the discussion of FWRITE in this manual. See also the *MPE/iX Intrinsics Reference Manual*.

FCONTROL(4)

Parameters

controlcode 4 — Sets a read timeout value for the next read.

param `16-bit unsigned integer by reference (required)`

Passes a timeout value (in seconds) to the device that owns the file whose file number is specified in **filenum**.

Notes A call to FCONTROL with a **controlcode** value of 4 allows you to set a time limit on the next read from a terminal. The timeout value is specified in seconds, with a maximum value of 65535.

Any read that does not complete before the time limit expires is transferred to the user buffer and the read length is returned. A condition code of CCL is returned to the calling program. A call to FCHECK after the timeout returns `Error code 22, Software Timeout`.

If you are setting a timeout value for a read posted against a device connected through a PAD, DTC Telnet, or Telnet/iX, you should take care to add enough time to compensate for any time delays that might be caused by the network.

A timeout value should be used any time your program is reading from an unattended device, to prevent “hangs” that might occur if the device is not ready or a problem exists with the device or its connection to the system. You can also use a timeout value to terminate binary reads. Refer to the discussion of FCONTROL(26,27) for more information on terminating reads in binary mode.

The timeout value is in effect only for the next read from the terminal, and must therefore be reissued for each read. A timeout value of 0 cancels the timeout.

During block mode reads the timer halts when the DC2 character is received. At this point, the block mode read timer is activated by the system. The block mode read timer is set by the system software, and its values are not user changeable.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Failed; timeout value not accepted.

Additional Discussion

See also FCONTROL(26,27) in this manual.

FCONTROL(10, 11)

Parameters

controlcode 10 or 11 — Changes the line speed of a device.

param `16-bit unsigned integer by reference (required)`

Passes a value which specifies a new line speed.

Notes A **controlcode** value of 10 or 11 is used to programmatically change the line speed associated with a device. Both input and output speeds are affected. The value passed in **param** must be expressed in characters per second (1 character per second = 10 bits per second). The choices supported by MPE/iX are 30, 120, 240, 480, 960, and 1920. For direct connect devices on DTC 72MX, 3840 is also valid. These equate to the MPE/iX supported speeds of 300, 1200, 2400, 4800, 9600,19200, and 38400 bits per second. Any other settings will cause the intrinsic to return a condition code of CCL.

The **controlcode** values of 10 and 11 function in an identical manner and can be used interchangeably. The duplication exists because early versions of the HP 3000 (Series II/III) supported a terminal controller (the ATC), which allowed different speeds to be used for input and output on the same device. For devices attached to the ATC controller a **controlcode** of 10 adjusted the input speed while a **controlcode** of 11 adjusted the output speed. Split speeds for input and output are not supported on MPE/iX.

You must be sure the speed associated with a device matches the physical speed setting of the device and of the modem if a modem is part of the connection. If you programmatically change the speed of a device, your program should request that the speed setting be manually changed at the device as well.

If either of these FCONTROL calls is issued against a terminal connected via PAD, DTC Telnet, Telnet/iX, or VT, the call will return a CCE condition code but will actually have no effect on the speed at which the device is transmitting. Additionally, the call will return a meaningless value in **param**.

When a device is initially opened programmatically, the speed is set to the default speed of the device. If a device is opened as a log on device (and the device was configured with speed and parity sensing enabled), the speed setting is sensed from the speed of the first carriage return character received. The default speed of a device is configured through the Node Management Configuration Manager (NMMGR), or through the OpenView DTC Manager, for networks managed by an OpenView workstation. You must reconfigure the device to permanently effect a change to the configured speed.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request failed because the process does not own the logical device, the device is not a terminal/printer, or the speed entered is not supported.

Additional Discussion

See FCONTROL(37) and FCONTROL(40) in this manual.

FCONTROL(12, 13)

Parameters

controlcode 12 — Enables character echoing.

13 — Disables character echoing.

param `16-bit unsigned integer by reference (required)`

Returns a value indicating the previous echo status to the program.

0 = Echo ON

1 = Echo OFF

Notes FCONTROL with a **controlcode** value of 12 or 13 specifies whether or not input echo is enabled at a terminal. Use 12 to turn echo on, 13 to turn echo off. You may also need to use FSETMODE(4), to suppress linefeed on inputs, to turn echo completely off.

When echo is enabled, all characters transmitted to the DTC are “echoed” back and appear on the terminal screen. This is referred to as **DTC echo**, and is initially enabled for all terminals when the MPE/iX system is brought up. All characters are echoed, with the exception of XON, XOFF, NULL, DEL, and DC2.

During binary reads, however, all characters, including XON and XOFF, are passed through as data. Since these characters are not recognized and acted upon as protocol characters, they will be echoed. An XOFF will result in the terminal being suspended as if the XOFF had been sent from the DTC. For this reason, you should make sure that echo is disabled while in binary mode. (An alternative is to disable XON/XOFF flow control at the terminal, but this could result in a loss of data.)

Echo should also be disabled during block mode processing, and **local echo** should be used to cause the terminal itself to write data to the screen. If you are using VPLUS for your block mode applications, this is handled for you automatically. For any other type of block mode, you need to disable echo before the block mode read begins.

You may also want to disable echo if the terminal user is asked to enter data that you do not want to appear on the screen, such as a password or lockword. Additionally, it may be necessary to disable echo if you are connecting certain non-supported devices to an asynchronous port.

Echo is not reset at FCLOSE, so your program should always restore the original echo setting before it ends.

If either of these FCONTROL calls is issued against a terminal connected via DTC Telnet, the call will return a CCE condition code but the echoing of read data will continue.

Echo may be changed locally without the remote host's knowledge in a VT connection. Applications run on the remote side will then receive inconsistent information about echo status. Hence, altering the echo state locally in a VT connection is discouraged.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

Refer to discussions of FCONTROL(26,27) and FSETMODE in this manual.

FCONTROL(14, 15)

Parameters

controlcode 14 — Disables the system break function.

15 — Enables the system break function.

param `16-bit unsigned integer by reference (required)`

This parameter has no meaning for these **controlcode** values. Enter a dummy value of 0 to satisfy the internal requirements of the intrinsic.

Notes A call to FCONTROL with a **controlcode** value of 14 or 15 determines whether or not a terminal will react to a system break request. Use 14 to disable system break, 15 to enable system break.

System break is enabled by default for any terminal on which a session is active. Any time a user presses the **[Break]** key (or the CAUSEBREAK intrinsic is called), MPE/iX will attempt to interrupt processing and place the terminal at the Command Interpreter level (: prompt). Many MPE/iX commands are breakable, as are program commands that invoke subsystems or run user programs.

When system break is enabled and a system break is received, EOR, read timer, terminal mode and echo values are saved by the DTS software, so that they can be restored when normal processing is resumed. If system break is entered during processing of a character mode read or write, data is lost. Any read that is interrupted by a break will be reissued by the system after a Resume command is typed.

Some application programs change the settings of terminals and/or the characteristics of their devicefiles. In such cases it may be undesirable to allow system break processing to occur. For example, you should disable system break when using block mode. Doing so will prevent data loss or corruption that could occur should the **[Break]** key be pressed during a block mode read. It also avoids problems that can occur because your program is unable to return the device to normal operating mode if system break is entered.

Use FCONTROL(14) to disable the system break function before your program enters block mode. Call FCONTROL(15) to reenable system break when block mode processing is completed.

If the **[Break]** key is pressed while system break is disabled no action is taken by any level of software. System break has no effect on a device with no active session.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

See the *MPE/iX Commands Reference Manual*.

FCONTROL(16, 17)

Parameters

controlcode 16 — Disables the subsystem break function.

17 — Enables the subsystem break function.

param `16-bit unsigned integer by reference (required)`

This parameter has no meaning for these controlcode values. Enter a dummy value of 0 to satisfy the internal requirements of the intrinsic.

Notes A call to FCONTROL with a **controlcode** value of 16 or 17 determines whether or not a terminal will react to a subsystem break request. (The default subsystem break character is **[CTRL]Y**.) Use 16 to disable subsystem break, 17 to enable subsystem break.

Subsystem break is initially disabled on all devices; it must be specifically enabled before it can be used. It is available only for devices running sessions, and has no effect on programmatically controlled devices. Subsystem break is reset to the disabled state at FCLOSE.

If enabled, subsystem break allows the terminal user to stop a program local or subsystem local command. It is invoked in standard editing mode by the end-of-medium (EM) character, produced on HP terminals by typing **[CTRL]Y**.

Even when subsystem break is enabled for a terminal connected via PAD, subsystem break must be followed by a **[Return]** to signal the PAD that data is ready to be forwarded.

You can also use FDEVICECONTROL to alter the subsystem break character in either standard or transparent mode. See the discussion of the FDEVICECONTROL intrinsic later in this chapter for the proper parameter settings to use for this purpose.

In transparent editing mode, you are allowed to define a different character to be used to invoke subsystem break by specifying the new character as part of **param** in a call to FCONTROL(41).

Before subsystem break can be successfully enabled, your program must call the XCONTRAP intrinsic, which arms a special trap procedure referred to as the **[CTRL]Y** Trap. The external label of a user written procedure is specified as the value of the **plabel** parameter in the call to XCONTRAP.

The procedure specified must define the steps that will be taken if subsystem break is entered during execution of your program.

Once the **[CTRL]Y** Trap has been armed by XCONTRAP, the terminal user can successfully invoke the subsystem break function.

Whenever a subsystem break is entered, the **[CTRL]Y** Trap is automatically disarmed. Your program must call the RESETCONTROL intrinsic to re-arm the trap when it is ready to allow another subsystem break to occur. If RESETCONTROL is not called after a subsystem break, it is not possible to enter another subsystem break.

The following summarizes subsystem break processing and the intrinsics involved:

1. A user written procedure must be provided to define how the program should act upon receiving a subsystem break.
2. A call to XCONTRAP must be included in the program, specifying the external label of the procedure written in Step 1 as the value of **plabel**. This arms the **[CTRL]Y** Trap.
3. A call to FCONTROL(17) must be included in the program to enable the subsystem break function.
4. If the subsystem break character is received during execution of the program, the procedure specified in the call to XCONTRAP is executed.
5. A call to RESETCONTROL must be executed when the program is ready to receive another subsystem break. (This is usually done in the user written subsystem break handling procedure.)
6. A new subsystem break character may be defined through FCONTROL(41), if the terminal is placed in transparent mode, or through FDEVICECONTROL.
7. If desirable at any time, FCONTROL(16) may be called to disable subsystem break. A call to FCONTROL(17) enables it again.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

See the discussion of RESETCONTROL, XCONTRAP, FCONTROL(41) and FDEVICECONTROL in this manual. See Chapter 5, “Using Subsystem Break,” in this manual for an example illustrating subsystem break processing. See also *Accessing Files Programmer’s Guide, Trap Handling Programmer’s Guide, and Data Types Programmer’s Guide*.

FCONTROL(22)

Parameters

controlcode 22 — Obtains the time used for completion of the last read.

param `16-bit unsigned integer by reference (required)`

Returns the measured time duration of the last read in hundredths of a second.

Notes A call to FCONTROL with a **controlcode** of 22 allows you to read the result of the read duration timer for the last read. The value is returned to your program in hundredths of a second, up to the 16-bit maximum limit of 655.35 seconds.

Since every read is timed on MPE/iX systems, you should be sure your call to this intrinsic follows immediately after the read you want timed with no subsequent reads in between.

FCONTROL(22) should not be confused with FCONTROL(4), which is a read timeout. FCONTROL(22) reports the length of time it took to satisfy a read request, after the read completes. No other action is taken as a direct result of this call.

If this FCONTROL call is issued against a terminal connected via Telnet/iX or VT, the call will return a CCE condition code but no device control action will take place. Any value returned in this call may not be a true reflection of device control settings.

Condition Codes

CCE Request granted.

CCG Request granted, but the result overflowed 16 bits (read took longer than 655.35 seconds).

CCL Request failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

See the discussion on “Timing a Read” in Chapter 3, “Common Device Control Functions,” of this manual.

FCONTROL(23, 24)

Parameters

controlcode 23 — Disables parity generation and checking.

24 — Enables parity generation and checking.

param `16-bit unsigned integer by reference (required)`

This parameter has no meaning for these **controlcode** values. Enter a dummy value of 0 to satisfy the internal requirements of the intrinsic.

Notes A call to FCONTROL with a **controlcode** value of 23 or 24 determines whether or not parity is generated and checked for data transmitted between the system and a terminal. Use 23 to disable parity, 24 to enable parity. FCONTROL(23) and (24) do not apply to printers.

Parity can be used for terminals opened programmatically as well as for those operating in session mode.The default for both session and programmatic devices is to not generate or check parity, but to operate in 8-bit pass-through mode.

When parity is enabled, the Asynchronous Serial Communications software (ASC) generates parity on outgoing data and checks for parity errors on incoming data. After the parity is checked, the parity bit is always set to zero because the program using the data has no need for parity information. Parity checking is handled the same way in block mode as in character mode.

The type of parity that will be used if parity is enabled can be modified through a call to FCONTROL(36). If FCONTROL(23) is used to disable parity after it was previously enabled, a subsequent call to FCONTROL(24) will reenable the same type of parity, unless it was changed through another call to FCONTROL(36).

Because parity requires the eighth bit to be set as a parity bit, it is not possible to use parity with 8-bit character sets. If binary mode is enabled, the parity bit will be passed through as data, and no parity checking will occur.

If enabling parity, make sure that your program also requests the terminal operator to change the physical parity setting of the terminal to the new setting of the ASC software. Additionally, if a user wants the terminal to check parity on incoming data, the local terminal control used to turn parity checking on or off should be set to `ON`.

If either of these FCONTROL calls is issued against a terminal connected via PAD, DTC Telnet, Telnet/iX, or VT (from a PC or HP-UX local client), the call will return a CCE condition code but will actually have no effect on the parity which the device is transmitting. Additionally, the call will return a meaningless value in **param**.

If a parity error is detected by the ASC software, the current read is completed in error, and no read data is returned. The ASC software reports the error to the program reading the data.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

See also discussion of FCONTROL(36) in this manual.

FCONTROL(25)

Parameters

controlcode 25 — Defines Additional End-of-Record (AEOR) character for terminal input.

param `16-bit unsigned integer by reference (required).`

Passes the value of a character to be used as an additional line terminator. The octal or decimal code for the character must be contained in the right byte of the parameter; the left byte is ignored. If the NULL character (%0) is specified, normal (default) line termination conditions are restored.

Notes An FCONTROL call with a **controlcode** value of 25 is used to establish an Additional End-of-Record (AEOR) character, which is then in effect for reads posted against the device that owns the file whose file number is specified in **filenum**. You can only assign one AEOR character at a time. If you need more than one AEOR, use FDEVICECONTROL instead.

Normally, character mode reads using standard editing are terminated by a carriage return (**[Return]**), the standard EOR character for HP terminal types. The system echoes the carriage return and sends a linefeed to the terminal, so that the cursor is positioned at the beginning of the next line, ready for the next read. Even if an AEOR character has been assigned by a call to FCONTROL(25), the EOR character will still terminate reads in the usual way.

When a read is terminated by an Additional End-of-Record character, the AEOR character is included in both the data and in the byte count. No carriage return or linefeed is sent to the terminal. The read terminates with an error condition which indicates that the Additional End-of-Record character has been encountered. A call to FCHECK returns an error code of 31. To recover from this error, your program can delete the AEOR character from the input data, subtract one from the byte count of the read, and send out a carriage return and linefeed to place the cursor in the proper place for the next read.

An AEOR character designated by FCONTROL(25) is also recognized as a line terminator during reads in transparent editing mode, along with a user defined EOR character that replaces **[Return]** as the normal EOR character.

See the discussion of FCONTROL(41) later in this chapter for more information on terminating reads in transparent mode.

To disable the Additional End-of-Record character, call FCONTROL(25) again specifying 0 as the value of **param**.

There are a number of characters that will not be recognized if used as AEOR characters, as listed in Table 8-4. You will receive no error message if you use one of these characters, but the character will not terminate a read if entered.

Table 8-4 Characters Not Recognized If Used as AEORs

ASCII Character	Terminal Key	Octal Code	Hex Code
NUL (Null)	**[CTRL]@**	% 0	$ 0
DC1 (XON)	**[CTRL]Q**	% 21	$11
DC3 (XOFF)	**[CTRL]S**	% 23	$13
Current Subsystem Break	**[CTRL]Y** (by default)[a]	% 31	$19
DEL (Rubout)	**[DEL]**	% 177	$7F

a. In transparent mode, you define the subsystem break character through FCONTROL(41). It may also be defined using FDEVICECONTROL.

NOTE

You should also avoid using a number of other characters which have special meaning to MPE/iX, specifically carriage return, linefeed, cancel (**[CTRL]X** by default), backspace, DC2, and escape. While these characters will be recognized and processed as AEOR characters, their use could yield unpredictable and often unsuccessful results. It is strongly recommended that you avoid using these characters unless the specific needs of your application make it absolutely necessary to do so.

In addition, **[CTRL]A** should not be used as an AEOR character at the console.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

See FCONTROL(41) and FDEVICECONTROL in this manual.

FCONTROL(26, 27)

Parameters

controlcode 26 — Disables Binary Mode

27 — Enables Binary Mode

param `16-bit unsigned integer by reference (required)`

This parameter has no meaning for these **controlcode** values. Enter a dummy value of 0 to satisfy the internal requirements of the intrinsic.

Notes An FCONTROL call with a **controlcode** value of 26 or 27 determines whether or not a terminal is operating in binary editing mode for the file specified in **filenum**. Use 26 to disable binary editing, 27 to enable binary editing.

These calls are not supported for terminals connected via DTC Telnet or via VT from an HP-UX local client (VT3K). If FCONTROL(26) is issued to a device connected by DTC Telnet, it will be ignored. If FCONTROL(27) is issued to a device connected via DTC Telnet the next FREAD posted to the device will return an error condition, until binary mode is disabled with FCONTROL(26). Binary editing should not be used on a terminal connected via DTC Telnet or VT from an HP-UX local client. The call will return a CCE condition code but all subsequent reads will return an error until binary editing is disabled. If FCONTROL(27) is issued to a Telnet/iX device when the local device is an HP-UX Telnet client, the user must set the HP-UX client into binary (using the HP-UX Telnet client toggle command) before the FCONTROL(27) is issued to the Telnet/iX Server. If the HP-UX client is not set into binary before the FCONTROL(27) is issued, the FCONTROL will return with CCL. The user must disable binary on the HP-UX client before binary is disabled on the Telnet/iX Server with FCONTROL(28).

When binary editing is enabled, no special characters are recognized, and no special character processing is done. All characters are considered to be data and are passed through without any terminal control actions

being taken. No carriage return or linefeed is sent to the terminal following a binary read. On writes, all carriage control directives are ignored. Binary editing mode is initially disabled by default.

Unlike most FCONTROL calls, binary mode does not actually take effect until a read is posted to the device following the FCONTROL call. If you are setting binary mode for both reads and writes you should post a 0-byte read immediately after the call to make sure binary mode is set.

Because no special character processing occurs in binary mode, any ASCII DC3 (XOFF) character passed in the data will be echoed to the terminal, and the terminal will be suspended as if an XOFF were sent from the DTC. This problem can be avoided by calling FCONTROL(13) to disable character echoing prior to entering binary mode. Alternatively, you can turn off XON/XOFF flow control at the terminal by sending the appropriate escape sequences. You should be aware, however, that disabling flow control may result in data being lost.

Binary editing is useful for transferring 8-bit data to and from a terminal. Because all 8 bits are considered to be data, binary editing is not compatible with parity checking, and parity must be disabled. In addition, since no special character processing is possible in binary mode, it cannot be used for block mode transfers.

In session mode, system break restores standard editing at the terminal. If **:RESUME** is entered following the break binary mode will be restored when the first read is posted following the **:RESUME**. FCONTROL(26) must be called to disable binary editing.

Binary reads are terminated by byte count, read timeouts, and error conditions. CCE status is returned for reads terminated on the byte count as specified in the FREAD, READ, or READX call. CCL status will be returned on timeouts and errors. Reads terminated on timeout will return data entered before the timeout occurred. For reads terminated on error, no data will be returned.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Failed due to an error.

Additional Discussion

See discussion of FCONTROL(4), FCONTROL(12,13) and FCONTROL(41) in this manual.

FCONTROL(28, 29)

Parameters

controlcode 28 — Disables User Block Mode.

29 — Enables User Block Mode.

param `16-bit unsigned integer by reference (required)`

This parameter has no meaning for these **controlcode** values. Enter a dummy value of 0 to satisfy the internal requirements of the intrinsic.

Notes A call to FCONTROL with a **controlcode** value of 29 enables user block mode processing, while a call using **controlcode** 28 disables user block mode processing.

Under system default conditions block mode processing occurs through a method called HP block mode. In HP block mode the system controls the block mode handshake and the user program need not concern itself with data transfer protocol. The MPE/iX host transmits the normal read trigger character, a DC1, to inform the terminal that the system is ready to receive data. The terminal responds by sending a DC2 character, which informs the system that the next data transfer will be a block mode transmission. The terminal then waits for the host to send back a second DC1 to trigger the read.

When user block mode is enabled it becomes possible for your program to intervene in the handshake. A user block mode transaction begins in the same way an HP block mode transaction begins, with host transmission of a DC1. At this point, the user application takes over control of the handshake and waits for the DC2. Your program can then perform additional terminal control functions, such as positioning the terminal cursor, performing status requests, or allocating additional space for buffers before continuing with the data transfer. When your program is ready to receive the block mode data, it must issue an additional read. This read will actually result in a DC1 being sent, thereby completing the handshake.

Some additional considerations apply when transmitting data in block mode. The terminal must be configured to perform in a manner compatible with the expectations of the system and the application requesting the block mode transfers. You should pay particular attention to how `Line/Page(D)`, `InhHndShk(G)` and `Inh DC2(H)` are set when configuring a terminal for block mode processing.

Block mode processing is not supported with terminal type 18.

User block mode is not supported on terminals connected through DTC Telnet, Telnet/iX (from an HP-UX Telnet client) or VT (from an HP-UX local client (VT3K)). If posted to a device connected via DTC Telnet, these calls will return a CCE condition code, but no device control action will occur.

NOTE Data overruns may occur during block mode transfers. Your program must check for successful completion of each read operation and retry as required. Use of timers during block mode reads is strongly encouraged, since a data overrun occurring when the last character is read would otherwise cause the port to “hang”. The standard block mode read timer does not operate when user block mode protocol is enabled.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

See the reference manuals for the terminals connected to your system.

FCONTROL(34, 35)

Parameters

controlcode 34 — Allows printing of ! ! ! when line deletion character is entered.

35 — Prevents printing of ! ! ! when line deletion character is entered

param `16-bit unsigned integer by reference (required)`

This parameter has no meaning for these **controlcode** values. Enter a dummy value of 0 to satisfy the internal requirements of the intrinsic.

Notes An FCONTROL call with a **controlcode** value of 35 suppresses output of the line deletion response (! ! !, carriage return and linefeed) at the terminal when the line deletion character (normally **[CTRL]X**) is entered. A call using **controlcode** 34 turns this output back on.

In character mode, with standard editing in operation, the computer outputs ! ! ! CR/LF whenever the line deletion (cancel) character is entered. This is referred to as **[CTRL]X** echo, since **[CTRL]X** is the default line deletion character.

You can cause **[CTRL]X** echo to be suppressed by calling FCONTROL(35). MPE/iX will still delete the data in the input buffer, but no ! ! ! CR/LF will be issued.

[CTRL]X Echo is initially enabled by default and is reset to the enabled state at FCLOSE.

For devices connected via PAD, the line deletion response is never printed in response to the line deletion character. If issued to such a device, these calls will return a CCE condition code, but no device control action will occur.

For devices connected via VT from an HP-UX local client, these **controlcode** values are not supported.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

None.

FCONTROL(36)

Parameters

controlcode 36 — Defines parity setting.

param `16-bit unsigned integer by reference (required)`

Passes a value (0, 1, 2, 3, or 4) which represents a specified parity condition.

Returns the value of the previous setting.

Refer to Table 8-5 for a definition of the condition represented by each value.

Notes An FCONTROL call with a **controlcode** value of 36 is used to specify the type of parity generation and checking that will be done if parity is enabled for a terminal.

When a terminal is initially opened certain parity conditions exist, as determined by the terminal type and port configuration, and whether the port is opened by an operator logon or by a user program. These parity conditions can be changed through a call to FCONTROL(36), specifying a number that corresponds to the desired parity as the value of **param**. Hewlett-Packard provides four parity options, called odd, even, ones and zeros. Table 8-5 shows the results of each of these options on input and output data, along with the **param** value associated with each option.

When FCONTROL(36) is called, the parity option specified replaces whatever option was previously associated with the terminal, and the previous option is returned to the call as the new value of **param**. This makes it possible to determine a terminal's parity setting even with parity disabled. You can also use the returned value to restore parity to its original setting when your program ends, since MPE/iX does not reset parity when a file is closed.

Once you have set the parity option, you can call FCONTROL(24) to enable parity. Your program should also request the terminal operator to change the terminal's parity setting to the new setting of the software. Additionally, if parity is to be checked on

incoming data, the local terminal control that determines whether or not parity is checked should be set to check parity.

Parity cannot be used with 8-bit character sets.

This call is ignored for devices connected via PAD, DTC Telnet, Telnet/iX, or VT. If issued against such a device it will return a condition code of CCE, but no device control action will occur.

Table 8-5 Parity Settings with FCONTROL(36)

Value of param	Parity Type	Results if Enabled
0	Zeros	No parity checked on input Eighth bit set to 0 on output
1	Ones	No parity check on input Eighth bit set to 1 on output
2	Even	Even parity checked on input Even parity generated on output
3	Odd	Odd parity check on input Odd parity generated on output
4	None	No parity checked on input No parity checked on output

Condition Codes

CCE Request granted.

CCG Not returned by this intrinsic.

CCL Failed because the file number specified did not belong to this process, the device is not a terminal, or an invalid value was used for **param**.

Additional Discussion

See also discussion of FCONTROL(23,24) in this manual.

FCONTROL(37)

Parameters

controlcode 37 — Sets both terminal type and speed.

param `16-bit unsigned integer by reference (required)`

Passes a value which specifies a new line speed and terminal type to be associated with an asynchronous port. Bits (0:11) specify the speed, bits (11:5) specify the terminal type.

Notes This use of the FCONTROL intrinsic stems from an early point in the development of Hewlett-Packard operating systems when it was necessary to allocate a terminal before it could be opened programmatically. This is no longer necessary, but the intrinsic is still in use today and allows you to programmatically set both terminal type and speed for a terminal through a single call. Both of these values can be set individually through other FCONTROL calls.

When you use FCONTROL(37), **param** passes the desired speed and terminal type values. The speed is represented in characters per second and is contained in bits (0:11). The terminal type is contained in bits (11:5).

You must be sure the speed associated with a device matches the physical speed setting of the device and of the modem, if a modem is part of the device connection. If you programmatically change the speed of a device your program should request that the speed setting be manually changed at the device as well.

You cannot use FCONTROL(37) to set the terminal to an unsupported speed or terminal type. Attempting to do so will cause an error condition of CCL. You cannot use FCONTROL(37) with user-defined terminal types. You must call FDEVICECONTROL to specify the terminal type when using a terminal type defined through the Workstation Configurator.

If issued against a terminal connected via PAD, DTC Telnet, or Telnet/iX, this call will change the terminal type setting, but will have no impact on how the device actually operates. A CCE condition code will be returned.

This call is unsupported for devices connected via VT. If used, the VT driver will return a CCE condition code, but no device control action will take place.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request failed because the process does not own the logical device, the device is not a terminal, or the speed or terminal type entered is not supported.

Additional Discussion

See discussions of FCONTROL(11) and FCONTROL(38) in this manual.

FCONTROL(38)

Parameters

controlcode 38 — Sets terminal type.

param `16-bit unsigned integer by reference (required)`

Passes the value if the system defined terminal type to be associated with an asynchronous port.

Notes FCONTROL with a **controlcode** value of 38 allows you to change the terminal type associated with a port to another system defined terminal type. The new terminal type will be in effect for the port until you change it through another FCONTROL call, or until the device is closed. When the device is closed, the port is reset to its configured default.

If you change the terminal type of a terminal running an interactive session, you should be sure to reset it to the previously assigned terminal type when your program ends, unless you want the new terminal type to be in effect for other files opened on the device. To be sure you are able to reset to the previous terminal type, your program should call FCONTROL(39) to determine the current terminal type before calling FCONTROL(38) to change to the new setting. (Note that the current terminal type may be different from the default (configured) terminal type.)

The terminal type specified by FCONTROL(38) must be a system defined terminal type currently supported on MPE/iX systems. If any other value is used, the intrinsic will return an error condition of CCL.

You cannot use FCONTROL to change to or from a user defined terminal type. You must use FDEVICECONTROL to perform this function with terminal types defined through Workstation Configurator.

If issued against a terminal connected via PAD, DTC Telnet, or Telnet/iX, this call will change the terminal type setting, but will have no impact on how the device actually operates. A CCE condition code will be returned.

If issued against a terminal connected via VT from an MPE/iX or MPE V local client, this call will change the terminal type setting and the characteristics of the terminal type supported by VT. The **controlcode** value is not supported for VT connections from a PC or HP-UX local client.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request failed because the process does not own the logical device, the device is not a terminal, or the terminal type entered is not supported.

Additional Discussion

See also the discussions of FCONTROL(37) and FCONTROL(39) in this manual. Refer to Chapter 7, “Programming for PAD, DTC Telnet, Telnet/iX Server, and VT Devices,” for more information on terminal type specifications for devices connected via PAD or DTC Telnet.

FCONTROL(39)

Parameters

controlcode 39 — Determines current terminal type.

param `16-bit unsigned integer by reference (required)`

Returns the terminal type currently associated with an asynchronous port.

Notes A call to FCONTROL with a **controlcode** value of 39 allows you to determine the terminal type that is currently associated with an asynchronous port. You can then use this information to return the port to its original setting after you have changed the terminal type through FCONTROL(37), FCONTROL(38), or FDEVICECONTROL.

Do not confuse the current terminal type returned by this intrinsic with the default terminal type specified during system configuration through NMMGR. Changing the terminal type during logon or through the system intrinsics can cause the current terminal type to be different from the configured setting. To reset to the configured terminal type, use FCONTROL(37) specifying 0 as the value of **param**. (Note that FCONTROL(37) will restore the default line speed as well.)

Closing the device also restores its settings to configured default values.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

See also discussions of FCONTROL(37) and FCONTROL(38) in this manual.

FCONTROL(40)

Parameters

controlcode 40 — Determines current speed setting.

param `16-bit unsigned integer by reference (required)`

Returns the line speed currently associated with an asynchronous port in characters per second.

Notes A call to FCONTROL with a **controlcode** value of 40 allows you to determine the line speed at which an asynchronous port is currently operating. The speed is specified in characters per second. You can use the information to return the port to its original speed setting after you have changed the line speed through FCONTROL(37) or FCONTROL(11).

Do not confuse the current line speed returned by this intrinsic with the default line speed specified during system configuration through NMMGR. Changing the line speed through the system intrinsics can cause the current line speed to be different from the configured line speed. To reset the line speed of the port to the default value you can use FCONTROL(37) specifying 0 as the value of **param**. (Note that FCONTROL(37) will restore the default terminal type as well.)

Closing the device also restores its settings to configured default values.

If issued against a terminal connected via PAD, DTC Telnet, Telnet/iX, or VT, this call will return a CCE condition code. The value returned in **param** will be 120 (for 1200 baud) but this value has no meaning for DTC Telnet, or PAD connections. For Telnet/iX or VT connections, the value returned in **param** will typically be 960 (for 9600 baud) but this value has no meaning.

Condition Codes

CCE Request granted.

CCG Not returned by this intrinsic.

CCL Failed because the file number specified did not belong to this process or the device is not a terminal.

Additional Discussion

See also discussions of FCONTROL(11) and FCONTROL(37) in this manual.

FCONTROL(41)

Parameters

controlcode 41 — Enables transparent editing.

param `16-bit unsigned integer by reference (required)`

Passes a value to be used as a subsystem break character in bits (0:8) and a value to be used as a read terminator in bits (8:8).

Returns the values previously assigned for use as subsystem break and read terminator characters.

Notes An FCONTROL call with a **controlcode** value of 41 will place a terminal in transparent editing mode. Also called unedited mode, this facility allows most special characters to be read and treated as data. Unlike binary mode, however, which allows no special character processing to occur, transparent mode allows a small subset of special characters to retain their meaning. These characters are listed in Table 8-6.

In addition to the characters shown in Table 8-6, two other characters have a special meaning to the system during transparent mode processing. The characters to be assigned these meanings are specified in the call through the value entered in **param**.

Bits (0:8) of **param** are used to specify the character that will replace the subsystem break character. Any character not otherwise defined as a special character in transparent mode, including the normal subsystem break character (**[CTRL]Y**), can be used. If this value is set to 0, subsystem break is disabled.

Bits (8:8) specify the character that will replace the line termination character (EOR) for the transparent mode read. Any character not otherwise defined as a special character in transparent mode may be used.

While binary mode reads terminate only on byte count or a read timeout, transparent mode reads can be terminated in any of the following ways:

- The read encounters the End-of-Record (EOR) character defined in **param**. This character functions like the **[Return]** character does in normal reads. The EOR is echoed but is stripped from the user data. No carriage return or line feed is sent to the terminal.
- The read encounters an Additional End-of-Record (AEOR) character defined through a previous call to FCONTROL(25). The read completes with error END OF LINE (FSERR 31), and the AEOR character is passed with the input data and included in the byte count. No carriage return or linefeed is sent to the terminal.
- The expected byte count specified in the READ, READX, or FREAD call is reached. The read is terminated normally, and no carriage return or linefeed is sent to the terminal.
- The read limit timer set through a call to FCONTROL(4) expires. The read terminates with error SOFTWARE TIMEOUT (FSERR 22), and the data entered before the read timed out is returned.

Table 8-6 Special Characters for Transparent Editing

Key	Function
AEOR Character	The AEOR character defined through a previous call to FCONTROL(25). The read terminates in error, and a call to FCHECK returns an `fserrorcode` of 31. The AEOR character is passed with the data and included in the byte count.
[Break]	Causes a system break (when enabled) and transfers control to MPE/iX. If **:RESUME** is entered transparent editing is resumed.
[CTRL]Q (DC1 or XON)	The “resume output” character of the XON/XOFF handshake. This character is stripped from the input data when XON/XOFF protocol is enabled.
DC2	When input as first character, it is stripped out. A DC1 ([CTRL]Q) is then written to the terminal. When not input as the first character, DC2 is a data character.
[CTRL]S (DC3 or XOFF)	The “halt output” character of the XON/XOFF handshake. This character is stripped from the input data when XON/XOFF protocol is enabled.

Break processing, if not disabled, occurs in transparent mode the same as in normal editing mode. During the break the terminal will operate in standard editing mode. The terminal will return to transparent mode if a **:RESUME** is entered.

Subsystem break processing also occurs as in normal editing mode, except that the subsystem break character is defined through the FCONTROL(41) call.

Because DC1 and DC2 (if the DC2 is the first byte of data) are recognized as special characters, transparent editing can be successfully used during block mode reads.

NOTE

When several files are open on a device at the same time (multiple FOPEN calls were issued), transparent editing will be in effect for all of the files if it was explicitly enabled for any one of them.

If you enable binary editing, transparent editing is automatically overridden. When you leave binary mode, however, transparent mode will still be enabled. Binary mode will only be enabled for the file on which it is enabled. Other files will continue to operate in the mode, normal or transparent, that was set before binary was enabled.

Parity processing is the same in transparent mode as in standard mode.

Standard editing is restored by calling FCONTROL(41) with the value of **param** set to 0. Any FCLOSE call issued against the terminal will also restore standard editing.

It is possible to enable transparent editing for devices connected via PAD. However, since the PAD will still expect to see a **[Return]** as the data forwarding character, data will not be forwarded from the terminal immediately after the characters defined as the subsystem break or EOR characters are typed. The terminal user will need to press **[Return]** to cause data to be transmitted, or wait for the data forwarding timer to expire if it has been set. The host software will still recognize and act on the characters defined in **param**.

If this call is issued against a terminal connected via VT from an MPE V or HP-UX local client, the subsystem break character may not be redefined. Bits (0:8) should be the existing subsystem break character to keep status unchanged or set to 0 to return to edited mode.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Failed because the file number specified did not belong to this process, the device is not a terminal, or an illegal value was passed in **param**.

Additional Discussion

See also discussions of FCONTROL(4), FCONTROL(16,17), FCONTROL(25), FCONTROL(26,27) and FSETMODE in this manual. See Chapter 7, “Programming for PAD, DTC Telnet, Telnet/iX Server, and VT Devices,” for more information on controlling PAD and DTC Telnet devices.

FDEVICECONTROL

Provides control operations to a printer, terminal, or spooled device file.

Syntax

```
                    I16V    UDS     I16V      I16V
DEVICECONTROL(filenum,buffer,length,controlcode,
                    U16V    U16V      U16
                  parm1,parm2,fserrorcode),
```

Use

The FDEVICECONTROL intrinsic allows you to perform a variety of control functions on devicefiles. The function performed by a specific FDEVICECONTROL call is determined by the values you set for its parameters.

This manual describes the functions performed when the **controlcode** parameter is set to a value of 192. This particular **controlcode** is used to apply various control directives to a devicefile.

A complete list of FDEVICECONTROL options is in Table 8-7. For **controlcode** values other than **controlcode** 192, see the *MPE/iX Intrinsics Reference Manual*.

Table 8-7 Valid FDEVICECONTROL Controlcode Values

Code	Operation Performed
128	Character set selection.
129	Logical page activation/deactivation request.
130	Relative pen displacement.
131	Absolute pen move.
132	Define job characteristics.
133	Define physical page.
134	Download/delete character set.
135	Download/delete form.
136	Download logical page table.
137	Download multicopy from overlay table.
138	Download/delete Vertical Format Control (VFC).
139	Download/delete a picture.

Code	Operation Performed
140	Page control.
141	Clear environment.
142	Reserved for MPE/iX.
143	Load the default environment.
144	Print picture.
145	End of job.
146	Device extended capability mode.
192	Device control operations.
193	Record processing information for NRJE spoolfiles.

The specific action that will be performed by a call to FDEVICECONTROL using 192 as a value for **controlcode** is determined by the value of the **parm1** parameter. The applicable **parm1** values and their meanings are described in the parameter descriptions that follow.

Note that all but one of the functions performed apply only to asynchronous terminals; **controlcode** 192 is not used for control of DTC serial printers. (The exception occurs when a **parm1** value of 1 is used to specify a printer type file.) The **parm2** parameter is used to indicate the Read/Write access desired for each call. That is, should the call replace old settings with new values, return information to your program, or both set new values and return old ones.

The **parm2** access values allowed with each **parm1** control function are shown in Table 8-8, along with a brief description of the control function being performed.

In Table 8-8 an R indicates that only Read access is allowed, while R/W indicates that Read, Write, or both Read and Write access are allowed.

Table 8-8 Allowable Access Value for Device Control

parm1 Value	Access Allowed	Device Control Operation
1	R/W	Specify terminal type or printer type file.
2	R/W	Set read timeout value for the next read.
3	R/W	Set the line speed for the device.
4	R/W	Set echo on or off at terminal.
5	R/W	Set system break response on or off at a terminal.
6	R/W	Set subsystem break response on or off at a terminal.

parm1 Value	Access Allowed	Device Control Operation
8	R	Obtain the time used for completion of the last read.
9	R/W	Set parity generation and checking on or off.
10	R/W	Set type of parity.
11	R/W	Set parity generation and checking on or off.
12	R/W	Set type of parity.
14	R/W	Set line deletion response.
15	R/W	Set transparent editing mode.
26	R/W	Device XON/XOFF enable.
27	R/W	Set XOFF timer value.
28	R	Block mode types supported.
29	R/W	Define block mode alert character.
30	R/W	Define block mode trigger character.
32	R/W	Define read trigger character.
36	R/W	Define backspace character.
37	R/W	Define cancel in character.
39	R/W	Define Type 1 EOR character.
40	R/W	Define Type 2 EOR character (AEOR).
41	R/W	Define subsystem break character.
51	R/W	Set typeahead mode.
52	R/W	Form feed allowed in output data.
53	R/W	Define form feed replacement character.
55	R/W	Select backspace response action.
56	R/W	Specify data bits per character.
57	R	Obtain subsystem break character.
60	R/W	Flush typeahead buffer.
61	R/W	Bypass typeahead buffer.
62	R/W	Quiesce I/O.
63	R/W	Single Echo Typeahead Mode.
64	R/W	Ignore Parity Error on Input Mode.
65	R/W	Inter-Byte Timer

parm1 Value	Access Allowed	Device Control Operation
66	R/W	Define Multiple Type 2 EOR characters (AEOR).
67	R/W	Treat Delete Character (DEL) like Backspace (BS).
68	R/W	Escape Sequence Read Termination.
69	R/W	Suppress echo of Read Termination Characters.
72	R	Return PAD line speed.
73	R/W	Set FCLOSE timeout value.
74	R/W	Suppress last form feed.
76	R/W	Host and Device XON/XOFF enable.
77	R/W	Set the number of stop bits.

See previous chapters for examples of using the FDEVICECONTROL intrinsic to perform device control operations.

Parameters

filenum `16-bit signed integer by value (required)`

Contains the file number of the target devicefile as returned by FOPEN (or HPFOPEN).

buffer `user-defined structure (required)`

Specifies and returns the device control value of the control directive specified in **parm1**. Examples of information specified or returned by the **buffer** parameter include character sets, forms or other printer control information, and device control actions such as line speed and parity settings.

The value of **parm2** indicates whether **buffer** specifies the Read/Write access, where you can specify a device control status, receive a device control status, or both. **buffer** is usually a 16-bit integer by reference; however, for some values of **parm1**, (1 and 66) **buffer** is a character array.

length `16-bit signed integer by value (required)`

Specifies the length of **buffer** in halfwords or bytes. Positive values of **length** indicates the length in halfwords (2 bytes), and negative values indicates the length in bytes. With **controlcode** 192, **length** will usually be set to 1.

The exception to this occurs when **parm1** is set to 1 to specify a new terminal type or printer type file for use. In this case, the length of the character array containing the name of the file to be used is specified in the **length** parameter.

controlcode 16-bit signed integer by value (required)

Specifies the code number of the operation to be performed. This section of the manual assumes that **controlcode** is 192 (device control directive). No **controlcode** values other than 192 can be used for asynchronous terminals. The permitted values for **controlcode** and their associated functions are listed in Table 8-7.

parm1 16-bit unsigned integer by value (required)

Specifies the control directive for the device file specified in **filenum**. The permissible values and their associated meanings are described later in the "Device Control Directives" section.

parm2 16-bit unsigned integer by value (required)

When **controlcode** is 192, indicates whether the call will read or write device control value. Use the following values to indicate the desired access for the call:

1 =	Return current value in **buffer** (Read)
2 =	Set item to value in **buffer** (Write)
3 =	Return current value in **buffer** (Read/Write) and set item to value in **buffer**

Table 8-8 shows the type of access allowed for each parm1 value.

fserrorcode 16-bit unsigned integer by reference (required)

Returns a file system error code number if an error occurs. If no error occurs, **fserrorcode** is set to zero.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request denied. An error occurred.

Device Control Directives

The following is a list of all possible values for the **parm1** argument, the control directive associated with that value, and a description of the operation of that control directive.

Value of parm1	Action
1	**Specify terminal type or printer type file**. This directive specifies a new terminal type or printer type file which describes the device to the device driver. The terminal type or printer type file must be in the format used by the TTUTIL utility. The **buffer** parameter contains the filename of the terminal type or printer type file. The **length** parameter contains the length of **buffer**. If this call is issued against a device connected via PAD, DTC Telnet, Telnet/iX or VT, a CCE condition code is returned and the terminal type or printer type file associated with the device will be changed. However, the device will continue to operate as if no device control action took place.
2	**Set read timeout value for the next read.** This directive specifies a timeout value to apply to the next read requested. **buffer** contains the number of seconds to wait before terminating the read. The read data is transferred to the user buffer and the read length is returned. It is the same as FCONTROL (4). A value of `0` means do not timeout. This device control effects only the next read. If the timer expires before the read completes, the read returns error status 22 (software timeout). This use of the FDEVICECONTROL intrinsic is the same as a call to FCONTROL (4). For reads posted to devices connected via PAD, DTC Telnet, or Telnet/iX, be sure to add enough time to compensate for any delays caused by the network. This call is not supported on devices connected via VT.
3	**Set line speed for the device.** This directive sets the line speed for the device. The speed is the same for input and output. It is the same as FCONTROL (10, 11, 40). Valid **buffer** values are (in

characters per second): 30, 120, 240, 480, 960, and 1920, (and 3840 for DTC 72MX direct connected devices only).

If this call is issued against a device connected through a PAD, DTC Telnet, or Telnet/iX connection, the terminal control will be ignored, and a meaningless value (120 for 1200 baud) will be returned in the call. For VT connections, the value returned will typically be 960 (for 9600 baud) but this value has no meaning.

4 **Set echo at a terminal.**

This directive enables and disables echoing of read data by the DTC. It is the same as FCONTROL (12, 13). If **buffer** equals `0`, read echo is set to off. If the **buffer** value is `1`, read echo is set to on.

This directive is not supported for use with VT. Changing echo on the local client of a VT connection may result in the remote host receiving false echo status; hence, it is not recommended for VT connections.

5 **Set system break response.**

This directive enables and disables the response to the system **[Break]** key at the terminal. It is the same as FCONTROL (14, 15). A **buffer** value of `0` disables system **[Break]**. A **buffer** value of `1` enables system **[Break]** response.

6 **Set subsystem break response.**

This directive enables and disables the response to the subsystem break character at the terminal. It is the same as FCONTROL (16, 17). A **buffer** value of `0` disables subsystem break response. A **buffer** value of `1` enables subsystem break response.

For terminals connected via PAD, the subsystem break must be followed by a **[Return]** to signal the PAD that data is ready to be forwarded.

8 **Obtain time used for completion of last read.**

This directive returns the amount of time, in seconds, required for the last read to complete. Since all reads are timed on MPE/iX systems there is no need to enable the timer. However, you should be sure that your call to this intrinsic is before the next read. Only Read access is allowed for this request. It is the same as FCONTROL (22). This directive is not supported for Telnet/iX or VT connections.

9, 11 **Set parity generation and checking.**

These directives enable and disable parity generation and checking between the DTC and the device. The functions of **parm1** values 9 and 11 are identical. It is the same as FCONTROL (23, 24).

A **buffer** value of 0 disables parity, and all eight bits of each character are passed through untouched. A **buffer** value of 1 enables parity that is set when **parm1** is 10 or 12, or when FCONTROL(36) is used. Both input and output parity are the same.

If this call is issued against a device connected through a PAD, DTC Telnet, Telnet/iX or VT connection, a CCE condition code is returned but there is no impact on the terminal.

10, 12 **Set type of parity.**

These directives enable or disable the kind of parity used when parity checking is enabled. The functions of **parm1** values 10 and 12 are identical. If parity checking is disabled, the setting of this directive is ignored. It is the same as FCONTROL (36).

There are five possible settings for **buffer**:

0	indicates no parity checking (eighth bit always set to 0)
1	indicates no parity checking (eighth bit always set to 1)
3	indicates even parity
4	indicates no parity checking
5	indicates even parity with eight bit data length
6	indicates odd parity with eight bit data length

If this call is issued against a device connected through a PAD, DTC Telnet, Telnet/iX or VT connection, a CCE condition code is returned but there is no impact on the terminal.

14 **Set line deletion response.**

This directive enables and disables the response to the line deletion character when received from the terminal. The response is ! ! ! !. **[CTRL]X** is the default line deletion character. A **buffer** value of 0 disables transmission of ! ! ! !, and a value of 1 enables

transmission. If the line delete response is disabled, line deletion still takes place, but ! ! ! is not echoed. This directive is the same as FCONTROL (34, 35).

If this call is issued against a device connected through a PAD, DTC Telnet or VT connection, a CCE condition code is returned but there is no impact on the terminal. Line deletion characters are controlled locally by PAD devices.

15 **Set transparent editing mode.**

This directive enables and disables transparent (unedited) mode, and sets the EOR and the subsystem break characters. It is the same as FCONTROL (41).

A **buffer** value of 0 disables transparent (unedited) mode, and enables normal editing mode, with **[CTRL]Y** as the subsystem break character and carriage return as the EOR character.

If **buffer** is non-0, the low order byte is treated as the subsystem break character and the high order byte is treated as the EOR character:

```
bits (0:8) = subsystem break character

bits (8:8) = EOR character
```

The special characters listed below may not be used as the subsystem break or EOR character. A value of 0 in the subsystem break field disables the subsystem break facility. The EOR character must be specified when setting transparent mode. A value of 0 in the EOR field and a non-zero value in the subsystem break field will still enable transparent mode but no EOR will be recognized. Transparent editing mode treats most special characters as data, but the following special characters retain their meaning:

AEOR character Alternate End of Record

[Break]	System Break
[CTRL]R	HP Block mode alert
[CTRL]Q	XON
[CTRL]S	XOFF

If the device is connected via PAD, the terminal user must enter a **[Return]** to signal to the PAD to transmit data. The subsystem break and EOR characters defined in this directive will still be recognized by the host software. If the device is connected via VT from an

MPE V or HP-UX local client, the subsystem break character must be set to the existing subsystem break character when enabling transparent mode.

26 **Device XON/XOFF enable.**

This directive enables and disables XON/XOFF flow control between the device and the DTC. A **buffer** value of `1` enables device XON/XOFF, and a value of `0` disables XON/XOFF. When device XON/XOFF is enabled, the DTC stops sending data to the device when it receives XOFF and resumes when it receives XON. The DTC does not send XON/XOFF characters to the host. When device XON/XOFF is disabled, the XON and XOFF characters are passed to the host as data. When XON/XOFF flow control is disabled, data overruns can occur.

If this control is issued to a terminal connected via PAD, DTC Telnet, Telnet/iX or VT, it will return a CCE condition code, but no terminal control action will occur.

27 **Set XOFF timer value.**

This directive enables or disables the XOFF timer<. The timer is enabled by passing a positive value, representing a time limit in seconds, in the **buffer** parameter. When enabled, the XOFF timer causes a warning message to be sent to the console if a device is XOFFed for a time exceeding the set limit. A **buffer** value less than or equal to 0 disables the XOFF timer.

If this control is issued to a terminal connected via PAD, DTC Telnet, Telnet/iX or VT, it will return a CCE condition code, but no terminal control action will occur.

28 **Block mode types supported.**

This directive returns a value indicating the type of block mode supported by the driver. The possible return values are shown below:

`7 =`	both line and DTC style page block mode
`15 =`	PAD terminal supporting page block mode

This directive is not supported for terminals connected via DTC Telnet or via VT from a PC or HP-UX local client.

29 **Define block mode alert character.**

This directive specifies the character that signals HP block mode transfers. The normal alert character is DC2. Note that this call only defines the alert character; it does not enable HP block mode. The low order byte of **buffer** contains the new alert character.

If issued against a terminal connected via PAD, DTC Telnet or VT, this call will return a CCE condition code, but the device control action will not occur.

30, 32 **Define read trigger character.**

These directives specify the character the DTC sends to trigger a read on the device. The normal trigger character is DC1. The low order byte of **buffer** contains the read trigger character. If the low order byte of **buffer** contains a NUL character, there is no trigger character.

If issued against a terminal connected via PAD, DTC Telnet or VT, this call will return a CCE condition code but the device control action will not occur.

36 **Define backspace character.**

This directive specifies the backspace character in normal editing mode. The low order byte of **buffer** contains the new backspace character.

If issued against a terminal connected via VT, this call will return a CCE condition code but no device control action will take place.

37 **Define cancel line character.**

This directive specifies the line deletion character for normal editing mode. The low order byte of **buffer** contains the new line deletion character. **[CTRL]X** is the default cancel line character.

If issued against a terminal connected via VT, this call will return a CCE condition code but no device control action will take place.

39 **Define type 1 EOR character.**

This directive specifies the EOR (End-of-Record) character to be used in normal or transparent editing mode. The high order byte of **buffer** (value*256) contains the new Type 1 EOR character. A NUL character disables the EOR character. **[Return]** is the default EOR character.

The following special characters retain their meaning:

AEOR character Alternate End of Record

[Break]	System Break
[CTRL]R	Same as XON if it is first character in the record
[CTRL]Q	XON
[CTRL]S	XOFF

If a **[Return]** is issued via PAD, and the EOR is not **[Return]**, the EOR will be set and a timer (.5 second) will be enabled that will timeout and transmit the data if the user enters the EOR but does not enter a **[Return]**. If **[Return]** is the EOR character, the system will wait for a **[Return]** and the timer (.5 second) will not time out.

If issued against a terminal connected via VT, this call will return a CCE condition code but no device control action will take place.

40 **Define single type 2 EOR character (AEOR).**

This directive specifies an AEOR (Additional End of Record) character and enables its use. It is the same as FCONTROL (25). The high order byte of **buffer** (value*256) contains the new AEOR character. Use a NUL character to disable the AEOR character. You can use any character as the AEOR character; however, avoid the following characters because their use may yield unpredictable results:

[CTRL]@	NUL (null)
[CTRL]Q	XON
[CTRL]S	XOFF
[CTRL]Y	the Subsystem break by default
[Return]	carriage return
[CTRL]J	line feed
[CTRL]X	cancel line character by default
[CTRL]H	backspace by default
[DC2]	block mode alert character by default
[ESC]	escape

The Define Single Type 2 EOR character (AEOR) control directive (FCONTROL 25 or FDEVICECONTROL 40) and Define Multiple Type 2

EOR characters (AEOR) control directive (FDEVICECONTROL 66) interact closely. A call to set a single AEOR sets the specified character as the AEOR character, and all other AEOR characters are cleared. A call to set multiple AEORs sets all of the characters specified in the array, and it clears the single AEOR.

This call is not supported for use with VT.

41 **Define subsystem break character.**

This directive specifies the subsystem break character. The high order byte of **buffer** (value*256) contains the new subsystem break character. The default subsystem break character is **[CTRL]Y**.

If this request is issued for a terminal connected via PAD, the host software recognizes the new subsystem break character, but the terminal user must type a **[Return]** following the subsystem break to signal the PAD to transmit the data.

If issued against a terminal connected via VT, this call will return a CCE condition code but no device control action will take place.

51 **Set typeahead mode.**

This directive enables and disables typeahead mode. A **buffer** value of `0` disables typeahead mode. A **buffer** value of `1` enables typeahead mode. Typeahead mode is disabled by default.

When a terminal is in typeahead mode, up to 224 characters may be entered before a read is posted. The device driver stores the typeahead data in a typeahead buffer until it receives a read. The terminal operator may enter data in advance of the actual posting of a read. When a terminal is not in typeahead mode, the terminal operator may enter data only when a read has been posted to the terminal.

If issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. This directive is not supported for VT connections to MPE V remote devices or PCs without appropriate NS software. Additionally, HP-UX (VT3K) supports single echo typeahead only.

Refer to Chapter 6, “Typeahead Mode,” for more information on using typeahead mode.

52 **Form feed allowed in output data.**

This directive enables and disables the substitution of the form feed character in the output stream. If **buffer** is 1, the device driver does not substitute the form feed character when it is encountered in the carriage control of terminals. If **buffer** is 0, the device driver substitutes any form feed characters encountered in the carriage control of terminals with form feed replacement character.

53 **Define form feed replacement character.**

This directive specifies the form feed replacement character. The low order 8 bits of **buffer** contains the new form feed replacement character. Note that form feed in carriage control information will be replaced; form feed in data will not. NUL is the default form feed replacement character.

55 **Select backspace response action.**

This directive specifies the action the DTC will take when it receives a backspace character. The valid values for **buffer** are shown below:

1	remove character from input and back cursor up one space
5	remove character from input and erase character backspace, space, backspace)

If issued against a terminal connected via PAD, DTC Telnet or VT, this call will return a CCE condition code, but the device control action will not occur.

56 Specify data bits per character.

This directive specifies whether 7 or 8 bits will be used for each character. If **buffer** is 7, then 7 data bit characters are used, and the current parity setting (directives 9 and 11 or FCONTROL(36)) controls parity generation and checking.

If **buffer** is 8, then 8 bit characters are used, and parity checking is disabled. The current parity setting is reenabled if you switch back to 7 data bits.

If issued against a terminal connected via PAD, DTC Telnet, Telnet/iX or VT, this call will return a CCE condition code, but the device control action will not occur.

57 **Obtain subsystem break character.**

This directive returns the current subsystem break character. It only returns a value, and may not be used to change the subsystem break character.

60 **Flush typeahead buffer.**

This directive flushes the typeahead buffer. If **buffer** equals `1`, the device driver flushes the typeahead buffer, and any data in the buffer is lost. This request is valid for the next read only. A value of `0` has no effect on the typeahead buffer. This request is only valid if typeahead is enabled. (See **parm1 = 51**)

If issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. If issued against a terminal connected via VT to an MPE V system, this call will return an error.

61 **Bypass typeahead buffer.**

This directive specifies that the next read shall be taken from the terminal and not the typeahead buffer. If **buffer** is `1`, the next read will receive data directly from the device; the read bypasses any data in the typeahead buffer. The data in the typeahead buffer will not be flushed, and can be obtained by subsequent reads. The request is valid for the next read only. A **buffer** value of `0` allows the typeahead buffer to function normally. (See **parm1 = 51**)

You may use this directive to obtain status information directly from a device. This request is only valid if typeahead is enabled.

If issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. If issued against a terminal connected via VT to an MPE V or HP-UX system, this call will return an error.

62 **Quiesce I/O.**

This directive ensures that all outstanding nowait I/O operations have completed. If **buffer** is 1, the call to FDEVICECONTROL does not complete until all nowait I/O operations have been fulfilled for the device. A **buffer** value of 0 has no effect, and returns immediately.

If issued against a terminal connected via DTC Telnet or via VT to an HP-UX or PC local client, this call will return an error.

63 **Single echo typeahead mode.**

This directive specifies that the device driver shall withhold the echoing of typeahead data until a read is posted for the data. This facility prevents passwords and other data from being echoed at inappropriate times. This directive is not valid if typeahead mode is disabled. A **buffer** value of `1` enables single echo typeahead mode. A **buffer** value of `0` restores normal typeahead mode.

If issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. If issued against a terminal connected via VT to an MPE V local client, this call will return an error. Also, this directive is supported for VT connections to an MPE/iX host or PC running appropriate software only.

64 **Ignore parity error on input mode**.

This directive causes the device driver to ignore parity errors on input data. If **buffer** is `1`, the device driver ignores parity errors on data received from the terminal. If **buffer** is `0`, the device driver reports all parity errors; this is the default setting.

If this directive is issued against a terminal connected via PAD, DTC Telnet, or Telnet/iX, this call will return a CCE condition code, but the device control action will not occur. This directive is not supported for use with VT connections and will return an error if used

65 **Inter-byte timer.**

This directive sets the inter-byte timer. When a read is posted with the inter-byte timer enabled, the DTC waits indefinitely to receive the first character of data. The inter-byte timer starts when the DTC receives a character from the terminal. The DTC restarts the timer each time it receives another character. If the timer reaches the specified duration before the DTC receives the next character, the read completes. If typeahead mode is enabled, the timer does not start until all data has been read from the typeahead buffer. The inter-byte timer and the read timer are independent of one another and either may complete a given read.

This control directive is incompatible with the Escape Sequence Read Termination control directive (**parm1 = 68**). If both are enabled, the second FDEVICECONTROL will return an error.

A positive value in **buffer** indicates the inter-byte timer value in tenths (.1) of a second. That is, a value of `10` in **buffer** sets the inter-byte timer to expire in 1 second. If **buffer** is `0`, the inter-byte timer is deactivated.

If this directive is issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. This directive is not supported for VT connections from an MPE V or HP-UX local client and will return an error if used. Also, this directive is supported for VT connections to an MPE/iX host or PC running appropriate software only.

66 **Define multiple type 2 EOR characters (AEOR).**

This directive specifies up to 16 Alternate End of Record characters. The **buffer** argument is a byte array containing up to 16 characters. The following characters may not be used as AEOR characters:

[CTRL]@	NUL
[CTRL]Q	XON
[CTRL]S	XOFF
[CTRL]Y	the Subsystem break by default

The characters shown below should not be used as AEOR characters because they may yield unpredictable results:

[Return]	carriage return
[CTRL]J	line feed
[CTRL]X	cancel line character by default
[CTRL]H	backspace by default
[DC2]	block mode alert character by default
[ESC]	escape

The **length** argument contains the length of the array. If **buffer** contains 0s or if **length** is 0, all AEOR characters are canceled.

The Define Single Type 2 EOR character (AEOR) control directive (**parm1 = 40**) and Define Multiple Type 2 EOR characters (AEOR) control directive (**parm1 = 66**) interact closely. A call to set a single AEOR sets the specified character as the AEOR character, and all other AEOR characters are cleared. A call to set multiple AEORs sets all of the characters specified in the array, and it clears the single AEOR.

If this directive is issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. This directive is not supported for VT connections from an MPE V or HP-UX local client and will return an error if used. Also, this directive is supported for VT connections to an MPE/iX host or PC running appropriate software only.

67 **Delete (DEL) to backspace (BS) Mapping.**

This directive causes the DTC to treat the DEL character (ASCII 127) like a backspace. The DTC treats both backspace and delete as defined by the Backspace Response Action control directive (**parm1 = 55**). If the backspace would normally be part of the read data, the DTC replaces the delete with backspace. If **buffer** is `1`, the DTC treats the delete character like the backspace (for example, echoes backspace-space-backspace). If **buffer** is `0`, the DTC processes the delete character normally in binary and unedited mode; otherwise it ignores the data. The backspace character is not affected by this directive.

If this directive is issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. Also, this directive is supported for VT connections to an MPE/iX host or PC running appropriate software only.

68 **Escape sequence read termination.**

This directive terminates reads after an escape (ESC) character is encountered. It is used to capture entire escape sequences (usually associated with a key) from the terminal without performing multiple reads. When you enable this mode, the DTC sets the inter-byte timer after receiving an escape character and then terminates the read.

The DTC sets the inter-byte timer to the time it takes to transmit 2 characters at the current baud rate (see **parm1 = 3**). The read returns with all characters received with less than a 2 character delay between them. This ensures that the read included all of the automatically generated escape sequence characters. Because the DTC uses the inter-byte timer to time the read termination, this control directive is incompatible with the Inter-Byte Timer (**parm1 = 65**). If both are enabled, the second FDEVICECONTROL will return an error.

A **buffer** value of `1` enables escape sequence read termination. A **buffer** value of `0` disables escape sequence read termination.

If this directive is issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. This directive is not supported for VT connections from an MPE V or HP-UX local client and will return an error if used. Also, this directive is supported for VT connections to an MPE/iX host or PC running appropriate software only.

69 **Suppress echo of read termination characters.**

This directive causes the DTC not to echo read termination characters to the terminal. This directive is particularly useful with special AEOR characters which may make cursor positioning uncertain. (For example, you might use a “?” character as a help request.)

If **buffer** is `1`, the DTC does not echo read termination characters (EOR and all AEOR characters). If **buffer** is `0`, the DTC echos all characters according to the setting of the Set Echo at Terminal control directive (**parm1 = 4**).

If this directive is issued against a terminal connected via PAD or DTC Telnet, this call will return a CCE condition code, but the device control action will not occur. This directive is not supported for VT connections from an MPE V or HP-UX local client and will return an error if used. Also, this directive is supported for VT connections to an MPE/iX host or PC running appropriate software only.

72 **Return PAD line speed.**

This directive will return the current line speed of a terminal connection, including PAD connections. The line speed in characters per second will be returned in **buffer**.

NOTE

When the connection is first established, the DTC sends the line speed to the host. If the user changes the line speed locally (between the PAD and the terminal), this FDEVICECONTROL will not reflect the change.

73 **Set FCLOSE timeout value.**

This directive establishes a timeout value for an FCLOSE request. If the timer expires before the close is terminated successfully, the connection is aborted.

This call is not supported for use with Telnet/iX or VT. The Telnet/iX or VT driver will return a condition code even though no device control action will take place.

WARNING

Data loss can occur when the connection is aborted due to the timer expiration.

This FDEVICECONTROL should be called immediately before the FCLOSE. Sending any I/O commands to the LDEV after this FDEVICECONTROL and before the FCLOSE will cause the port to become inoperative.

buffer value:

1–2147	timer is set to value in seconds; values less than 1 or greater than 2147 are invalid

These are the error conditions returned in the **fserrorcode** field:

68	internal error—most likely a parameter error; verify that timer value is within range 1–2147 seconds
32	software abort—timer expired and connection aborted, data loss may have occurred
0	successful—connection closed normally

This call is not supported for use with DTC Telnet, Telnet/iX or VT. The driver will return a condition code even though no device control action will take place.

74 **Suppress the last form feed.**

For printers only.

This directive will suppress the form feed normally sent by the serial printer driver during close processing. This FDEVICECONTROL is often used when an application sends its own form feed at the end of a document. The combination of the application's form feed and the serial printer driver's form feed produces an extra blank page at the end of the document. The use of this FDEVICECONTROL will suppress the serial printer driver's form feed. This FDEVICECONTROL must be called for every document since it is automatically turned off at the end of printing.

The **buffer** parameter is not used in this FDEVICECONTROL.

This call is not supported for use with DTC Telnet, Telnet/iX or VT. The driver will return a condition code even though no device control action will take place.

76 **Host and Device XON/XOFF enable.**

This directive enables and disables XON/XOFF flow control between device and the DTC.

Device XON/XOFF:

When device XON/XOFF is enabled, the DTC stops sending data to the device when it receives XOFF. It resumes it on receiving an XON. However, when the XON/XOFF is disabled, the XON and XOFF characters received by the DTC are simply passed to the host without any interpretation.

Host XON/XOFF:

When Host XON/XOFF is enabled, the DTC sends an XOFF character to the device when its receive buffer becomes full. Then, when the data is transmitted to the CPU, the DTC sends an XON character to the device and restarts the input data flow.

There are four possible flow control settings for buffer values:

`0`	disable both device and host flow control
`5`	enable device flow control
`9`	enable host flow control
`13`	enable both device and host flow control

77 **Set the number of stop bits.**

This directive specifies the number of stop bits used.

There are two possible settings for buffer:

`0`	indicate the number of stop bits to 1
`1`	indicate the number of stop bits to 2

Additional Discussion

See the discussions of the various uses of the FCONTROL intrinsic. See also the *MPE/iX Intrinsics Reference Manual*, the discussion of providing control operations to devices in *Accessing Files Programmer's Guide*, and the *IFS/3000 Reference Manual*. For examples of using FDEVICECONTROL, see this manual.

9 Intrinsics Reference (cont)

This chapter presents reference information pertaining to the intrinsics discussed in this manual. Each is described using the same format as that used in the *MPE/iX Intrinsics Reference Manual*; that is, each intrinsic description does the following:

- Gives the intrinsic name.
- Describes the syntax of a call to the intrinsic.
- Summarizes the use of the intrinsic.
- Defines intrinsic parameters.
- And, where applicable:
  - Explains condition codes
  - Gives information on the functional return.
  - Discusses any special considerations.
  - Points to areas of additional discussion.

NOTE

This manual applies explicitly to the use of intrinsics in native mode programs running on MPE/iX systems. Certain terms are therefore given specific connotations. The term *word* is used to designate a 32-bit discrete object, while *halfword* designates a 16-bit discrete object, and byte designates an 8-bit discrete object.

FERRMSG

Returns a message corresponding to an FCHECK error number.

Syntax

```
                I16         CA          I16
FERRMSG(fserrorcode,msgbuffer,msglength);
```

Use

You can use FERRMSG in your program to obtain the error message associated with a specific error number returned in a call to FCHECK. The FCHECK error number is used as the value of **fserrorcode** in the call to FERRMSG. The textual message associated with the error number is then returned in **msgbuffer**, with the length of the message returned in **msglength**.

A typical error handling routine might consist of the following steps:

1. Check condition code.
2. If CCL or CCG, call FCHECK to obtain specific details about the error that occurred, including the error number.
3. Call FERRMSG using the error number obtained in the FCHECK call as the value of **fserrorcode**.

You can then print the error message, display it to the terminal screen or console, or take whatever action is appropriate.

Parameters

fserrorcode `16-bit signed integer by reference (required)`

Passes an error code returned by the FCHECK intrinsic indicating which message is to be returned in **msgbuffer**.

msgbuffer `character array (required)`

Returns the error message associated with **fserrorcode**. To contain the longest possible message, **msgbuffer** must be a minimum of 72 bytes long. Note that all character arrays must start on a halfword boundary.

msglength `16-bit signed integer by reference (required)`

Returns the length of the error message contained in **msgbuffer**. The length is returned as a positive byte count.

Condition Codes

CCE	Request granted.
CCG	Request denied. No error message exists for the **fserrorcode**.
CCL	Request denied. The **msgbuffer** address was out of bounds, **msgbuffer** was not large enough, or **msglength** was out of bounds

Additional Discussion

See the discussion of returning error messages to your program in *Accessing Files Programmer's Guide.*

FFILEINFO

Provides information about the characteristics of a file.

Syntax

```
                I16V    I16V    *
FFILEINFO(filenum[,itemnum,item
                    [,itemnum,item
                    [,itemnum,item
                    [,itemnum,item
                     [,itemnum,item]]]]);
```

Use

The FFILEINFO intrinsic returns information about a file located on any device. The file must be opened by the calling process at the time of the FFILEINFO call. You specify the type of information you want to access by using an item number that corresponds to that information as the value of *itemnum*. The file information requested is then returned to your program in the *item* parameter that constitutes the other half of the *itemnum/item* pair.

The *itemnum* and *item* parameters must always be specified in pairs. You can retrieve up to 5 types of information in one FFILEINFO call by specifying 5 *itemnum/item* pairs.

Table 9-1 shows some of the information options available through a call to FFILEINFO. Only the subset of options most applicable to files in asynchronous devices are listed here. For a complete list of available options, see the MPE/iX Intrinsics Reference Manual.

Parameters

filenumn 16-bit signed integer by value (required)

The file number of the file about which information is requested.

itemnum 16-bit signed integer by value (optional)

Passes the item number (in any order) of an information option. You can request up to 5 options in one call.

item type varies (optional)

Returns information about the file as pertains to the option specified in the related **filenum**.

Table 9-1 Selected Item Values Returned by FFILEINFO

Item No.	Info. Returned In Item	Type
1	Actual file designator of the file, in the format: `filename.groupname.accountname`	Character array, at least 28 bytes in length. Unused bytes are filled with right justified blanks and a nameless file returns an empty string. The fully qualified name of the file referenced by **filenum** is returned as the value of this **itemnum**. Only names which can be expressed using MPE only semantics are returned by this **itemnum**. if the name of the object referenced by **filenum** cannot be expressed using MPE name semantics a CCL condition code is returned. Calling FCHECK for **filenum** after this error occurs, will result in error.
2	File options (refer to the summary figure listing **foptions** bit settings).	16-bit unsigned integer by reference.
3	Access options (refer to the summary figure listing **aoptions** bit settings).	16-bit unsigned integer by reference.
5	Device type indicated by: bits (8:8) = device type.	16-bit signed integer by reference.
6	Local device number of the device where the file label resides.	16-bit unsigned integer by reference.
12	Number of logical records passed to and from your program during the current access.	32-bit signed integer by reference.
13	Number of buffered physical I/O operations performed since the last FOPEN/HPFOPEN call (records).	16-bit signed integer by reference.
43	Environment file name.	Character array, at least 36 bytes in length.
50	Real device number of the file.	16-bit unsigned integer by reference.
60	Terminal type, defined as follows: 0–File's associated device is not a terminal. 1–Configured as standard hardwired terminal. Could also be TELNET/iX device. 2–Configured as a modem-connected terminal. Could also be TELNET/iX device. 4–DTC PAD Terminal. 5–NS Virtual Terminal.	16-bit signed integer by reference.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Access or calling sequence error.

Additional Discussion

See also *Getting System Information* and the *MPE/iX Intrinsics Reference Manual*. See the discussion of FOPEN in this manual for a description of **aoption** and **foption** values.

FGETINFO

Returns access and status information about a file.

Syntax

```
                I6V        CA              U16       U16
FGETINFO(filenum,formaldesignator,foptions,aoptions,
                I16       I16       U16     U16
        ,devtype,ldevnum,hdaddr,
                I16      I32      I32     I32       I32
        filecode,lrecptr,EOF,filelimit,logcount,
                I32       I16       U16       I16
        physcount,blksize,extsize,numextents,
                I16          CA       I32
        userlabels,creatorid,labaddr);
```

Use

The FGETINFO intrinsic returns access and status information about a file located on any device. The file must be opened by the calling process at the time of the FGETINFO call. When using this intrinsic you need only specify those parameters that return the information options you want. The default in all cases is that information that is not explicitly requested is not returned.

Only those parameters that return information that applies to asynchronous devices are described below. For an explanation of all the parameters, you should see the *MPE/iX Intrinsics Reference Manual*.

Parameters

filenum 16-bit signed integer by value (required)

The file number of the file about which information is required.

formaldesignator character array (optional)

Returns the actual designator of the file being referenced in the following format:

```
filename.groupname.accountname
```

The *formaldesignator* array must be at least 28 bytes in length. When the actual designator is returned, unused bytes in the array are filled with

blanks on the right. A nameless file returns an empty string. Note that all character arrays must start on a halfword boundary.

The fully qualified name of the file referenced by **filenum** is returned as the value of this parameter. Only names which can be expressed using MPE only semantics are returned. If the name of the object referenced by **filenum** cannot be expressed using MPE name semantics, a CCL condition code is returned. Calling FCHECK after this error will result in an error.

foptions `16-bit unsigned integer by reference (optional)`

Returns information about the *foption* characteristics that are set through the FOPEN or HPFOPEN call. The bit settings that designate each of these characteristics are listed below. Note that not all of the ***foptions*** are meaningful for files on asynchronous devices.

Bits (14:2) —Domain *foption*.

Domain is meaningless for asynchronous devicefiles. However, if file redirection was anticipated at the time the file was opened, these bits may have been set.

`=00`	The file is a new file.
`=01`	The file is an old permanent file.
`=10`	The file is an old temporary file.
`=11`	The file is an old file.

Bit (13:1)—ASCII/binary **foption**.

Note that all files not on disk are treated as ASCII.

`=0`	Binary.
`=1`	ASCII.

Bit (10:3)—Default file designator *foption*.

`=000`	The actual and formal file designators are the same.
`=001`	The actual file designator is `$STDLIST`.
`=010`	The actual file designator is `$NEWPASS`.
`=011`	The actual file designator is `$OLDPASS`.
`=100`	The actual file designator is `$STDIN`.
`=101`	The actual file designator is `$STDINX`.

=110 The actual file designator is $NULL.

Bits (8:2)—Record format ***foption***.

These bits are not meaningful for asynchronous devices since all records are of undefined length. However, if redirection was anticipated, “fixed-length” may have been specified.

=00 Fixed-length records.

=01 Variable-length records.

=10 Undefined-length records.

=11 Spoolfile.

Bit (7:1)—Carriage control ***foption***.

=0 No carriage control character expected.

=1 Carriage Control character expected.

Bit (6:1)—MPE/iX tape label ***foption***.

Not meaningful for asynchronous devices, but may have been set if file redirection was anticipated.

=0 Nonlabeled tape.

=1 Labeled tape.

Bits (5:1)—Disallow **:FILE** equation ***foption***.

If this bit is set to 1, any **:FILE** equation intended to override the specifications set through the FOPEN or HPFOPEN call will be ignored.

=0 Allow **:FILE** command

=1 Disallow **:FILE** command.

Bits (2:3)—File type ***foption***.

Will be set to 000 for asynchronous devicefiles.

=000 Standard file (STD).

=001 Keyed Sequential Access Method file (KSAM).

=010 Relative I/O file (RIO).

=100 Circular file (CIR).

=110 Message file (MSG).

Bits (0:2)—Reserved for MPE/iX. These should be set to 0.

aoptions 16-bit unsigned integer by reference (optional)

Returns information about the ***aoption*** characteristics that are set through the FOPEN or HPFOPEN call. The bit settings that designate each of these characteristics are listed below. Note that not all of the **aoptions** are meaningful for files on asynchronous devices.

Bits (12:4)—Access type ***aoption***.

The type of access allowed users of the file. Refer to Table 9-2 for a summary of how these settings apply to terminals and printers.

=0000	Read access only.
=0001	Write access only.
=0010	Write access only, but previous data in file not deleted.
=0011	Append access only.
=0100	Input/output access.
=0101	Update access.
=0110	Execute access.
=0111	Execute/read access.
=1000	No access.

Bit (11:1)—Multirecord ***aoption***.

Allows a block of data that exceeds the file's physical record size to be transmitted as several records. This option is available for use with terminals, but not for printers.

=0	Select non-multirecord mode.
=1	Select multirecord mode.

Bit (10:1)—Dynamic locking ***aoption***

Not meaningful for asynchronous devices.

Bit (8:2)—Exclusive access ***aoption***

Specifies whether you have continuous exclusive access to this file. This option does not apply to printers.

=00	Default.
=01	Exclusive access.
=10	Semi-exclusive access.

=11 Shared access.

Bit (7:1)—Inhibit buffering ***aoption***

Not meaningful for asynchronous devices.

Bit (5:2)—Multiaccess mode ***aoption***

Not meaningful for asynchronous devices.

Bit (4:1)—NOWAIT I/O ***aoption***

Indicates whether or not NOWAIT I/O has been specified for the file. NOWAIT I/O allows the accessor to initiate an I/O request and to have control returned before the completion of the request.

=0 NOWAIT I/O not in effect.

=1 NOWAIT I/O in effect.

Bits (3:1)—File copy access ***aoption***

Specifies whether or not a file may be treated as a standard sequential file rather than as a file of its own type.

=0 File must be accessed as its own type.

=1 File may be accessed as standard sequential file.

Bits (0:3)—Reserved for MPE/iX.

Table 9-2 Access Types

Access Types	Value	Used with a Terminal	Used with a Printer
Read only	0000	OK.	Returns File System Error 40: Operation Inconsistent with Access type.
Write only (Write/Delete)	0001	OK.	OK.
Write (save) only	0010	OK.	Executed as **Write only.**
Append only	0011	OK.	Executed as **Write only.**
Read/Write	0100	OK.	Executed as **Write only.**
Update	0101	OK.	Executed as **Write only.**
Execute	0110	Returns File System Error 40: Operation Inconsistent with Access type.	Returns File System Error 40: Operation Inconsistent with Access type.

lrecsize 16-bit signed integer by reference (optional)

Returns the logical record size associated with the file. This parameter is subject to record size limits imposed by MPE/V based systems, and is maintained only for backward compatibility.

If the record size of the specified file exceeds MPE/V limits, a zero is returned.

devtype `16-bit signed integer by reference (optional)`

Returns the type and subtype of the device being used, where bits (0:8) indicate device subtype, and bits (8:8) indicate device type. Note that subtypes do not apply for terminals and printers.

If the file is not spooled the *devtype* returned is actual. The same is true if a spool file was opened via the logical device number. However, if an output file is spooled and was opened by device class name, *devtype* will return information about the first device in the class, which may not be the device actually being used.

ldevnum `16-bit unsigned integer by reference (optional)`

Returns the logical device number of the port (and device) associated with the file.

hdaddr `16-bit unsigned integer by reference (optional)`

Returns 2048. This value is meaningless on MPE/iX based systems; it is maintained only for backward compatibility.

filecode `16-bit signed integer by reference (optional)`

Not meaningful for asynchronous devices.

lrecptr `32-bit signed integer by reference (optional)`

Not meaningful for asynchronous devices.

EOF `32-bit signed integer by reference (optional)`

Not meaningful for asynchronous devices.

filelimit `32-bit signed integer by reference (optional)`

Not meaningful for asynchronous devices.

logcount `32-bit signed integer by reference (optional)`

Returns the total number of logical records passed to and from the program during the current access of the file.

physcount `32-bit signed integer by reference (optional)`

Returns the total number of physical input/output operations performed within this process against the file since the last FOPEN or HPFOPEN call.

blksize `16-bit signed integer by reference (optional)`

Not meaningful for asynchronous devices.

extsize `16-bit unsigned integer by reference (optional)`

Not meaningful for asynchronous devices.

numextent `16-bit signed integer by reference (optional)`

Not meaningful for asynchronous devices.

userlabels `16-bit signed integer by reference (optional)`

Not meaningful for asynchronous devices.

creatorid `character array (optional)`

Not meaningful for asynchronous devices.

labaddr `32-bit signed integer by reference (optional)`

Not meaningful for asynchronous devices.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request denied. An error occurred.

Additional Discussion

See the discussion of requesting file information in *Getting System Information* and the *MPE/iX Intrinsics Reference Manual*. See also discussions of FFILEINFO and FOPEN in this manual.

FOPEN

Opens a file, defines its physical characteristics, and sets up access to the file.

Syntax

```
 I16                CA            U16V     U16V
filenum:=FOPEN(formaldesignator,foptions,aoptions,
                 I16V     CA     CA       16V
                recsize,device,formmsg,userlabels,
                 I16V        I16V       I32V
                blockfactor,numbuffers,filesize,
                 I16V        I16V       I16V
               numextents,initialloc,filecode);
```

Use

The FOPEN intrinsic sets up access to a file and defines its characteristics and the manner in which it may be accessed. In the FOPEN intrinsic call, you can specify a formal file designator to be associated with the file. This allows you to control the file through file equations when the program is run.

When the FOPEN intrinsic is executed, it returns a file number to the user's process as its functional return value. Because this file number is unique among all other file numbers currently assigned by the system to this process, it is this file number, rather than the formal file designator, that is used by subsequent intrinsic calls to reference the file.

Functional Return

filenum 16-bit signed integer (assigned functional return)

Returns the file number that will be used to identify the opened file in subsequent intrinsic calls.

Parameters

formaldesignator character array (optional)

Passes a string to be associated with the file as its formal file designator. The string can contain alphanumeric characters, slashes or periods, and terminates with any nonalphanumeric character except

a slash (/), a colon (:), or a period (.). If the string is the name of a system defined file it will begin with a dollar sign ($). If the string is the name of a user predefined file, it can begin with an asterisk (*). You can specify the remote location of a device as *filename:envid*.

If you use one of the system defined files, the file can be used for either input, ($STDIN, $STDINX) or output, ($STDLIST), but not for both.

Though not required, the use of a formal file designator is recommended in programs written for asynchronous devices because of the flexibility it provides. If no *formaldesignator* is specified, a temporary nameless file is assigned that you can read from or write to, but not save. Note that all character arrays must start on a halfword boundary.

foptions 16-bit unsigned integer by value (optional)

You can specify various file characteristics by setting corresponding bit groupings in *foptions*. The correspondence is from right to left, beginning with bit 15. These characteristics are as follows, proceeding from the rightmost bit groups to the leftmost bit groups in the word. Not all of the characteristics specified in ***foptions*** have meaning for asynchronous devices, as noted in the descriptions of the individual bit settings.

Specifying *foptions* as zero is equivalent to not specifying *foptions*.

Bits (14:2)—Domain *foptions*.

Used to specify the file domain. If you anticipate redirection of the file to some other device, you should select a domain of 11 (Old Permanent or Temporary).

=00	The file is a NEW file.
=01	The file is a PERMANENT file.
=10	The file is a TEMPORARY file.
=11	The file is an OLD file.

Bit (13:1)—ASCII/binary *foptions*.

Ignored for asynchronous devices. All files not on disk are treated as ASCII files. Use FCONTROL(27)to enable transfer of binary data. Set to 1.

=0	Binary.
=1	ASCII.

Bit (10:3)—Default file designator *foptions*.

You should select $STDIN (100), $STDINX (101) or $STDLIST (001) for terminals, $STDLIST (001)for printers.

=000	The actual and formal file designators are the same.
=001	The actual file designator is $STDLIST.
=010	The actual file designator is $NEWPASS.
=011	The actual file designator is $OLDPASS.
=100	The actual file designator is $STDIN.
=101	The actual file designator is $STDINX.
=110	The actual file designator is $NULL.

Bits (8:2)—Record format *foption*.

The format in which the records in the file are recorded. This field is ignored when a file is opened on a terminal, since records of files on terminals are always of undefined length. However, if you anticipate the need to redirect the file to tape or disk, you may want to set these bits to 00, which specifies “fixed-length”.

=00	Fixed-length records.
=01	Variable-length records.
=10	Undefined-length records.
=11	Spoolfile.

Bit (7:1)—Carriage control *foption*.

Specifies whether or not you will supply a carriage control directive in the calling sequence of each FWRITE call that writes records to the file.

=0	No carriage control character expected (NOCCTL).
=1	Carriage control character expected (CCTL).

A carriage control character passed through the **controlcode** parameter of FWRITE is recognized as such and acted upon only for files that have carriage control specified in FOPEN.

A carriage control character sent to a file on which the control cannot be executed directly (for example, line spacing characters sent to a file spooled to disk), causes

the control character to be embedded as the first byte of the record. Thus, the first byte of each record in such a spooled file contains control information. Carriage control characters sent to other types of files are sent to the driver and acted upon directly.

If CCTL is not specified, only single spacing will be available to this file.

Bit (6:1)—MPE/iX tape label *foption*.

Meaningless for asynchronous devices, but may be set if file redirection is anticipated.

=0	Nonlabeled tape.
=1	Labeled tape.

Bits (5:1)—Disallow **:FILE** equation *foption*.

If this bit is set to 1, any **:FILE** equation intended to override the specifications set through the FOPEN call will be ignored. Note that a leading * in a formal file designator effectively overrides the disallow file equation *foption*.

=0	Allow **:FILE** command.
=1	Disallow **:FILE** command.

Bits (2:3)—File type *foption*.

Set to 000 (standard file) for asynchronous devicefiles.

=000	Standard file (STD).
=001	Keyed Sequential Access Method file (KSAM).
=010	Relative I/O file (RIO).
=100	Circular file (CIR).
=110	Message file (MSG).

Bits (0:2)—Reserved for MPE/iX. These should be set to 0.

aoptions 16-bit unsigned integer by reference (optional)

You can specify up to seven different file access options by setting corresponding bit groupings in *aoptions*. The correspondence is from right to left, beginning with bit 15. These access options are described below, proceeding from the rightmost bit groups to the leftmost bit groups in the word.

If *aoptions* are not specified, the value of all bits is assumed to be 0.

Bits (12:4)—Access type *aoptions*.

=0000	Read access only.
=0001	Write access only.
=0010	Write access only, but previous data in file not deleted.
=0011	Append access only.
=0100	Input/output access.
=0101	Update access.
=0110	Execute access.

The type of access allowed users of the file. Table 9-2 summarizes how these settings apply to terminals and printers.

Bit (11:1)—Multirecord *aoption*.

Specifies whether or not a block of data (size specified in the **tcount** parameter of the read or write request) that exceeds the file's physical record size may be transmitted as several records. This option is only available for terminals. Data written to printers is not broken up on record boundaries. Instead it is written as a single long record.

=0	Select non-multirecord mode (NOMULTI).
=1	Select multirecord mode (MULTI).

Bit (10:1)—Dynamic locking *aoption*.

Meaningless for asynchronous devices. If specified for files not residing on disk, this option will be ignored.

Bit (8:2)—Exclusive access ***aoption***.

Specifies whether you have continuous exclusive access to this file. This option does not apply to printers. Exclusive access is available during the first FOPEN issued against a terminal. If a session is already running on a terminal, it is not possible to issue another FOPEN with "exclusive" specified. (See the MPE/iX Intrinsics Reference Manual for more information on what each of the bit settings implies.)

=00	Default.
=01	Exclusive access.

=10	Semi-exclusive access.
=11	Shared access.

Bit (7:1) —Inhibit buffering *aoption*.

Not meaningful for asynchronous devices. Buffering is inhibited by default. (Inhibited buffering means that your process is assigned no system buffers.) Terminals always are treated with buffering inhibited.

Bit (5:2)—Multiaccess mode *aoption*.

Not meaningful for asynchronous devices.

Bit (4:1)—NOWAIT I/O *aoption*.

Determines whether or not NOWAIT I/O is specified for the file. NOWAIT I/O allows the accessor to initiate an I/O request and to have control returned before the completion of the request. Terminal operation with NOWAIT I/O has the standard NOWAIT ramifications; that is, Privileged Mode is required, no buffering is allowed, and IOWAIT or IODONTWAIT must be called after each I/O request.

=0	NOWAIT I/O not in effect.
=1	NOWAIT I/O in effect.

Bits (3:1)—File copy access *aoption*.

Specifies whether or not a file may be treated as a standard sequential file rather than as a file of its own type.

=0	File must be accessed as its own type.
=1	File may be accessed as standard sequential file.

Bits (0:3)—Reserved for MPE/iX. You should set each of these bits to 0.

recsize 16-bit signed integer by value (optional)

Specifies the size of the logical records in the file. A positive number is used to represent halfwords, a negative number to represent bytes. This value indicates the maximum logical record length allowed if the records in the file are of variable length.

For terminal and printer files, no rounding up occurs if you specify a record size consisting of an odd number of bytes. The record size may be different from that

configured for the port. The default value of **recsize** is the configured record size (normally 40 halfwords for terminals, 66 halfwords for printers).

device `character array (optional)`

A byte array in which you can specify a particular device class or logical device number. On a terminal, you must use this parameter if you intend to read and write through a single file. (That is, if you intend to FOPEN the device once, with read and write access, you must use this parameter.) Note that all character arrays must start on a halfword boundary.

This parameter optionally specifies environment files for certain Hewlett-Packard laser printers (**;ENV=printenv**).

formmsg `character array (optional)`

Passes a forms message that you can use for such purposes as telling the System Operator what type of paper to use in the line printer. This message will be displayed on the console when the file is ready to print, and processing will be delayed until the System Operator verifies that the correct forms are in the printer and responds accordingly. The message can also be used for terminal files.

The message itself is a string of ASCII characters terminated by a period. The maximum number of characters allowed in the array is 49, including the terminating period. Arrays with more than 49 characters are truncated. Note that all character arrays must start on a halfword boundary.

This array also has significance when used for magnetic tape files. See the MPE/iX Intrinsics Reference Manual for more information on this usage.

userlabels `16-bit signed integer by value (optional)`

Not meaningful for asynchronous devices. Applies to disk files only.

blockfactor `16-bit signed integer by value (optional)`

Not meaningful for asynchronous devices. Blocking factor is always 1.

numbuffers `16-bit signed integer by value (optional)`

The number of buffers, number of copies, and output priority to be applied to the file, indicated by three bit groupings.

Bits (11:5)—Number of buffers.

Not meaningful for asynchronous devices. Data transfers to and from terminals and printers are always unbuffered.

Bits (4:7)—Number of copies.

For spooled output devices only, specifies the number of copies of the entire file to be produced by the spooling facility. You can specify number of copies for a file already opened (for example, $STDLIST), in which case the highest value supplied before the last FCLOSE takes effect. The copies do not appear contiguously if the System Operator intervenes or if a file of higher output priority becomes READY before the last copy is complete. These bits are ignored for nonspooled output devices. The default value is 1.

Bits (0:4)—Output priority.

For spooled output devices only, specifies the output priority to be attached to the file. This priority is used to determine the order in which files are output when several are waiting for the same device. Accepted values are between 1 (lowest priority) and 13 (highest priority) inclusive. If this value is less than the current OUTFENCE set by the System Operator, file output is deferred until the operator raises the output priority of the file or lowers the OUTFENCE. You can specify priority for a file already opened (for example, $STDLIST), in which case the highest value supplied before the last FCLOSE takes effect. These bits are ignored for nonspooled output devices. The default priority is 8.

filesize `32-bit signed integer by value (optional)`

Not meaningful for asynchronous devices.

numextents `16-bit signed integer by value (optional)`

Not meaningful for asynchronous devices.

initialloc `16-bit signed integer by value (optional)`

Not meaningful for asynchronous devices.

filecode `16-bit signed integer by value (optional)`

Not meaningful for asynchronous devices.

Condition Codes

CCE Request granted. The file is open.

CCG	Not returned by this intrinsic.
CCL	Request denied. This can occur because another process already has EXCLUSIVE or SEMI-EXCLUSIVE access for this file, the privilege level of this file is not “user” (3), or an initial allocation of disk space cannot be made due to lack of disk space. If the file is not opened successfully, the file number value returned by FOPEN is 0. Call the FCHECK intrinsic for more details.

Additional Discussion

See the *MPE/iX Intrinsics Reference Manual* and *Accessing Files Programmer's Guide*.

FREAD

Transfers a record of data from an input device to the user's data area.

Syntax

```
    I16                 16V     UDS    I16V
transfercount:=FREAD(filenum,buffer,length);
```

Use

Use the FREAD intrinsic to read a logical record from a file to the user's data area. For asynchronous devices, this implies the transfer of data (one record at a time) from an input device, usually a terminal, to the data area. Condition codes are used to indicate completion status and should be checked after each read, whether you are using NOWAIT I/O or normal I/O processing. Read timeouts or timers must be set prior to an FREAD by calls to the appropriate FCONTROL functions.

When the logical end-of-data is encountered during a read, CCG is returned to your process. The end-of-data is indicated by a hardware end-of-file; on $STDIN, by any record beginning with a colon (:); or on $STDINX by **:EOD**.

Functional Return

transfercount 16-bit signed integer (assigned functional return)

Assumes a value corresponding to the length of the information transferred. If the value specified in the **length** parameter of the call was a positive number, the value returned in *transfercount* is positive, and represents a halfword count. If the value specified in the **length** parameter was negative, the value returned in *transfercount* represents a byte count.

FREAD always returns a zero if you specified NOWAIT I/O in FOPEN or HPFOPEN. In this case, the actual record length is returned in the **length** parameter of the IOWAIT or IODONTWAIT intrinsic.

Parameters

filenum 16-bit signed integer by value (required)

The file number of the file to be read.

buffer user-defined structure (required)

Returns the record that was read. This structure should be large enough to hold all of the information to be transferred.

length `16-bit signed integer by value (required)`

The number of halfwords or bytes to be transferred. If this value is positive, it signifies the length in halfwords; if negative, in bytes. A zero indicates that no transfer is to take place.

NOTE

Whether you specify the value of length as halfwords or bytes, the actual physical transfer is done on a byte-by-byte basis for an asynchronous device. If the number of bytes input exceeds the value specified, the read will terminate on byte count.

Condition Codes

CCE	The information was read.
CCG	The logical end-of-data was encountered during reading.
CCL	The information was not read because an error occurred: a terminal read was terminated by a special character or timeout interval, as specified in the FCONTROL intrinsic.

Additional Discussion

See *Accessing Files Programmer's Guide*. See also discussion of IOWAIT and IODONTWAIT in this manual.

FSETMODE

Suppresses the output of linefeed.

```
          I16V    U16V
FSETMODE(filenum,modeflags);
```

Use

For asynchronous devices, the only significant use of the FSETMODE intrinsic is to determine whether or not a linefeed is sent to the terminal at the end of each input line. For certain applications, you may need to suppress the linefeed that is sent automatically during normal operating conditions. In such cases, a call to FSETMODE with the **modeflags** parameter set to 4 should be used. To restore transmission of linefeed, call FSETMODE with **modeflags** set to 0. FCLOSE also resets this option.

Parameters

filenum `16-bit signed integer by value (required)`

The file number of the file whose access modes are to be set.

modeflags `16-bit unsigned integer by value (required)`

This parameter uses bit groupings to indicate desired settings of various options. The only option that applies to asynchronous devices is specified by the value of bit (13:1), as follows:

Bit (13:1) —Controls issuing of a linefeed to a terminal after each terminal read.

`=1`	Inhibit normal terminal control by terminal driver. The result is that the driver does not automatically send out a linefeed at the end of each terminal input line. (**modeflags = 4**).
`=0`	Restore normal terminal control by terminal driver. If the terminal operator ends a line of input by pressing **[Return]**, the Echo Facility echoes the carriage return back to the terminal and the driver sends out a linefeed to the terminal. If input reaches the read byte count limit (as set in the **length** parameter of FREAD or in the **expectedlength** parameter

of READ or READX), the driver sends out a carriage return and a linefeed. (**modeflags = 0**).

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request denied. An error occurred.

Additional Discussion

See also discussions of FCONTROL(12,13) and FCONTROL(41) in this manual.

FWRITE

Transfers a record of data to a file on any device.

Syntax

```
        I16V    UDS    I16V      U16V
FWRITE(filenum,buffer,length,controlcode);
```

Use

The FWRITE intrinsic transfers a logical record of data to a file on any device. When programming for asynchronous devices this means writing data to a terminal or printer. Completion status is indicated by condition codes, which your program should always check.

This intrinsic is completed “logically”; that is, once data is passed to an output buffer the call is considered to have completed and control is returned to the calling program. If the operation of the output device has been suspended through flow control, the actual writing of the data may not have taken place. There is no programmatic way to determine that printing has taken place.

Parameters

filenum `16-bit signed integer by value (required)`

The file number of the file to be written to.

buffer `user-defined structure (required)`

Passes the record to be written.

length `16-bit signed integer by value (required)`

The number of halfwords or bytes to be written to the record. If this value is positive, it signifies halfwords; if negative, bytes. If the number is zero, no data transfer will occur.

NOTE

Whether or not “bytes” is specified, the physical data transfer will be on a byte-by-byte basis for an asynchronous device. If halfwords are specified, an “even” number of bytes are output.

controlcode `16-bit unsigned integer by value (required)`

This parameter indicates the carriage control action to be used when the file is transferred to a line printer or terminal (including a spooled file whose ultimate destination is a line printer or terminal). This

parameter is effective only for files opened with carriage control (CCTL) specified in the FOPEN or HPFOPEN call.

The **controlcode** parameter may contain 0, 1, or one of the octal codes from Table 9-3. These options are used to indicate carriage control action as follows:

0	Print the full record transferred, using single spacing. This results in a maximum of 132 characters per printed line.
1	Use the first character of the data written to satisfy space control and suppress this character on the printed output. The results are a maximum of 132 characters of data per printed line. The values permissible for use as the first character to specify control are the same as the octal values described below, and listed in Table 9-3.

When one of the octal codes listed in Table 9-3 is used as the value of **controlcode** the carriage control action associated with the specific code is applied to the data being transferred. (You must have specified CCTL through the FOPEN or HPFOPEN call, or through an overriding **:FILE** command.) The full record transferred is then printed, with a maximum of 132 characters per printed line.

If the file being written is not a spooled file, (that is, it is not written to disk before being output to the device), all of the octal codes listed in Table 9-3 can also be passed through FCONTROL(1).

A carriage control directive passed through FCONTROL(1) takes effect whether or not CCTL was specified for the file.

For spooled files, however, carriage control must be set using FWRITE. Additionally, the prespace/postspace control normally set through FCONTROL must be set through FWRITE for spooled files. You can use control codes %100 through %103 and %400 through %403 for this purpose.

If the **controlcode** parameter is not 0 or 1, and **length** is 0, only the space control is executed and no data is transferred.

The default carriage control is postspacing with automatic page eject. This applies to all Hewlett-Packard supported subsystems, with the exception of FORTRAN77/iX and COBOLII/iX, which have prespacing with automatic page eject.

Table 9-3 Selected Carriage Control Directives

Octal Code	Carriage
%2–%52	Single space (with or without automatic page eject).
%53	No space, (next printing at column 1).
%54	Single space (with or without automatic page eject).
%60	Double space (with or without automatic page eject).
%61	Conditional page eject (form feed) is performed by the software. If the printer is not already at the top of the form, a page eject is performed. Ignored if: • Postspace mode: The current request has a transfer count of 0 and the current request was FOPEN (or HPFOPEN), FCLOSE, or FWRITE specifying a carriage control directive of %61. • Prespace mode: The previous request has a transfer count of 0, and the current request and previous request are any combination of FOPEN (or HPFOPEN), FCLOSE, or FWRITE specifying a carriage control directive of %61.
%62–%77 %104–%177	Single space (with or without automatic page eject).
%2*nn*	Space nn lines (no automatic page eject); *nn* is any octal number from 0 through 77.
%300	Skip to top of form (page eject).
%301	Skip to bottom of form.
%302	Single spacing with automatic page eject.
%303	Skip to next odd line with automatic page eject.
%304	Skip to next third line with automatic page eject.
%305	Skip to next 1/2 page.
%306	Skip to next 1/4 page.
%307	Skip to next 1/6 page.
%310	Skip to bottom of form.
%311	Skip to one line before bottom of form.

Octal Code	Carriage
%312	Skip to one line before top of form.
%313	Skip to top of form (same as %300).
%314	Skip to next seventh line with automatic page eject.
%315	Skip to next sixth line with automatic page eject.
%316	Skip to next fifth line with automatic page eject.
%317	Skip to next fourth line with automatic page eject.
%320	No space, no return (next printing physically follows this).
%321–%377	Single space (with or without automatic page eject).
%400 or %100	Sets postspace movement option; this first print, then spaces. If previous option was prespace movement, the driver outputs a one line advance (equivalent to an octal code of %201 without automatic page eject) to clear the buffer.
%401 or %101	Sets prespace movement option; this first spaces, then prints.
%402 or %102	Sets single-space option, with automatic page eject (60 lines per page).
%403 or %103	Sets single-space option, without automatic page eject (66 lines per page).

NOTE

If you select octal codes %55 and %60 with automatic page eject in effect (by default or following an octal code of %102 or %402), the resulting skip across a page boundary is to the top of the page. If automatic page eject is not in effect, a true double or triple space results, but the perforation between pages is not automatically skipped.

Condition Codes

CCE	Request granted.
CCG	Physical EOF. Should not occur for terminals or printers.
CCL	Request denied due to an error condition. Use FCHECK to obtain details.

Additional Discussion

See the discussion of FCONTROL(1) in this manual. See also the *MPE/iX Intrinsics Reference Manual*.

HPFOPEN

Opens a file, defines its physical characteristics, and sets up access to the file.

Syntax

```
          I32   I32     I32V   *
HPFOPEN(filenum,status[,itemnum,item
                      [,itemnum,item
                      [,itemnum,item
                            .
                            .
                            .
                      [,itemnum,item]...]]]);
```

Use

The HPFOPEN intrinsic sets up access to a file and defines its characteristics and the manner in which it may be accessed. HPFOPEN provides a superset of the options provided by the FOPEN intrinsic. That is, all of the file options available through an FOPEN call are provided, along with some additional options and enhanced features. When these additions or enhancements apply to asynchronous device control, they will be noted in the parameter descriptions that follow. For a complete summary of all the additional options and features provided by HPFOPEN you should see the *MPE/iX Intrinsics Reference Manual*.

NOTE Because HPFOPEN is designed to be more flexible and extensible than the FOPEN intrinsic, HPFOPEN is the recommended intrinsic for file access.

The HPFOPEN intrinsic uses *itemnum/item* pairs to specify file options. The *itemnum* indicates the option being set, and the item specifies the desired setting. The *itemnum/item* optional parameters must always appear in pairs, except for Item # = 0, which can be used to end the option list. You can specify up to 41 option pairs in the same call, following the rules of precedence listed below:

- An *itemnum* takes precedence over any previously specified duplicate *itemnum*. Any duplicated *itemnum* is flagged as a warning.
- Some pairs are mutually exclusive. When applicable, these are noted in the *itemnum/item* summary that follows.

- Each optional parameter specified in the *file equation string option* (Item# = 52) is treated as if it were the equivalent *itemnum/item* pair when determining precedence.

Parameters

filenum 32-bit signed integer by reference (required)

Returns a file number used to identify the opened file in subsequent intrinsic calls.

status 32-bit signed integer by reference (optional)

Returns the status of the HPFOPEN call. If no errors or warnings are encountered, *status* returns 32 bits of zero. If errors or warnings are encountered, *status* is interpreted as two 16-bit fields. The **status** parameter is used in place of error checking routines that call FCHECK. Such routines are not useful with HPFOPEN.

Bits (0:16) comprise *status.info*. A negative value indicates an error condition, and a positive value indicates a warning condition. See the *MPE/iX Intrinsics Reference Manual* for a list of possible values.

Bits (16:16) comprise *status.subsys*. The value represented by these bits defines the subsystem that set the status information. The File System identification number is 143.

CAUTION Since HPFOPEN can return information on the success of its execution in the **status** parameter, it is good programming practice to specify this parameter and check its value after the intrinsic call. If an error or warning condition is encountered and you did not specify the **status** parameter, HPFOPEN causes the calling process to abort.

filenum 32-bit signed integer by value (optional)

The item number of an HPFOPEN option, as defined in the *itemnum/item* summary that follows.

item type varies by reference (optional)

Passes and/or returns the HPFOPEN option indicated by the corresponding itemnum parameter, as described in the *itemnum/item* summary that follows.

NOTE All **item** parameters of type character array (CA), except for the `fill character option` (Item# = 45), must be delimited by a single character which appears only as the first and last element of the specified character array. For example:

%filename% (% is the delimiter, filename is the designator)

fabcxyzf (f is the delimiter, abcxyz is the designator)

Item Number, Item Summary

The following discussion lists the optional *itemnum/item* pairs. A short descriptive name is provided for each pair, along with a mnemonic indicating the parameter type of each *item*. All **item** parameters are passed by reference.

Item# = 0 Indicates end of the option list. This is the only *itemnum* that does not have a corresponding *item*. The absence of an *itemnum* after the last *itemnum/item* pair is equivalent to specifying this option.

Item# = 2 *formaldesignator option (CA)*

Passes a string to be associated with the file as its **formal file designator**. The string can contain alphanumeric characters, slashes, or periods and terminates with any nonalphanumeric character except a slash (/), a colon (:), or a period (.). If the string is the name of a system defined file, it will begin with a dollar sign ($). If the string is the name a user predefined file, it can begin with an asterisk (*). You can specify the remote location of a device as `filename:envid`.

If you use one of the system defined files the file can be used for either input, (`$STDIN`, `STDINX`) or output, (`$STDLIST`), but not for both.

Though not required, the use of a formal file designator is recommended in programs written for asynchronous devices because of the flexibility it provides. If no *formaldesignator* is specified, a temporary nameless file is assigned that you can read from or write to, but not save.

Note that all character arrays must start on a halfword boundary.

Item# = 3 *domain option (I32)*

Passes a value indicating the file domain. HPFOPEN assumes that all devicefiles are PERMANENT files. However, to maintain compatibility with MPE V/E, devicefiles can be opened with the *domain option* set to NEW, but a warning message is returned in the **status** parameter.

0	The file is a NEW TEMPORARY file.
1	The file is a PERMANENT file.
2	The file is a TEMPORARY file.
3	The file is an OLD PERMANENT or TEMPORARY file.
4	The file is a NEW PERMANENT file.

Item# = 5 *designator option (I32)*

Passes a value indicating a special file opening. Note that special files can also be specified through the *formaldesignator option* (Item# = 2). You should select $STDIN (4), $STDINX (5) or $STDLIST (1) for terminals, $STDLIST (1) for printers.

0	Allows all other options to specify the file.
1	The actual file designator is $STDLIST.
2	The actual file designator is $NEWPASS.
3	The actual file designator is $OLDPASS.
4	The actual file designator is $STDIN.
5	The actual file designator is $STDINX.
6	The actual file designator is $NULL.

Item# = 6 *record format option (I32)*

Passes a value indicating the format in which the records in the file are recorded. This field is ignored when a file is opened on a terminal, since records of files on terminals are always of undefined length. However, if you anticipate the need to redirect the file to tape or disk, you may want to set this value to 0, which specifies “fixed-length”.

0	Fixed-length records.
1	Variable-length records.
2	Undefined-length records (no implied record structure).

Item# = 7 *Carriage Control Option (I32)*

Passes a value indicating whether or not you will supply a carriage control directive in the calling sequence of each FWRITE call that writes records to the file.

0	No carriage control directive is expected (NOCCTL).
1	Carriage control directives are expected (CCTL).

A carriage control character passed through the **controlcode** parameter of FWRITE is recognized as such and acted upon only for files that have carriage control specified in FOPEN.

A carriage control character sent to a file on which the control cannot be executed directly (for example, line spacing characters sent to a file spooled to disk) causes the control character to be embedded as the first byte of the record. Thus, the first byte of each record in such a spooled file contains control information. Carriage control characters sent to other types of files are sent to the driver and acted upon directly.

Item# = 8 *labeled tape label option (CA)*

Not meaningful for asynchronous devices.

Item# = 9 *Disallow file equation option (I32)*

Passes a value indicating whether or not you wish to allow MPE/iX file equations. Note that a leading * in a formal file designator effectively overrides the disallow option.

0	Allow **:FILE** equations to override programmatic or system defined file specifications.
1	Disallow **:FILE** equations from overriding programmatic or system defined file specifications.

Item# = 10 *file type option (I32)*

Set to 0 (standard file) for asynchronous devicefiles.

0	Standard file (STD).
1	Keyed Sequential Access Method file (KSAM).
2	Relative I/O file (RIO).

4	Circular file (CIR).
6	Message file (MSG).

Item# = 11 access type option (I32)

Passes a value that specifies access restrictions for the file. See Table 9-2 for a summary of how the access type settings apply to terminals and printers.

0	Read access only.
1	Write access only.
2	Write access only, but previous data in file not deleted
3	Append access only.
4	Input/output access.
5	Update access.
6	Execute access.
7	Execute-Read access. (Not included in Table 9-2. Returns File System Error 40 if applied to an asynchronous devicefile.)
8	Reserved for MPE/iX. No access; opens the file without any access checking. A process must be executing in system code to use this access type.

Item# = 12 dynamic locking option I32)

Not meaningful for asynchronous devices. If specified it will be ignored.

Item# = 13 exclusive option (I32)

Passes a value indicating whether you have continuous exclusive access to this file. This option does not apply to printers. Exclusive access is available during the first open issued against a terminal. If a session is already running on a terminal, it is not possible to issue another HPFOPEN (or FOPEN) with “exclusive” specified. (See the *MPE/iX Intrinsics Reference Manual* for more information on what each of the settings implies.)

0	Default.
1	Exclusive access.
2	Semi-exclusive access.

3	Shared access.

Item# = 14 *multiaccess option (I32)*

Not meaningful for asynchronous devices.

Item# = 15 *multirecord option (I32)*

Passes a value indicating whether or not a block of data that exceeds the file's physical record size may be transmitted as several records.

This option is only available for terminals. Data written to printers is not broken up on record boundaries. Instead it is written as a single long record.

0	Non-multirecord mode (NOMULTI).
1	Multirecord mode (MULTI).

Item# = 16 *nowait i/o option (I32)*

Passes a value that determines whether or not NOWAIT I/O is specified for the file. NOWAIT I/O allows the accessor to initiate an I/O request and to have control returned before the completion of the request. Terminal operation with NOWAIT I/O has the standard NOWAIT ramifications; that is, Privileged Mode is required, no buffering is allowed, and IOWAIT or IODONTWAIT must be called after each I/O request.

0	NOWAIT I/O not in effect.
1	NOWAIT I/O in effect.

Item# = 17 *copy mode option (I32)*

Passes a value that specifies whether or not a file may be treated as a standard sequential file rather than as a file of its own type.

0	File must be accessed as its own type.
1	File may be accessed as standard sequential file.

Item# = 18 *short mapped option (@32)*

Not meaningful for asynchronous devices.

Item# = 19 *record size option (I32)*

Passes the size, in bytes, of the logical records in the file. This value indicates the maximum logical record length allowed if the records in the file are of variable length.

For terminal and printer files, no rounding up occurs if you specify a record size consisting of an odd number of bytes. The record size may be different from that configured for the port. The default value is the configured record size (normally 40 words for terminals, 66 words for printers).

Item# = 20 *device name option (CA)*

Passes the logical device number, in ASCII form, of a specific device. The device specified should be ready prior to the HPFOPEN call or an error will result. You can obtain the logical device number through the FFILEINFO intrinsic. Note that all character arrays must start on a halfword boundary.

Only one of the following options can be in effect when a file is opened: *device name option* (Item# = 20), *volume class option* (Item# = 22), *volume name option* (Item# = 23), *device class option* (Item# = 42). The last itemnum specified takes precedence over all previously specified *itemnums* from this group. Any duplication is flagged as a warning.

Item# = 21 *long mapped option (@64)*

Not meaningful for asynchronous devices.

Item# = 22 *volume class option (CA)*

Not meaningful for asynchronous devices.

Item# = 23 *volume name option (CA)*

Not meaningful for asynchronous devices.

Item# = 24 *density option (I32)*

Not meaningful for asynchronous devices.

Item# = 25 *printer environment option (CA)*

Passes the name of a file that contains a printer environment. This option is valid only for certain printer devices. Note that all character arrays must start on a halfword boundary.

Item# = 26 *remote environment option (CA)*

Passes the node name of the remote computer where the file is located. This option is used when you are referencing a file located on a remote computer. Note that all character arrays must start on a halfword boundary.

Item# = 27 *output priority option (I32)*

Passes the output priority to be attached to the file for spooled output. This priority is used to determine the order in which files are output when several are waiting for the same device. Accepted values are between 1 (lowest priority) and 13 (highest priority) inclusive. If this value is less than the current OUTFENCE set by the System Operator, file output is deferred until the operator raises the output priority of the file or lowers the OUTFENCE. You can specify priority for a file already opened (for example, $STDLIST), in which case the highest value supplied before the last FCLOSE takes effect. This parameter is ignored for nonspooled devices. The default priority is 8.

Item# = 28 spooled message option (CA)

Passes a spooler message that is associated with a spool file. For example, you can pass a message that you can use for such purposes as telling the System Operator what type of paper to use in the line printer. This message must be displayed to the System Operator and verified before the file can be printed on a line printer. The maximum number of characters allowed is 48. An array of more than 48 elements is truncated by MPE/iX. Note that all character arrays must start on a halfword boundary.

Item# = 29 privileged access option (I32)

Passes a value that temporarily restricts access to the file number returned from HPFOPEN to a calling process whose execution level is equal to or less than the value specified in this option. This restriction lasts until you close the file associated with the restricted file number. You cannot specify a value less than the execution level of the calling process.

0	Privilege level zero (most privileged level).
1	Privilege level one.
2	Privilege level two
3	Privilege level three (least privileged level).

Item# = 30 labeled tape type option (I32)

Not meaningful for asynchronous devices.

Item# = 31 labeled tape expiration option (CA)

Not meaningful for asynchronous devices.

Item# = 32 *labeled tape sequence option (CA)*

Not meaningful for asynchronous devices.

Item# = 33 *user labels option (I32)*

Not meaningful for asynchronous devices.

Item# = 34 *spooler copies option (I32)*

Passes a non-negative value indicating the number of copies of the entire file to be produced by the spooling facility. This option is applicable to spooled devices only. You can specify this option for a file already opened (for example, $STDLIST) in which case the highest value supplied before the last FCLOSE takes effect. The copies do not appear contiguously if the System Operator intervenes or if a file of higher output priority becomes READY before the last copy is complete. This option is ignored for nonspooled output devices. The default is to print 1 copy.

Item# = 35 *filesize option (I32)*

Not meaningful for asynchronous devices.

Item# = 36 *initial allocation option (I32)*

Not meaningful for asynchronous devices.

Item# = 37 *filecode option (I32)*

Not meaningful for asynchronous devices.

Item# = 38 *file privilege option (I32)*

Passes a value that determines a permanent privilege level to be associated with a newly created file. This option permanently restricts file access to a process whose execution level is less than or equal to the specified value. You cannot specify a value less than the execution level of the calling process. This option is applicable only at file creation.

0	Privilege level zero (most privileged level).
1	Privilege level one.
2	Privilege level two.
3	Privilege level three (least privileged level).

Item# = 39 *will access option (I32)*

Not meaningful for asynchronous devices.

Item# = 40 *block factor option (I32)*

Not meaningful for asynchronous devices.

Item# = 42 *device class option (CA)*

Passes a device class name which is used by the file system to select a nonsharable device from a configured list of available devices within that device class. The name can have a length of up to eight alphanumeric characters, beginning with a letter. If you specify a device class, the file is allocated to any available device in that class. Note that all character arrays must start on a halfword boundary.

Only one of the following options can be in effect when a file is opened: *device name option* (Item# = 20), *volume class option* (Item# = 22), *volume name option* (Item# = 23), *device class option* (Item# = 42). The last *itemnum* specified takes precedence over all previously specified *itemnums* from this group. Any duplication is flagged as a warning.

Item# = 43 *UFID option (record)*

Not meaningful for asynchronous devices.

Item# = 44 *numbuffers option (I32)*

Not meaningful for asynchronous devices. Data transfers to and from terminals and printers are always unbuffered.

Item# = 45 *fill character option (CA)*

Passes two ASCII characters that determine what padding character you wish to use at the end of blocks or unused pages. You must not use delimiter characters for this option. The fill character must be a two-byte array. The first character only is used as the padding character. The second character is reserved for future use. This option is applicable only at file creation. The default is the ASCII null character. Note that all character arrays must start on a halfword boundary.

Item# = 46 *inhibit buffering option (I32)*

Not meaningful for asynchronous devices.

Item# = 47 *numextents option (I32)*

Not meaningful for asynchronous devices.

Item# = 48 *reverse VT option (I32)*

Passes a value that indicates whether or not the given device name is to be allocated on a remote machine. You must specify the remote environment in the same open request, using either the *formaldesignator* option or the *remote environment option*. Reverse VT behaves nearly the same as a terminal opened through remote file access, except that no session is required on the remote machine.

0	No reverse VT.
1	Reverse VT.

Item# = 49 Reserved for MPE/iX.

Item# = 50 *final disposition option (I32)*

Not meaningful for asynchronous devices.

Item# = 51 *Pascal/iX string option (Pascal/iX STRING type)*

Passes a formal file designator, following MPE/iX file naming conventions, but using the Pascal/iX STRING type format. This option is identical to *formaldesignator option* (Item# = 2) except for the type of the item. No delimiters are needed.

Item# = 52 *file equation string option (CA)*

Passes a character string that matches the MPE/iX file equation specification syntax exactly. This enables you to specify the options available in the **:FILE** command. Note that all character arrays must start on a halfword boundary.

Item# = 53 *ASCII/binary option (I32)*

Ignored for asynchronous devices. All files not on disk are treated as ASCII files. Use FCONTROL(27) to enable the transfer of binary data. Set to 1.

0	Binary file.
1	ASCII file.

Item# = 54 *KSAM parm option (REC)*

Not meaningful for asynchronous devices.

Item# = 55 Reserved for MPE/iX.

Item# = 56 *object class option (I32)*

Passes a user object class number, in the range 0..10, that is associated with the file.

Item# = 57 Reserved for MPE/iX.

Item# = 58 Reserved for MPE/iX.

Item# = 59 Reserved for MPE/iX.

Item# = 60 Reserved for MPE/iX.

Item# = 61 Reserved for MPE/iX.

Condition Codes

The HPFOPEN intrinsic does not return meaningful condition code values. Status information is returned in the optional *status* parameter described above.

Additional Discussion

See the discussions of creating files and opening files in Accessing Files. See also the *MPE/iX Intrinsics Reference Manual* and the *MPE/iX Commands Reference Manual*.

IODONTWAIT

Initiates completion operations for an I/O request. Allows program processing to continue before an I/O operation completes.

Syntax

```
 I16              I16V     UDS     16      U16
fnum:=IODONTWAIT(filenum,buffer,length,cstation);
```

Use

The IODONTWAIT intrinsic initiates completion operations for an I/O request and returns control to the calling process even if no I/O has completed.

A call to either IODONTWAIT or its companion intrinsic, IOWAIT, must follow every read or write request posted against a file that was opened with NOWAIT I/O specified in the FOPEN or HPFOPEN call. (The process must have been running in privileged mode for NOWAIT I/O to have been specified.)

When NOWAIT I/O is used, the parameters associated with the IODONTWAIT or IOWAIT call receive the values that normally would have been returned in the read or write request being processed. See the parameter explanations for what these values represent.

If you call IODONTWAIT and no I/O has completed, control is returned to the calling process, CCE is returned, and the result of IODONTWAIT is zero.

If you call IOWAIT and no I/O has completed, the calling process is suspended until some I/O completes.

Functional Return

fnum 16-bit signed integer (assigned functional return)

The file number for which the completion occurred. If no completion occurred, zero is returned.

Parameters

filenum 16-bit signed integer by value (required)

The file number for which there is a pending I/O request. If you specify 0, the IODONTWAIT intrinsic checks for any I/O completion.

buffer user-defined structure (optional)

Passes an input buffer. Since terminals always operate without system buffering it is not necessary for this parameter to follow a read.

length `16-bit signed integer by reference (optional)`

Returns the length of the received or transmitted record. If the original request specified a byte count, the integer represents bytes. If the original request specified halfwords, the integer represents halfwords. This parameter is pertinent only if the original I/O request was a read request. The FREAD intrinsic always returns zero as its functional return if you specify NOWAIT I/O. Instead, the actual record length is returned in the length parameter of IODONTWAIT.

cstation `16-bit unsigned integer by reference (optional)`

Not meaningful for asynchronous devices.

Condition Codes

CCE	Request granted. If the functional return is not zero, then I/O completion occurred with no errors. If the functional return is zero, no I/O has completed.
CCG	An end-of-file condition was encountered.
CCL	Request denied. Normal I/O completion did not occur because there were no I/O requests pending, a parameter error occurred, or an abnormal I/O completion occurred.

Special Considerations

You must be running in privileged mode to specify the NOWAIT I/O option in the FOPEN/HPFOPEN call.

Additional Discussion

See the discussion of initiating completion operations for I/O requests in *Accessing Files Programmer's Guide*. See also discussions of FOPEN, HPFOPEN and IOWAIT in this manual.

IOWAIT

Initiates completion operations for an I/O request. Delays program processing until an I/O operation completes.

Syntax

```
 I16          I16V    UDS    I16     U16
fnum:=IOWAIT(filenum,buffer,length,cstation);
```

Use

The IOWAIT intrinsic initiates completion operations for an I/O request and suspends the calling process until some I/O completes.

A call to either IOWAIT or its companion intrinsic, IODONTWAIT, must follow every read or write request posted against a file that was opened with NOWAIT I/O specified in the FOPEN or HPFOPEN call. (The process must have been running in privileged mode for NOWAIT I/O to have been specified.)

When NOWAIT I/O is used the parameters associated with the IODONTWAIT or IOWAIT call receive the values that normally would have been returned in the read or write request. See the parameter explanations that follow for what these values represent.

If you call IOWAIT and no I/O has completed, the calling process is suspended until some I/O completes.

If you call IODONTWAIT and no I/O has completed, control is returned to the calling process, CCE is returned, and the result of IODONTWAIT is zero.

Functional Return

fnum 16-bit signed integer (assigned functional return)

The file number for which the completion occurred. If no completion occurred, zero is returned.

Parameters

filenum 16-bit signed integer by value (required)

The file number for which there is a pending I/O request. If you specify 0, the IOWAIT intrinsic checks for any I/O completion.

buffer user-defined structure (optional)

Passes an input buffer. Since terminals always operate without system buffering, it is not necessary for this parameter to follow a read.

length `16-bit signed integer by reference (optional)`

Returns the length of the received or transmitted record. If the original request specified a byte count, the integer represents bytes. If the original request specified halfwords, the integer represents halfwords. This parameter is pertinent only if the original I/O request was a read request. The FREAD intrinsic always returns zero as its functional return if you specify NOWAIT I/O. Instead, the actual record length is returned in the length parameter of IOWAIT.

cstation `16-bit unsigned integer by reference (optional)`

Not meaningful for asynchronous devices.

Condition Codes

CCE	Request granted. I/O completion occurred with no errors.
CCG	An end-of-file condition was encountered.
CCL	Request denied. Normal I/O completion did not occur because there were no I/O requests pending, a parameter error occurred, or an abnormal I/O completion occurred.

Special Considerations

You must be running in privileged mode to specify the NOWAIT I/O option in the FOPEN/HPFOPEN call.

Additional Discussion

See the discussion of initiating completion operations for I/O requests in *Accessing Files Programmer's Guide*. See also discussions of FOPEN, HPFOPEN and IODONTWAIT in this manual.

PRINT

Prints a character string on job/session list device.

Syntax

```
        CA      I16V     I16V
PRINT(message,length,controlcode);
```

Use

The PRINT intrinsic enables you to write a string of ASCII characters from your program to the job/session listing device ($STDLIST). This provides a quick method of sending output to the terminal. However, because the PRINT intrinsic does not allow you to take full advantage of the capabilities of the file system, it is recommended that you use FWRITE for permanent programs.

Parameters

message `character array (required)`

Passes the string of ASCII characters to $STDLIST. Note that all character arrays must start on a halfword boundary.

length `16-bit signed integer by value (required)`

The length of the character string to be passed in message. If length is positive, the length is in halfwords. If length is negative, the length is in bytes. If length exceeds the configured record length of the device, successive records are written only on terminals.

controlcode `16-bit signed integer by value (required)`

A carriage control directive. See Table 9-3 for a list of valid codes.

Condition Codes

CCE	Request granted.
CCG	End-of-data was encountered.
CCL	Request denied. An input/output error occurred. Further error analysis through the FCHECK intrinsic is not possible.

Additional Discussion

See the discussion of printing character strings to $STDLIST in *Accessing Files Programmer's Guide*.

PRINTFILEINFO

Prints a file information display on a job/session list device.

Syntax

```
                I16V
PRINTFILEINFO(filenum);
```

Use

The PRINTFILEINFO intrinsic causes MPE/iX to print a file information display on the standard list device. You can use this intrinsic to display error information after a condition code of CCG or CCL is returned. The display, sometimes referred to as the tombstone, appears in one of two formats, depending on whether or not the file whose number is specified in filenum was opened at the time the error occurred. See the *MPE/iX Intrinsics Reference Manual* for examples of the two formats.

Parameters

filenum 16-bit signed integer by value (required)

The file number of the file about which you are requesting information.

Condition Codes

The PRINTFILEINFO intrinsic does not alter the condition code.

Additional Information

See the discussion of print formats and writing a file system error check procedure in *Getting System Information*. See also the *MPE/iX Intrinsics Reference Manual*.

READ

Transfers a record of data from the file $STDIN to the data area.

Syntax

```
     I16              CA         I16V
transfercount:=READ(message,expectedlength);
```

Use

The READ intrinsic reads an ASCII string from $STDIN into an array.

This is similar to calling the FREAD intrinsic against the file $STDIN. READ is intended as a quick method of obtaining input from the terminal for temporary programming. Read timers or timeouts must be set prior to calling READ by first calling FCONTROL using the appropriate **controlcode** values.

When you use the READ intrinsic, the full capabilities of the file system are not available to you. For example, since no file number can be specified in the call it is not possible to use the **:FILE** command to redirect input for programs using READ. (The only way that input can be redirected is to use the $STDIN=*filename* option of the **:RUN** command.) Input always comes from the $STDIN file opened by the session at logon time.

READ differs from READX in how it interprets an end-of-file. READ reads a record and, if there is a colon in the first column, end-of-file is set.

Functional Return

transfercount 16-bit signed integer (assigned functional return)

The length of the ASCII string that was read. If **expectedlength** is positive, the count represents halfwords. If **expectedlength** is negative, the count represents bytes.

Parameters

message character array (required)

Returns the ASCII characters that were read. Note that all character arrays must start on a halfword boundary.

expectedlength 16-bit signed integer by value (required)

The maximum length of the **message** array. If **expectedlength** is positive, it specifies the length in halfwords. If **expectedlength** is negative, it specifies the length in bytes. When the record is read, the first **expectedlength** characters are input. If the number of bytes input exceeds this value, the read terminates on byte count.

Condition Codes

CCE	Request granted.
CCG	A record with a colon in the first column, signaling the end-of-data or a hardware end-of-file, was encountered.
CCL	Request denied. A physical input/output error occurred. Further analysis through the FCHECK intrinsic is not possible.

Additional Discussion

See the discussion of reading ASCII strings from `$STDIN` into an array in *Accessing Files Programmer's Guide*.

READX

Transfers a record of data from the file $STDINX to the data area.

Syntax

```
     I16                CA          I16V
transfercount:=READX(message,expectedlength);
```

Use

The READX intrinsic reads an ASCII string from $STDINX into an array. This is similar to calling the FREAD intrinsic against the file $STDINX. READX is intended as a quick method of obtaining input from the terminal for temporary programming. Read timers or timeouts must be set prior to calling READX by first calling FCONTROL using the appropriate **controlcode** values.

When you use the READ intrinsic, the full capabilities of the file system are not available to you. For example, since no file number can be specified in the call, it is not possible to use the **:FILE** command to redirect input for programs using READX. (The only way that input can be redirected is to use the $STDINX=*filename* option of the **:RUN** command.) Input always comes from the $STDINX file opened by the session at logon time.

READX differs from READ in how it interprets an end-of-file. READX interprets **:EOD** as an end-of-file indication.

Functional Return

transfercount 16-bit signed integer (assigned functional return)

The length of the ASCII string that was read. If **expectedlength** is positive, the count represents halfwords. If **expectedlength** is negative, the count represents bytes.

Parameters

message character array (required)

Returns the ASCII characters that were read. Note that all character arrays must start on a halfword boundary

expectedlength 16-bit signed integer by value (required)

The maximum length of the **message** array. If **expectedlength** is positive, it specifies the length in halfwords. If **expectedlength** is negative, it specifies the length in bytes. When the record is read, the first **expectedlength** characters are input. If the number of bytes input exceeds this value, the read terminates on byte count.

Condition Codes

CCE	Request granted.
CCG	An **:EOD** was encountered.
CCL	Request denied. A physical input/output error occurred. Further analysis through the FCHECK intrinsic is not possible.

Additional Discussion

See the discussion of reading ASCII strings from $STDINX into an array in *Accessing Files Programmer's Guide*.

RESETCONTROL

Allows another subsystem break to occur.

Syntax

```
RESETCONTROL;
```

Use

The RESETCONTROL intrinsic lets the calling process accept another subsystem break signal. (**[CTRL]Y** is the default subsystem break character.) The process must have previously enabled a **[CTRL]Y** trap with the XCONTRAP intrinsic. After your **[CTRL]Y** trap handler has been invoked, you should call RESETCONTROL when you are ready to receive another **[CTRL]Y** signal. RESETCONTROL can be called from within the **[CTRL]Y** trap handler or from any other procedure.

Parameters

None.

Condition Codes

CCE	Request granted.
CCG	Not returned by this intrinsic.
CCL	Request denied because the trap handling procedure was not invoked or because RESETCONTROL was already called.

Additional Discussion

See the discussion of resetting the **[CTRL]Y** trap in *Accessing Files, Trap Handling, and Data Type Conversion Programmer's Guides*. See also discussions of FCONTROL(16,17) and XCONTRAP in this manual. See Chapter 5, "Using Subsystem Break," of this manual for an example illustrating subsystem break processing.

XCONTRAP

Arms or disarms the user written subsystem break trap handling procedure.

Syntax

```
I32V I32
XCONTRAP(plabel,oldplabel);
```

Use

The XCONTRAP intrinsic arms or disarms the user-written subsystem break (**[CTRL]Y**) trap handling procedure.

When a session is initiated, the user written **[CTRL]Y** trap handler is disarmed. If you are running a program in an interactive session, you can arm a special trap that transfers control in the program to a trap handling procedure whenever a subsystem break signal is entered from the session terminal. (This is called the **[CTRL]Y** trap since **[CTRL]Y** is the default subsystem break character.)

The subsystem break signal is normally transmitted by pressing **[CTRL]Y**. In transparent editing mode, you are allowed to define a different character to be used to invoke subsystem break by specifying the new character as part of **param** in a call to FCONTROL(41).

You can also use FDEVICECONTROL to alter the subsystem break character in either standard or transparent mode. See the discussion of the FDEVICECONTROL intrinsic earlier in this chapter for the proper parameter settings to use for this purpose.

If enabled, subsystem break allows the terminal user to stop a “program local” or “subsystem local” command. Before subsystem break can be enabled, however, you must call XCONTRAP, specifying the external label of a user written procedure which contains the steps that will be taken if subsystem break is entered during execution of your program as the value of **plabel**.

If it is desirable for your program to do so, subsystem break can be temporarily disabled through a call to FCONTROL(16). It can then be reenabled through a call to FCONTROL(17).

Only one process in a session can receive a **[CTRL]Y** trap at any one time. The process that called XCONTRAP most recently receives the next **[CTRL]Y** trap. Once a process has received a **[CTRL]Y** trap, it cannot receive another until it calls the RESETCONTROL intrinsic. Only processes running in a session (not in a job) can arm **[CTRL]Y** traps. The

trap handler can be any procedure in the program or in the libraries to which the program is bound. The **[CTRL]Y** trap handler has no parameters.

The following summarizes subsystem break processing and the intrinsics involved:

1. A user written procedure must be provided to define how the program should act upon receiving a subsystem break.
2. A call to XCONTRAP must be included in the program, specifying the external label of the procedure written in Step 1 as the value of **plabel**. This arms the **[CTRL]Y** Trap.
3. A call to FCONTROL(17) must be included in the program to enable the subsystem break function.
4. If the subsystem break character is received during execution of the program, the procedure specified in the call to XCONTRAP is executed.
5. A call to RESETCONTROL must be executed when the program is ready to receive another subsystem break.
6. A new subsystem break character may be defined through FCONTROL(41), if the terminal is placed in transparent mode or through FDEVICECONTROL
7. If desirable at any time, FCONTROL(16) may be called to disable subsystem break. A call to FCONTROL(17) enables it again.

Parameters

plabel `bit signed integer passed by value (required)`

Passes the **plabel** of your **[CTRL]Y** trap handling procedure. This **plabel** can be either an NM or a CM **plabel**. If this value is 0, XCONTRAP disarms **[CTRL]Y** traps for the process.

How you obtain the external **plabel** of your NM trap handling procedure depends on your programming language. In Pascal/iX, for example, you can obtain the **plabel** by using the **baddress** or **waddress** function. Supply the name of your **[CTRL]Y** trap handler as an argument to **baddress** or **waddress**.

If a program executes only in native mode, it should have a native mode **[CTRL]Y** trap handler. Programs executing in compatibility mode must have a compatibility mode **[CTRL]Y** trap handler.

If a program executes in both compatibility mode and native mode, it is preferable to have an NM trap handler. However, it is possible to pass a CM **plabel** for your CM **[CTRL]Y** trap handler as follows:

Obtain the 16-bit external CM **plabel** of your CM **[CTRL]Y** trap handler.

Pass this 16-bit **plabel** in the following 32-bit format:

Bits	Setting
(0:16)	16-bit external CM **plabel**
(16:13)	Reserved. Set to 0.
(29:1)	Set to 1.
(30:1)	Set to 0.
(31:1)	Set to 1.

oldplabel `32-bit signed integer passed by reference (required)`

Returns the **plabel** of the **[CTRL]Y** handler previously used by your process. This **plabel** can be either a CM or NM **plabel**, as described above. If no **plabel** was previously configured, **oldplabel** returns 0.

Condition Codes

CCE	Request granted. Trap enabled.
CCG	Request granted. Trap disabled.
CCL	Request denied because of an illegal **plabel**, or because XCONTRAP was called from a job.

Additional Discussion

See the discussion of enabling/disabling the user written **[CTRL]Y** trap handler in *Trap Handling Programmer's Guide*. See also the discussion of FCONTROL(16,17), FCONTROL(41), and RESETCONTROL in this manual. See Chapter 5, "Using Subsystem Break," of this manual for an example illustrating subsystem break processing.

A Comparing MPE/iX Systems to MPE V Systems

When upgrading from an HP 3000 computer using the MPE V operating system to an HP 3000 computer using the MPE/iX operating system, a variety of migration issues must be addressed. Differences between early and later versions of MPE/iX must also be considered.

This Appendix discusses migration from the ATP or ADCC subsystem on an MPE V computer to the Datacommunications and Terminal Subsystem (DTS) on an MPE/iX computer. The discussion includes comparisons of the following:

- Physical appearance.
- Configuration.
- PAD support.
- Terminal types and printer types.
- Software characteristics.

Physical Appearance

The most apparent change when migrating from MPE V to MPE/iX is the hardware configuration. The ATP/ADCC interface cards are placed directly in the SPU of MPE V computers. Therefore, the RS-232/422 cabling for asynchronous devices connects directly into the system cabinet. In comparison, MPE/iX computers use DTCs (Datacommunications and Terminal Controllers) connected as nodes on a LAN. The interface cards to which the cables of devices are connected reside in the DTC. To communicate with asynchronous devices, the SPU must hold only the interface card for the LAN.

Although the physical environment of the subsystem on MPE/iX computers looks very different from that of MPE V computers, the resultant functions and characteristics of the subsystem software are very similar. In fact, as long as an asynchronous device is supported on both systems, very little difference should exist between using the device on an MPE V computer and using it on an MPE/iX computer.

Configuration

Three areas of configuration have changed from MPE V to MPE/iX:

- General system configuration.
- Terminal and serial printer subsystem configuration.
- Terminal configuration settings.

These changes are discussed below.

General System Configuration

With MPE V computers, the INITIAL/SYSDUMP utility is used to configure the system. MPE/iX computers require you to use a utility called SYSGEN. SYSDUMP is an interactive program in which you answer questions to enter configuration information. SYSGEN is a command based interactive program. In SYSGEN, because you enter commands instead of answering questions, you have more control over the process of entering configuration data. SYSGEN also has other advantages over SYSDUMP.

For more information on SYSGEN, see I*ntroduction to MPE/iX for MPE V System Administrators*. Note that SYSDUMP is used to configure the terminal and serial printer subsystem but SYSGEN is not.

Terminal and Serial Printer Subsystem Configuration

For MPE V computers, you must configure the asynchronous subsystem by assigning device characteristics one-by-one through the use of INITIAL/SYSDUMP dialogue questions. For MPE/iX computers, you must use the< Node Management Configuration Manager (NMMGR) to configure the Datacommunications and Terminal Subsystem (DTS).

With NMMGR, device profiles are created. A profile allows you to specify values for a set of characteristics, such as a terminal type and line speed, and apply those values to as many devices as you wish by assigning the same device profile. This shortens the time needed to configure devices into the subsystem.

Alternatively, some MPE/iX networks can include a network management workstation running the OpenView DTC Manager. When the OpenView DTC Manager is used, some aspects of serial device configuration are offloaded from each MPE/iX system. These characteristics are configured on the OpenView Windows Workstation

through the interface provided by the OpenView DTC Manager. For more information on configuration with the OpenView DTC Manager see Using the OpenView DTC Manager.

Terminal Configuration Settings

With several exceptions, terminal configuration settings for supported terminals on MPE/iX systems are the same as the settings that were used for supported terminals on MPE V systems. The following configuration values are the only settings that you will need to change so that the terminal will operate properly when connected to a DTC:

- `RecvPace` (Receive Pacing) must be set to XON/XOFF.
- `XmitPace` (Transmit Pacing) must be set to XON/XOFF.

Note that the `EnqAck` setting can be left on although the ENQ/ACK protocol is not used.

PAD Support

On MPE V computers, PAD support is provided over the NS X.25 3000/V Link product, which must be purchased and configured on systems to which PAD access is required. On MPE/iX systems, PAD support is provided through the DTC/X.25 Network Access card which is mounted in the DTC. Access through a private or public PAD is available through the DTC to any MPE/iX computer on the LAN to which that DTC is connected (within configured security limitations).

PAD Programming Considerations

Programmatically, PAD support on MPE/iX is nearly identical to PAD support on MPE V. The only difference is with the FCONTROL37 intrinsic. On MPE V, the FCONTROL intrinsic has no effect. On MPE/iX, calling the intrinsic will cause the terminal type setting to be changed, but will have no impact on how the device actually operates.

Terminal Types and Printer Types

The following terminal types files are supported on the ATP and ADCC terminal/printer controllers with MPE V T-MIT or later (MPE V version G.01.00 or later):

- 6, 9, 10, 12, 13, 15, 16, 18, 19, 20, 21, 22, TTPCL18, TTPCL19, TTPCL22, and TTPCL26.

DTS on MPE/iX computers supports three terminal types and four printer types:

- Terminal type 10.
- Terminal type 18.
- Terminal type 24.
- Printer type 18.
- Printer type 21.
- Printer type 22.
- Printer type 26.

Table A-1 and Table A-2 provide an introduction to the MPE/iX terminal types and printer types by comparing them to the MPE V terminal types that are the most similar. See Table A-1 for the terminal types that apply to terminals, and Table A-2 for the MPE V terminal types and MPE/iX printer types that apply to printers.

Table A-1 MPE/iX Terminal Types Overview

MPE/iX Terminal Type	Similar MPE V Terminal Type	Differences
10	10	Enhanced XON/XOFF protocol on MPE/iX. No ENQ/ACK protocol on MPE/iX.
18	18	Enhanced XON/XOFF protocol on MPE/iX.
24	24	For terminal devices connected via PAD.

Table A-2 MPE/iX Printer Types Overview

MPE/iX Printer Type	Similar MPE V Terminal Type	Differences
18	18	Enhanced XON/xoff protocol on MPE/iX
21, 22, 26	21, 22, 26	Enhanced XON/XOFF protocol on MPE/iX. No ENQ/ACK protocol on MPE/iX. Printer initialization string of MPE V's TTPCL22 is used. Printer status checks are done less frequently on MPE/iX; status requests are not sent after each printed line.

As devices are moved from an MPE V system to an MPE/iX system, see Table A-3 and Table A-4 to decide which terminal or printer type to use with each of your asynchronous devices.

Table A-3 Terminal Type Migration for Terminals

MPE V Terminal Type	Description	Use MPE/iX Terminal Type	Comments
6, 9	Non-HP hardcopy device needing delays after linefeed or formfeed.	None	Devices that need delays are not supported on MPE/iX.
10	General HP CRT terminal using both ENQ/ACK and XON/XOFF protocol.	10	Only XON/XOFF protocol is used on MPE/iX.
12	8-bit character version of terminal type 10. Used with languages that need extended character sets.	10	Bits per character no longer defined by terminal type.
13	Terminal type 10 with no echo or ENQ/ACK protocol. Used for plotters or the HP 2601 printer.	None	Similar terminal type can be created with TTUTIL.
15, 16	8-bit and 7-bit HP 2635 hardcopy terminal.	None	These terminal types specify use of a backspace response action not supported on MPE/iX.
18	Terminal type 10 without ENQ/ACK protocol or a read trigger. Used with non-HP devices.	18	Enhanced XON/XOFF protocol is used on MPE/iX.
24	Terminal type used for PAD terminals supporting page block mode.	24	Terminal type used for PAD terminals supporting page block mode. (Note that the block mode support is for VPLUS block mode only.)

Table A-4 Terminal Type Migration for Printers

MPE V Terminal Type	Description	Use MPE/iX Printer Type	Comments
18	Non-HP devices or application printers	18	MPE V's terminal type 18 and MPE/iX's printer type 18 are exactly the same except that an enhanced XON/XOFF protocol is used on MPE/iX.
TTPCL18	Terminal type 18 with a printer initialization string and an XOFF timer. Used with HP 2687A.	18	The HP 2687 A is not supported on MPE/iX.
19	Remote serial spooled printer.	21	Status checking is done less frequently on MPE/iX. The printer initialization string is the same as MPE V's TTPCL22.
TTPCL19	PCL remote serial spooled printer.	21	Status checking is done less frequently on MPE/iX.
20	8-bit serial spooled printer.	22	Status checking is done less frequently on MPE/iX. Initialization string is the same as MPE V's TTPCL22.
21	Serial spooled printer with no status checking after XOFF.	21	Status checking is done less frequently on MPE/iX. Initialization string is the same as MPE V's TTPCL22.
22	8-bit serial spooled printer with no status checking after XOFF.	22	Status checking is done less frequently on MPE/iX. Initialization string is the same as MPE V's TTPCL22.
TTPCL22	PCL 8-bit serial spooled printer with no status checking after XOFF.	22	Status checking is done less frequently on MPE/iX.
26	For use with remote spooled printers over a buffered network.	26	Same as MPE V's TTPCL26.
TTPCL26	PCL for use with remote spooled printers over a buffered network.	26	Same as MPE V's TTPCL26.

Software Characteristics

The following are differences between the ASC software and the software that controls the ATP/ADCC:

- Changed functions. These are functions that have been implemented differently in the ASC software (and/or the DTC) from the way they are implemented for ATP/ADCC.
- Programming considerations. These are more functions that have been implemented differently. However, these functions apply only to programs written to control asynchronous devices.
- Functions no longer supported. These are functions that are supported with the ATP/ADCC software which are not supported with the ASC software.
- Related operating system changes. These are functions that are supported on MPE V computers but are not supported on MPE/iX computers.

NOTE

The following pages document all known differences between the functionality of ATP/ADCC software and the functionality of ASC software. Some of the following differences are very detailed, but are not likely to impact the migration of most customer sites.

Changed Functions

The following functions have been modified. See also “Programming Considerations” later is this Appendix for features that have changed.

Echo Facility

Pressing **[Esc]** is no longer necessary to turn the echo facility on or off. The MPE/iX **:SET** command is used now. The syntax used to turn echo on or off is as follows:

- `:SET ECHO = ON`
- `:SET ECHO = OFF`

XON/XOFF Protocol

With MPE V, the XON/XOFF protocol is used, but certain cases exist when the XON (DC1) and XOFF (DC3) characters are ignored. Therefore, it is difficult to determine when these characters will be acted on. With ASC software, these characters are always acted on when the XON/XOFF protocol is enabled.

For instance, XOFF (**[CTRL]-S** on terminals) is ignored in MPE V during reads or when no I/O is pending on the device. (This means that nothing is waiting to be printed on the device and no program is waiting for input from the device.) With MPE/iX, the **[CTRL]-S** character is not ignored (except during a binary read).

Therefore, during a read, if **[CTRL]-S** is pressed, the read appears to be suspended. Press **[CTRL]-Q** to echo the characters typed to the screen. Therefore, **[CTRL]-Q** must be pressed to continue and complete the read.

The only exception is during binary mode. During a binary read, an ASCII DC1 or DC3 can be a data character and not an XON or XOFF. The ASC software does not react to them as if they are protocol characters, but passes them through as data. Since these protocol characters are accepted as data, they will be echoed. However, when an XOFF is echoed, it will suspend the terminal as if an XOFF were sent to it from the DTC. This means that echo should be disabled while in binary mode.

Note that the XON/XOFF protocol characters are accepted as data only when coming from the device. When an ASCII DC1 or DC3 is sent from the DTC, the terminal will still react to the character as a protocol character.

Using Smooth Scroll

A terminal uses its buffer to store incoming data and makes use of the XON/XOFF protocol to keep the DTC from overflowing the terminal buffer. Some supported devices (namely the HP150 and HP293x terminals) support smooth scroll. When a device is in smooth scroll mode, it transfers data to its screen more slowly (to make the output look “smooth”) and therefore uses its buffer to store more incoming data. This is especially true at higher speeds such as 9600 and 19,200 bps.

As a terminal user, use the XON/XOFF protocol to monitor what is written to the screen by pressing **[CTRL]-S** (XOFF) and **[CTRL]-Q** (XON).

When both the terminal and its user are sending XONs and XOFFs to the DTC, the DTC has no way of knowing the source of the protocol characters. Therefore, it can appear that the **[CTRL]-S** and **[CTRL]-Q** characters are not being recognized.

To avoid this problem, press **[Stop]** instead of **[CTRL]-S** and **[CTRL]-Q**. When **[Stop]** is pressed, the terminal’s buffer stops processing data, so no more will be printed to the terminal screen. When **[Stop]** is pressed a second time, the terminal will resume processing data in and out of its buffer. (Therefore, the **[Stop]** key functions as a toggle switch.) In this way, the terminal controls which protocol characters are sent to the DTC.

Another way to alleviate the problem is to lower the terminal's transmission speed; devices running at 4800 bps or lower will not have this problem.

Parity Error

With an ATP on MPE V, a read that contains a parity error still completes normally. Then, all read data is discarded. With a DTC, the first parity error ends the read immediately and the read data is discarded.

Parity Error in EOR

With an ATP, a parity error in the EOR character prevents the read from completing normally. Unless a timer expires or a byte count is satisfied, the device can hang. This is not a problem with a DTC. The read completes when the parity error is detected and all data is discarded.

Stripped Characters

When the terminal controller reads input from a device, certain characters are searched for in the stream of data. When found, some of those are stripped from the data. The differences between the MPE V ATP and the ASC software are as follows:

- **[Esc]:** This two character sequence is not stripped from the input data on MPE/iX computers. It is stripped from input data on MPE V computers.
- **[Esc];** This two character sequence is not stripped from input data on MPE/iX computers. It is stripped from input data on MPE V computers.
- LF (linefeed). Linefeeds are stripped from input data with MPE V. With MPE/iX, linefeeds are not stripped.

System Break and Terminal States

When a system break is received, the ATP saves the EOR character, the read timer value, and the terminal mode. Under the same circumstances, the DTC saves all of these, plus echo. Therefore, with ATP, if echo was disabled, it will be disabled in the Command Interpreter (CI). With the DTC, echo is enabled and disabled when the user resumes.

Using [Break] (and Later :RESUME) During a Read

When you are entering data or a command in a subsystem and press **[Break]** (before pressing **[Return]**), MPE V keeps the data entered and completes the read if the **:RESUME** command is entered. MPE/iX disregards the read data.

Using [Break] During Writes

Because of the speed with which MPE/iX systems process writes, more data can be lost if **[Break]** is entered while a program is writing to a terminal than was the case for terminals connected to MPE V systems.

The Command Interpreter and [Break]

During a read in the MPE V CI, **[Break]** is disabled. It is not disabled in the MPE/iX CI.

Preemptive Writes

With MPE V, a preemptive write can interrupt a read or another write. With the ASC software on MPE/iX computers, a preemptive write will interrupt a read or a write not yet started, not a write that has already started.

During Reads. Suppose, you are using the CI to type in MPE V commands on a terminal. You want to use the **:SHOWJOB** command and begin typing. The system operator sends you a message with the **:WARN** command after you have typed only the characters **SH**. The **:WARN** message interrupts your read and is displayed on the screen. The **SH** characters you have already entered are discarded, and a new DC1 character is sent to your terminal. You must begin typing the word **SHOWJOB** again because the **SH** was lost when the read was preempted by the message.

With MPE/iX, a **:WARN** message will interrupt the read as if it were a **:WARN** message on MPE V. However, in the example above, when you have typed **SH** of the **:SHOWJOB** command and receive a **WARN** message, the **SH** is not lost, and no new DC1 character is issued. You need to type only **OWJOB** to execute the command.

During Writes. On MPE/iX, a preemptive write will not interrupt another write that has already started. Writes that have not yet started will be delayed until after the preemptive write.

Logical Console Functionality

On all HP 3000 computers, terminals other than the system console can temporarily obtain partial console functionality with the **:CONSOLE** command. This is called moving the logical console to a terminal. MPE V systems allow the logical console to receive console messages, execute normal console commands, and execute console commands that require the console attention character (**[CTRL]-A**).

The functionality of the logical console on MPE/iX computers is the same, with one exception: the logical console cannot execute console commands that require **[CTRL]-A**. The **[CTRL]-A** character is not

recognized as the console attention character on any device other than the system console. (Do not attempt to move the logical console to a terminal connected via a PAD or to a terminal with switching enabled.)

Printer Status Request

The printer status request, also known as status checking or the HP2631B handshake, is how the controller monitors the condition of serial printers. Under MPE V, printer status checking can be done quite frequently. Terminal type files 21 and 22 send a status request at FOPEN, after each record is printed, and at FCLOSE. Terminal type files 19 and 20 send an additional status request every time the printer sends an XOFF. With MPE/iX printer type files 21 and 22, a status request is sent only at FOPEN and FCLOSE. (Neither MPE V terminal type 18 nor MPE/iX printer type 18 uses status checking.) Since status checking is done less frequently with MPE/iX, serial printers have better performance. (Printer type file 26 on MPE/iX functions the same as terminal type 26 file on MPE V.)

Programming Considerations

The following functions have been implemented differently and apply only to programs written to control asynchronous devices.

Changing Parity

When programmatically changing the parity checking and generation of a specific connection (through the FCONTROL36 and FCONTROL24 intrinsics), the type of parity is determined by the value of a parameter of the FCONTROL36 intrinsic. The differences between MPE V's ATP and MPE/iX's DTC are listed in Table A-5.

Table A-5 Parity Settings with FCONTROL(36)

FCONTROL Control Code Value	ATP	DTC
0	Even parity checked on input Eighth bit set to 0 on output	No parity checked on input Eighth bit set to 0 on output
1	Odd parity checked on input Eighth bit set to 1on output	No parity checked on input Eighth bit set to 1on output
2	Even parity checked on input Even parity generated on output	Even parity checked on input Even parity generated on output
3	Odd parity checked on input Odd parity generated on output	Odd parity checked on input Odd parity generated on output

NOTE When a device is closed (using the FCLOSE intrinsic) on an MPE V computer, the initial default parity setting is restored. On MPE/iX computers, parity is not restored at FCLOSE. This means that to change parity using the FCONTROL intrinsic, the default parity in the application must be restored before closing the device.

Read Timer

On MPE V computers, the read timer must be enabled (FCONTROL20) before every read whose duration you wish to measure. With MPE/iX computers, the timer is always enabled and measures every read. This means the following:

- There is no need to use the FCONTROL20 intrinsic. The timer is automatically enabled.
- The read timer's value should be obtained before any other reads occur. Otherwise, the value of the read timer will be for the most recent read, not the read that originally needed to be timed.
- FCONTROL21 cannot be used to disable the timer. (Calling this intrinsic has no effect on the timer.)

Additional End-of-Record Characters

With MPE/iX, the characters listed in Table A-6 should not be used as additional end-of-record characters. An error message will not be received if one of the characters in the table is entered as an additional EOR, but the character will not terminate a read when it is entered. With MPE V, any character can be used as an additional end-of-record character, except for the list in Table A-7.

Table A-6 Unrecognized AEOR Characters (MPE/iX)

ASCII Character	Terminal Keys	Octal Code	Hex Code
NUL(Null)	**[CTRL]-@**	% 0	$ 0
DC1(XON)	**[CTRL]-Q**	% 21	$11
DC3(XOFF)	**[CTRL]-S**	% 23	$13
Current Subsystem Break	**[CTRL]-Y** (by default)[a]	% 31	$19
DEL(Rubout)	**[DEL]**	% 177	$7F

a. In transparent mode, define the subsystem break character through the FCONTROL41 intrinsic

On MPE/iX computers, only one AEOR character can be designated for a device at any given time with FCONTROL(25), and multiple AEORs can be designated with FDEVICECONTROL(192,40).

Table A-7 Unrecognized AEOR Characters (MPE V)

ASCII Character	Terminal Keys	Octal Code	Hex Code
NUL(Null)	**[CTRL]-@**	% 0	$ 0
BS(Backspace)	**[CTRL]-H**	% 10	$ 8
LF(Linefeed)	**[CTRL]-J**	% 12	$ A
CR(Carriage Return)	**[CTRL]-M**	% 15	$ D
DC1(XON)	**[CTRL]-Q**	% 21	$11
DC2	**[CTRL]-R**	% 22	$12
DC3(XOFF)	**[CTRL]-S**	% 23	$13
CAN(Cancel)	**[CTRL]-X**	% 30	$18
EM(End-of-Medium)	**[CTRL]-Y**	% 31	$19
ESC(Escape)	**[CTRL]-]**	% 33	$1B
DEL(Rubout)		% 177	$7F

Transparent Mode

On MPE V computers, the ASCII characters DC1 and DC3 are treated as data in transparent mode. With MPE/iX, DC1 and DC3 are not treated as data, but as the protocol characters XON and XOFF.

Binary Mode and Echo

When initiating binary mode (through the FCONTROL27 intrinsic) on an MPE/iX computer, turn echo off on the device (FCONTROL13). This is necessary because if a DC3 character is in the data, it will be echoed and will XOFF the device. Refer to “XON/XOFF Protocol” under “Software Characteristics” earlier in this appendix for more information.

Using FCONTROL(35)

The FCONTROL35 intrinsic is used to inhibit the printing of ! ! ! (three exclamation points) when the line deletion character (**[CTRL]-X** by default) is entered. When the line deletion response is disabled on MPE V computers, a carriage return and linefeed are sent when the line deletion request is received, and all read data is disregarded. The user can start typing the line from the beginning. On MPE/iX computers, the line deletion request functions the same except for one thing: the carriage return and linefeed are not sent.

The FDEVICECONTROL Intrinsic

MPE V computers, a limited number of device control functions can be modified through the FDEVICECONTROL intrinsic. On MPE/iX computers, FDEVICECONTROL has expanded device control functionality. Additionally, on MPE/iX computers, the FDEVICECONTROL intrinsic is supported for devices connected through a private or public PAD.

Functions No Longer Supported

The following functions supplied by the MPE V ATP/ADCC are not supported by the MPE/iX ASC software.

ENQ/ACK Protocol

The ENQ/ACK protocol handshake between devices and the HP 3000 MPE V ATP/ADCC is not supplied by the ASC software.

Transmission Speeds of 110, 150 and 600

The speeds 110 bps, 150 bps, and 600 bps are rarely used on the HP 3000 computer, so the DTC does not support them

:EOF: and :EOD: Commands

The **:EOF:** and **:EOD:** commands, which can be used on MPE V computers, are not supported on MPE/iX computers.

Carriage Control Delays

Some hardcopy devices could not receive data quickly enough, especially when the carriage of the device had to move back to the beginning of the next line after each line was printed. MPE V supports terminal type files that allow extra time for the device to prepare for the next line of output. It does this by providing extra time for each carriage return, each linefeed, and the formfeed that the device must perform. Since these devices are obsolete, Hewlett-Packard does not provide this feature with new terminal type files and printer type files.

Special Response to a Backspace Character

Some devices require the terminal controller to handle the backspace character in a specific way. For instance, when backspace is used on a hardcopy device, a character typed after the backspace will be difficult to see because it will be typed directly over the previous character. Some MPE V terminal type files react to a backspace by going to the next line, so that characters typed after a backspace would appear under, instead of on top of, previous characters. Since devices that require this special attention are not supported with the ASC software, only two responses to the backspace character are supplied. Through

the FDEVICECONTROL intrinsic, you can choose to either remove the character from input and back the cursor up one space, or to remove the character from input and erase it (backspace, space, backspace).

Special Response to a Linefeed Character

The MPE V ATP/ADCC software will react to a linefeed (LF) character by echoing a carriage return and linefeed, then stripping the linefeed from input data. The ASC software will not do this and will not strip linefeeds from input data. This is different during line block mode only.

Special Output for a Formfeed Character

For some devices, the MPE V ATP/ADCC software will substitute the linefeed (LF) character for the formfeed (FF) character. On MPE/iX computers, formfeed replacement is supported in carriage return, but not in user data.

Critical Writes

With MPE V, all writes to asynchronous printers are critical writes. This means that the file system reports the completion of the write to the program initiating the write only after the data from the write has been printed out. In other words, the write is complete only after the write is successful. With MPE/iX, the completion of every write is reported by the file system when the DTC sends the data to the printer. Since this is before the data has actually printed, the completion of a write does not guarantee that the write was successful.

Since the ASC software does not wait for each line to print before continuing, data can be sent to printers more quickly.

Related Operating System Changes

The following functionality of the MPE V operating system is not supported on the MPE/iX operating system, and has some significance for the asynchronous subsystem.

MPE Commands

The **:DATA** and **:COMMAND** commands, sometimes used to log on to terminals, are not supported on MPE/iX. The **:JOB** command, while still useful to stream jobs, cannot be used to log onto a terminal.

File System Intrinsics

The FCARD and PTAPE intrinsics, sometimes used with asynchronous devices, are not supported on MPE/iX.

A

ASCII A special interface card in the system cabinet through which the MPE/iX system console is connected.

ADCP Avesta Device Control Protocol. An HP proprietary protocol which provides device control features. ADCP is optimized for communications between a DTC and MPE/iX systems.

address A numerical identifier defined and used by a particular protocol and associated software to distinguish one node from another.

address key *See* **X.25 address key**.

address resolution In NS networks, the mapping of node names to IP addresses and the mapping of IP addresses to subnet addresses.

address resolution protocol (ARP) A protocol used to convert an IP address to a low level hardware address. ARP can be used only over a single physical network and is limited to networks that support hardware broadcast.

adjacent A node on a point-to-point network that is connected to another node by a single link with no intervening nodes.

ADP Active Distribution Panel. *See* **MDP.**

AFCP Avesta Flow Control Protocol. An HP proprietary protocol which provides data flow control features. AFCP is optimized for communications between nodes, including DTCs, in an MPE/iX environment.

ARP *See* **address resolution protocol**.

ARPA Advanced Research Projects Agency. It is the former name of the Defense Advanced Research Project Agency (DARPA). The Internet suite of protocols was developed under ARPA guidance.

ARPA DTC Telnet Access An HP product which is a card that can be installed in a DTC 48 and DTC 72MX. The Telnet Access Card performs protocol translation. Calls to the IP address of an MPE/iX system are detected by the Telnet Access Card and are retransmitted on the LAN using AFCP, allowing connections from terminals on ARPA systems to HP 3000s.

ARPA DTC Telnet Express Box An HP product which is a standalone hardware platform dedicated to performing protocol translation. Calls to the IP address of an MPE/iX system are detected by the Telnet Express Box and are retransmitted on the

LAN using AFCP, allowing connections from terminals on ARPA systems to HP 3000s.

ASCII American National Standard Code for Information Interchange. A character set using 7-bit code used for information interchange among data processing and data communications systems. The American implementation of International Alphabet No. 5.

asynchronous A device's mode of operation in which a sequence of operations are executed irrespective of time coincidence with any event. Devices that are directly accessible by people (for example, terminal keyboards) operate in this manner.

asynchronous processor board The new 24-port mux board for the DTC 72MX. Allows up to 24 direct and modem connections per board. A DTC 72MX can have up to 3 asynchronous processor boards (for a total of 72 connections).

Attachment Unit Interface

AUI. The cable that runs between each node (host, DTC, or other device) and the Medium Attachment Unit (MAU) that connects it to the LAN in a ThickLAN configuration.

autodial A dial link in which the remote node's telephone number is automatically dialed by a modem or other device with this capability.

B

backbone LAN A thick LAN cable conforming to the IEEE 802.3 Type 10 BASE 5 Standard.

back-to-back configuration A DTC configuration whereby MPE users connected to one DTC can communicate with a non-MPE/iX system connected to another DTC via the LAN. *See also* **Local Switching**.

banner A welcome message displayed on your screen. On the local OpenView workstation a banner appears when a remote connection is established with the OpenView DTC Manager. A banner also can appear when you log on to MPE.

baud The measure of the speed at which information travels between devices, most commonly used in reference to terminal speed settings. Baud represents signal events per second. When one bit represents each signal change, baud is the same as "bits per second."

binary mode A data transfer scheme in which no special character processing is performed. All characters are considered to be data and are passed through with no control actions being taken.

bit Binary digit. A unit of information that designates one of two possible states, which are represented by either 1 or 0.

block mode A terminal processing mode in which groups, or "blocks," of data are transmitted all at once.

BNC T-Connector A connector used to connect a computer or a component such as a DTC to the LAN in a ThinLAN configuration.

boundary *See* **network boundary**.

bps Bits per second. The number of bits passing a point per second.

bridge A device that is used to connect LAN segments.

broadcast Communication method of sending a message to all devices on a link simultaneously.

buffer A logical grouping of a system's memory resources used by NS3000/iX.

byte A sequence of eight consecutive bits operated on as a unit.

C

call In X.25, a call is an attempt to set up communication between two DTEs using a virtual circuit. Also known as a virtual call.

call collision A conflict that occurs at a DTE/DCE interface when there is a simultaneous attempt by the DTE and DCE to set up a call using the same logical channel identifier.

called address When a node sends out a call request packet, the packet contains the address of the destination node. The address of the destination node is the called address.

calling address When a node receives an incoming call packet, the packet contains the address of the sending node. The address of the sending node is the calling address.

carrier A continuous wave that is modulated by an information bearing signal.

catenet *See* **internetwork**.

CCITT Consultative Committee for International Telephony and Telegraphy. An international organization of communication carriers, especially government telephone monopolies, responsible for developing telecommunication standards by making recommendations. The emphasis is on "recommendations"; no carrier is required to adhere to a CCITT recommendation, although most do so in their own interests.

CIB The channel input/output bus in the backplane of an HP 3000.

circuit-switching network A type of data communications network wherein a physical and exclusive link is maintained between two communicating devices for the call duration. An

all digital, circuit switching network is often referred to as an X.21 network.

closed user group An X.25 user facility that allows communication to and from a pre-specified group of users and no one else.

compatibility mode A processing mode on HP 3000 Series 900 computers that allows applications written for MPE V/E based systems to be ported and run without changes or recompilation.

computer network A group of computer systems connected in such a way that they can exchange information and share resources.

configuration 1) The way in which computer equipment is physically interconnected and set up to operate as a system. 2) The layout of the computer system, including the MPE table, memory, and buffer sizes, that tells which peripheral devices are (or can be) connected to the computer and how they can be accessed. 3) The process of defining the characteristics of a network in software.

For MPE/iX based computers, the operating systems are configured through use of the SYSGEN utility.

Next, the Distributed Terminal Subsystem (DTS) link is configured by using NMMGR (running on the host) and can, in addition, be configured using the OpenView DTC Manager software (running on the OpenView Windows Workstation) depending on the type of network management you use.

A system that is to run network services (NS3000/iX) is configured through use of NMMGR.

Access to X.25 is configured in two parts. The X.25 MPE/iX System Access software is configured on the host through use of NMMGR. The DTC/X.25 Network Access software residing on the DTC is configured at the OpenView Windows Workstation through use of the OpenView DTC Manager.

configuration file The configuration file contains the information that the network needs in order to operate. This file also contains information necessary for link level and NetIPC logging. The only file name that the system recognizes is `NMCONFIG.PUB.SYS`.

control-X echo Three exclamation marks (! ! !) output to the terminal screen when the cancel character (normally **[CTRL]-X**) is entered.

control-Y trap A user written procedure to which control is passed when the subsystem break

character (normally **[CTRL]-Y**) is entered during execution of a program with subsystem break enabled.

cross-validate The process of assuring that information contained in two locations is consistent where it is imperative that it be consistent. For example, an automatic cross validation occurs when you enter SYSGEN to assure that information contained in `NMCONFIG.PUB.SYS` agrees with system configuration data.

CSMA/CD Carrier Sense Multiple Access with Collision Detect, transmission access method used by the IEEE 802.3 LAN standard.

CSN *See* **circuit-switching network**.

CTB The cache transfer bus in the backplane of an HP 3000.

CUG *See* **closed user group**.

D

data Basic elements of information that can be processed or produced by a computer.

Datacommunications and Terminal controller *See* **DTC**.

datagram A self contained packet that is independent of other packets. It does not require an acknowledgment and it carries information which is sufficient to route it from one DTE to another DTE without relying on earlier exchange between the DTEs.

data overrun Transmitted data that is sent faster than the receiving equipment can receive it. The resultant overflow data is lost. *See also* **flow control**.

Datapac The national public PSN of Canada.

Datex-P The national public PSN of West Germany.

D bit Delivery confirmation bit. Used in the X.25 protocol, the setting of the D bit in DATA packets indicates whether delivery acknowledgment of the packet is required from the local DCE or from the remote DTE. It therefore allows the choice between local and end-to-end acknowledgment.

DCE Data circuit terminating equipment. The interfacing equipment required in order to interface to data terminal equipment (DTE) and its transmission circuit. Synonyms: data communications equipment, dataset. A modem is an example of a DCE.

DDX The national public PSN of Japan.

DDFA DTC Device File Access Utilities. A set of HP-UX utilities which is used by systems and user written applications to programmatically access devices attached to DTC ports.

DDP Direct Distribution Panel; a distribution panel that serves as the electrical and physical interface between a DTC 72MX mux board (asynchronous processor board) and up to eight asynchronous devices for direct connections. *See* **MDP** for modem connections.;

dedicated printer A printer that can be used only by one host on the LAN—the one specified in the Destination Node Name in that printer's configuration screen.

demodulation The process by which the information bearing signal is retrieved from a modulated carrier wave. The inverse of modulation.

destination node name In DTS configuration, it is either 1) the name of a host that a user can be connected to by default (if switching is not enabled for that user, or if automatic modem connection is enabled), or 2) the name of the only host that can access a dedicated printer.

device class A collection of devices that have some user defined relation. Device classes are assigned through use of the NMMGR configuration program.

device-dependent characteristic A file specification for which modifications are restricted because of the type of device on which the file is opened. For example, data directed to terminals must have a blocking factor of one.

device driver A software module that controls a specific type of input/output device.

devicefile A file being input to or output from any peripheral device except a disk. MPE/iX allows operations to be performed on the device itself as if it were a file.

device independence A characteristic of the operating system that allows users to selectively redirect input/output from a program, session, or job without regard to the nature of the device.

device name *See* **PAD name**.

Dial ID protocol A proprietary Hewlett-Packard protocol that provides security checking and address exchange for dial links.

dial link A connection made through public telephone lines.

direct-connect device An asynchronous device that is connected directly to a DTC through an RS-232-C or RS-422 cable, with no intervening communications equipment. Also referred to as a "local connection."

direct connection A leased line, private line, or other non-switched link in a network.

direct dial A dial link through which only one remote node can be reached.

direct-path branching The process of directly accessing any screen in NMMGR by entering a path name in the **Command:** field. The path name must be preceded by an at sign (@).

download The process of loading operating code and configuration files into the DTC's memory. The DTC is downloaded by the MPE/iX host for LANs using host-based network management, and by the PC for DTCs managed by the OpenView DTC Manager.

driver Software that controls input/output devices including NS3000/iX links.

DTC Datacommunications and Terminal Controller. The DTC is a hardware device, configured as a node on a LAN, that enables asynchronous devices to access HP 3000 Series 900 computers. Terminals can either be directly connected to the DTC, or they can be remotely connected through a Packet Assembler Disassembler (PAD). The DTC can be configured with DTC/X.25 Network Access cards and DTC/X.25 Network Access software. A DTC/X.25 iX Network Link consists of two software modules: the X.25 iX System Access software (running on the host) and the DTC/X.25 Network Access software (running on the DTC).

DTC 16 HP 2340A product. A DTC that provides 2 slots for asynchronous connections with each slot allowing up to 8 direct connections or 6 modem connections, plus an optional X.25 link supporting up to 32 virtual circuits at speeds of up to 19.2 Kbps.

DTC 16iX HP J2062A product. A DTC terminal server for HP 3000 systems. It allows up to 16 asynchronous connections to HP 3000 systems.

DTC 16MX HP J2063A product. A DTC terminal server for HP 3000 and DTC Telnet systems. It allows up to 16 asynchronous connections for the HP 3000 environment and for computer systems running ARPA, such as HP 9000s and third party systems.

When managed by the HP 3000, the DTC 16MX is configured and functions exactly like a DTC 16iX. The full functionality of the DTC 16MX is only available when it is managed by the OpenView DTC Manager.

DTC 16TN HP J2060A product. A DTC terminal server for Telnet systems. It allows up to 16 asynchronous connections to computer systems running ARPA, such as HP 9000s and third party systems.

DTC 48 HP 2345B product. A DTC that provides 6 slots for asynchronous connections, X.25

links, or DTC Telnet Access. Each slot allows up to 8 direct connections or 6 modem connections. A DTC 48 may have up to 3 slots used for X.25 links, each supporting up to 256 virtual circuits at speeds of up to 64 Kbps or one Telnet Access Card with 40 Telnet connections to HP 3000 Series 900 connections.

DTC 72MX HP J2070A product. A DTC that provides 3 slots allowing up to 72 asynchronous connections. Each slot accommodates up to 24 direct or modem connections. A DTC 72MX may have up to 3 slots used for X.25 links, each supporting up to 256 virtual circuits at speeds of up to 64 Kbps. It may also use one slot for Telnet Access Card with 40 Telnet connections to HP 3000 Series 900 computers.

DTC identifier An identifier used only within NMMGR to define the branch of the configuration file containing information about a particular DTC. The identifier must begin with a letter and can be up to eight characters long.

DTC Manager *See* **OpenView DTC Manager**.

DTC node name A unique name used to identify a DTC on a LAN. The node name format is nodename.domain.organization, with each of the three parts having up to 16 characters. The name begins with either a letter or a digit.

DTC station address (802.3 address) A 12-digit hexadecimal number used to identify the DTC as a node belonging to the network configuration. Also called the LAN address or node address.

DTC switching A facility enabling terminal users to select any host system that they want to connect to. DTC switching is available only when the OpenView DTC Manager is used for network management.

DTC/X.25 Network Access The software that resides on the Datacommunications and Terminal Controller (DTC). To configure access to an X.25 network, you must configure two software components: the X.25 iX System Access (residing on the HP 3000 host and configured through use of NMMGR software), and DTC/X.25 Network Access (configured on the OpenView Windows Workstation through use of the OpenView DTC Manager software for PC-based management, and through NMMGR for host-based management).

DTC/X.25 Network Access card The hardware card and channel adapter that provides X.25 Network Access. It resides in the Datacommunications and Terminal Controller (DTC).

DTC/X.25 iX Network Link

Software and hardware that provides MPE/iX access to private and public X.25 networks. The

X.25 iX System Access software resides on an HP 3000 host and is configured through use of NMMGR. The DTC/X.25 Network Access software resides on the Datacommunications and Terminal Controller and is configured at the OpenView Windows Workstation for PC-based management and through NMMGR for host-based management.

DTE Data Terminal Equipment. Equipment that converts user information into data-transmission signals or reconverts received data signals into user information. Data terminal equipment operates in conjunction with data circuit-terminating equipment.

DTS Distributed Terminal Subsystem. This consists of all of the Datacommunications and Terminal Controllers (DTCs) on a LAN, their LANIC cards (attached to the host), the LAN cable, and the host and DTC software that controls all related DTS hardware.

duplex A transmission method that allows two way communication. If both ends of the transmission link can transmit simultaneously, it is called full duplex. If only one end can transmit at a time, it is half duplex transmission.

E

entry priority In a point-to-point network, it is a ranking that identifies the most desirable route for data to travel from a given local node to a remote node.

environment A session that is established on a remote node.

escape from data transfer character A character that allows a user who is connected to a host system through the DTC, to break that connection and return to the DTC switching user interface. The default is [CTRL]-K. This character is used only on networks managed by the OpenView Windows Workstation.

escape sequence A sequence of characters beginning with the escape character and followed by one or more other characters, used to convey control directives to printers, plotters, or terminals.

Ethernet A Local Area Network system that uses baseband transmission at 10 Mbps over coaxial cable and unshielded twisted pair. Ethernet is a trademark of Xerox Corporation.

event log One of three circular files stored on the OpenView windows workstation. It contains lists of events that are reported by the DTCs for which it is responsible.

extended packet sequence numbering One of the optional Network Subscribed Facilities that provides packet sequence numbering using modulo 128. If not subscribed, modulo 8 is used.

F

facility An optional service offered by a packet switching network's administration and requested by the user either at the time of subscription for network access or at the time a call is made. Also known as user facility.

facility set A facility set defines the various X.25 connection parameters and X.25 facilities that can be negotiated for each virtual circuit on a per-call basis.

first select An optional packet switching network facility by which user data can be transmitted as part of the control packets that establish and clear a virtual connection.

FCS Frame Check Sequence. A sequence of bits generated by X.25 at Level 2 that forms part of the frame and guarantees the integrity of its frame's contents. The FCS is also used by the IEEE802.3 protocol to check the validity of frames.

file equation An assignment statement used to associate a file with a specific device or type of device during execution of a program.

file number A unique number associated with a file when the file is opened. The file number is returned in the FOPEN or HPFOPEN call used to open the file. It can be used to access that file until the file is closed.

file specification The name and location of a file. The full specification for a file includes the file name, group, and account.

file system The part of the operating system that handles access to input/output devices (including those connected through the DTC), data blocking, buffering, data transfers, and deblocking.

flow control A means of regulating the rate at which data transfer takes place between devices to protect against data overruns.

flow control negotiation One of the network subscribed facilities selected at subscription time. This facility allows the Flow Control parameter to be negotiated at call set-up time, as opposed to having a predefined value.

formal file designator A name that can be used programmatically or in a file equation to refer to a file.

FOS Fundamental Operating System. The programs, utilities, and subsystems supplied on the

Master Installation Tape that form the basic core of the MPE/iX operating system.

full gateway A full gateway is a node that belongs to more than one network and has one IP address for each network. It uses store and forward to transfer packets between each network that it belongs to.

G

gateway A node that connects two dissimilar network architectures. A gateway can be either a single node (full gateway) or two gateway halves.

gateway half A node that works in conjunction with another node on another network to form an internetwork. The only protocol used by gateway halves is the NS Point-to-Point 3000/iX Link. See also full gateway.

gateway-half link A link between the two nodes of a gateway-half pair. Each of the two nodes of a gateway-half pair has a configured link (hardware interface card) that is used for the gateway half network interface. The NS Point-to-Point 3000/iX Link is the only link that can be used as a gateway-half link.

gateway-half pair A set of two nodes that are joined by a gateway-half link. Each node in the pair must have a gateway-half network interface configured, using the link.

Guided Configuration A method of configuring a node in which a subset of the complete NMMGR interface is presented, and defaults of configurable values are used automatically.

H

handshaking A communications protocol between devices or between a device and the CPU. Provides a method of determining that each end of a communications link is ready to transmit or receive data, and that transmission has occurred without error.

hardware handshake Uses modem signals CTS and RTS to pace the data transfer from the DTC to the attached device. (For DTC 72MX only.)

hop count *See* **internet hop count and intranet hop count**

host-based network management A method of managing asynchronous communications for HP 3000 Series 900 computers. All of the control software is configured on a single MPE/iX host and is downloaded to the DTCs that are managed by that host. With host based management, a permanent relationship exists between each DTC and the host. Terminal users can access only the single MPE/iX system that owns the DTC their terminal is connected to.

host computer The primary or controlling computer on a network. The computer on which the network control software resides. For HP purposes, it can also be used to distinguish the MPE/iX system (host) from the DTC.

HP block mode A block mode transmission method employed by HP computers where the system controls the block mode handshake. When HP block mode is used, the user program need not concern itself with data transfer protocol.

HP ARPA DTC Telnet Express A DTC dedicated to providing protocol conversion between Telnet on TCP/IP and AFCP to allow incoming calls from the ARPA environment to HP 3000 systems.

HP PPN Hewlett-Packard Private Packet Network. Hewlett-Packard's own packet switching X.25 network, which gives users full control over the administration and security of their data communication.

HP TS8 A terminal server that can support up to eight asynchronous serial connections. When used in back-to-back configuration, users can access HP 3000 MPE/V systems on it through a DTC.

I

idle device timeout A timeout defined by the **`Configure:CPU`** command. When the timer lapses, a device connected to the DTC user interface that is still inactive will be disconnected.

IEEE 802.3 A standard for a broadcast local area network published by the Institute for Electrical and Electronics Engineers (IEEE). This standard is used for both the ThinLAN and ThickLAN implementations of the LAN.

IEEE 802.3 multicast address

A hexadecimal number that identifies a set of nodes. This address is used for multicast delivery.

IEEE 802.3 nodal address A unique hexadecimal number that identifies a node on an IEEE 802.3 LAN.

initialization string A sequence of control characters used to initialize a terminal, printer, or plotter when a connection is established from a host on the network.

INP Intelligent Network Processor. The card residing in the back of an MPE V-based node that provides a point-to-point or X.25 interface.

interactive communications

Processing that allows users to enter commands and data at the terminal and receive an immediate response. Interactive processing occurs in session mode on MPE/iX systems.

internet communication

Communication that occurs between networks.

internet hop count The number of full gateways plus the number of gateway-half links that a packet must pass through in moving from source node to destination.

Intranet Protocol A protocol used to provide routing between different local networks in an internetwork, as well as among nodes in the same local network. The Internet Protocol corresponds to Layer 3, the Network Layer, of the OSI model. See also IP address.

internet routing Internet routing involves all the processes required to route a packet from a node on one network to a destination node on another network.

intrinsic A system routine accessible by user programs. It provides an interface to operating system resources and functions. Intrinsics perform common tasks such as file access and device control.

IP *See* **Internet Protocol**

IP address Internet Protocol address. An address used by the Internet Protocol to perform internet routing. A complete IP address consists of a network portion and a node portion. The network portion of the IP address identifies a network, and the node portion identifies a node within the network.

IP router A node in an IP network that connects two or more networks and provides address mapping between them. The router selects messages from incoming buffers and places them into the appropriate outgoing message queues.

ISO International Organization of Standards. An international federation of national standards organizations involved in developing international standards, including communication standards.

L

LAN Local Area Network. A collection of data communication systems sharing a common cable whereby each system can communicate directly with another.

LAN address *See* **DTC station address**.

LANIC *See* **Local Area Network Interface Controller**.

LANIC physical path The physical location (slot number) of the LANIC within the SPU.

LANIC Self-Test A ROM-based program on a LANIC card that tests and reports the status of the LANIC hardware.

LAP Link Access Protocol. The data link protocol specified by older versions (prior to 1980) of X.25 at Level 2 but still permitted and therefore usable. All new implementations of X.25 must use LAP-B, and all old implementations must migrate to LAP-B at a future date.

LAP-B Link Access Protocol-Balanced. The data link protocol specified by the 1980 version of X.25 at Level 2 that determines the frame exchange procedures. LAP-B must also be used over direct-connect NS Point-to-Point 3000/iX Links.

LCI Logical Channel Identifier. Local value on a network node which identifies the channel used to establish a virtual circuit (SVC or PVC) through an X.25 network.

ldev *See* **logical device number**.

leased line A data-grade telephone line leased directly to a subscriber and allocated specifically for the subscriber's needs.

line speed The speed at which data is transferred over a specific physical link (usually measured in bits or kilobits per second).

link name A name that represents a hardware interface card. The link name can contain as many as eight characters. All characters except the first can be alphanumeric; the first character must be alphabetic.

Local Area Network Interface Controller (LANIC) A hardware card that fits into the backplane of the HP 3000 Series 900 computer and provides a physical layer interface for IEEE 802.3 local area networks.

local connection *See* **direct connection**.

local node The computer that you are configuring or that you are logged on to.

local switching A feature of the DTC which permits back-to-back configuration (for connections to an HP 3000 MPE/V host), using two ports of the same DTC.

logging The process of recording the usage of network resources. Events can be logged to both the OpenView workstation and to the MPE/iX host.

logging class A number defining the severity of any given event logged. An operator uses the logging classes to specify which

events are to be logged. Class 1 (catastrophic event) is always logged.

logical device number (ldev)

A value by which MPE/iX recognizes a specific device. All DTC devices that are configured as nailed devices through the NMMGR configuration have ldev numbers permanently assigned. The DTC devices can then be accessed programmatically through use of their ldev number. Non-nailed devices have ldev numbers that are assigned from a pool of available ldev numbers for the life of their connection to a system. Each nailed port configured in NMMGR must have a unique ldev number.

log off The termination of a job or session.

log on The process of initiating a job or session.

logon device See session-accepting device.

loopback The routing of messages from a node back to itself.

LUG Local User Group. A list defined for a particular DTC and card that specifies which remote nodes this DTC can send data to and also which remote nodes this DTC can receive data from. See also Closed User Group.

M

map, network A drawing that shows the topology of the network. For networks managed by the OpenView DTC Manager a network map must be created through use of the OVDraw capability provided with the management software. A network map is also a hardcopy drawing used when planning a network. It shows network topology, node and network names, addresses, network boundaries (for an internetwork map), and link types.

mapping A set of characteristics that describe a route taken by messages to reach a destination node. This set of characteristics is configured with NMMGR at every node on a point-to-point network. One mapping is configured at each node for every other node on the network to which messages will be sent.

MAU Medium Attachment Unit. A device attached to a ThickLAN coaxial cable that provides the physical and electrical connection from the AUI cable to the coaxial cable.

M bit More data bit. Setting this bit in a DATA packet indicates that at least one more DATA packet is required to complete a message of contiguous data.

MDP Modem Distribution Panel; a distribution panel that serves as the electrical and physical

interface between a DTC 72MX mux board (asynchronous processor board) and up to eight asynchronous devices for direct or modem connections. Also called ADP.

MIT Master Installation Tape. A magnetic tape containing the Fundamental Operating System for an HP 3000 Series 900 computer.

modem modulator/demodulator. A device that modulates and demodulates signals. Primarily used for modulating digital signals onto carriers for transmission and for performing the inverse function at the receiving end. Modems are essential for transmitting and receiving digital signals over telephone lines.

modulo Value used as the counting cycle for determining the send sequence number (N(S)) of frames sent across an X.25 network.

modulation The process in which certain characteristics of a carrier signal are altered in accordance with the changes of an information-bearing signal.

MPE/iX MultiProgramming Executive iX The operating system of the HP 3000 Series 900 computers. The NS3000/iX network services operate in conjunction with the MPE/iX operating system.

multiplexer MUX. A device that allows multiple communication links to use a single channel.

N

nailed device A device with a permanently assigned ldev. The assignment is established through the system configuration of the MPE/iX host system. Nailed devices can be accessed programmatically through their ldev number. Nailed devices can also be assigned to more than one host.

native mode The run-time environment of MPE/iX. In Native Mode, source code has been compiled into the native instruction set of the HP 3000 Series 900 computer.

neighbor gateway A gateway that is in the same network as a given node.

NetIPC Network Interprocess Communication. Software that enables programs to access network transport protocols.

network A group of computers connected so that they can exchange information and share resources.

network address This can be either 1) the network portion of an IP address as opposed to the node portion, or 2) when referring to X.25 networks, it is a node's X.25 address.

network boundary The logical division between networks in an internetwork.

network directory A file containing information required for one node to communicate with other nodes in 1) an internetwork, 2) an X.25 network, or 3) a network that contains non-HP nodes. The active network directory on a node must be named `NSDIR.NET.SYS`.

network interface NI. The collective software that enables data communication between a system and a network. A node possesses one or more network interfaces for each of the networks to which it belongs. Network interface types are LAN802.3, router (point-to-point), X.25, loopback, and gateway half. The maximum number of supported NIs is 12, one of which is reserved for loopback.

network management The collective tasks required to design, install, configure, maintain, and if necessary, change a network.

network map A drawing that shows the topology of the network. For networks managed by the OpenView DTC Manager, a network map must be created using the OVDraw capability provided with the management software.

Network Services NS. Software application products that can be used to access data, initiate processes, and exchange information among nodes in the network. The HP 3000/iX Network Services include RPM, VT, RFA, RDBA, and NFT.

network subscribed facilities

A set of parameters that the user chooses when he subscribes to the X.25 network; they include Flow Control Negotiation, Use of D-bit, Throughput Class Negotiation and Extended Packet Sequence Numbering.

NFT Network File Transfer. The network service that transfers disk files between nodes on a network.

NI *See* **network interface**.

NLP Name Lookup Protocol. A protocol used when setting up links between DTCs, as happens when DTCs are used in a back-to-back configuration.

NMCONFIG.PUB.SYS The file that contains all of the network configuration data for the HP 3000 Series 900 computer on which it resides. It includes information about the DTCs that can access the system as well as information about any Network Service (NS) products running on the system. This is the only file name allowed at run-time.

NMDUMP A utility used to format log and trace files.

NMMAINT A utility that lists the software module version numbers for all HP AdvanceNet products, including NS3000/iX. It detects missing or invalid software modules.

NMMGR Node Management Services Configuration Manager. A software subsystem that enables you to configure DTC connectivity and network access parameters for an HP 3000 Series 900 computer.

NMMGRVER A conversion program called `NMMGRVER.PUB.SYS`. It converts configuration files created with NMMGR from an earlier version to the latest format.

NMSAMP1.PUB.SYS A sample configuration file supplied with FOS that can be used as a template for DTS configuration.

node A computer that is part of a network. The DTC is also considered to be a node and has its own address.

node address The node portion of an IP address. The IP address consists of a node portion and a network portion.

Node Management Services Configuration Manager *See* **NMMGR**.

node name A character string that uniquely identifies each system in a network or internetwork. Each node name in a network or internetwork must be unique; however, a single node can be identified by more than one node name.

node names list A list defined on the OpenView windows workstation and subsequently downloaded to all DTCs for which it is the "owner." The list specifies all of the HP 3000 Series 900 hosts on the LAN that are accessible from the DTCs.

non-adjacent Describes a node on an NS Point-to-Point 3000/iX network that is separated from a given node by intervening or intermediate node.

non-nailed device A session accepting device that is not permanently associated with an ldev number at configuration time. When the user at such a device logs on to an MPE/iX system, an ldev is assigned from a pool of ldevs set aside for this purpose at configuration time. The association between a non-nailed device and this assigned ldev exists only for the duration of the session. One advantage of the use of non-nailed device connections is that configuration is simplified, since it is not required that each non-nailed device be individually configured.

NS3000/iX A Hewlett-Packard data communication product that provides networking capabilities for MPE/iX based HP 3000

minicomputers. NS3000/iX consists of a link and network services.

NS3000/iX Link Software and hardware that provides the connection between nodes on a network. Some of the NS3000/iX links available are the ThinLAN 3000/iX Link and its ThickLAN option, the DTC/X.25 iX Network Link, the NS Point-to-Point 3000/iX Link, and the StarLAN 10 3000/iX link.

NS3000/iX Network Services

Software applications that can be used to access data, initiate processes, and exchange information among nodes in a network. The services are RPM, VT, RFA, RDBA, and NFT.

NSDIR.NET.SYS Name of the active network directory file. See also network directory.

O

octet An eight-bit byte operated upon as an entity.

OpenView HP OpenView Windows is HP's network management environment. It provides the basic services for accessing and managing networks used by the DTC Manager, and other applications, such as Switch/PAD Manager, Hub Manager, etc.

OpenView Admin An OpenView Windows program that enables you to configure how your OpenView Windows applications will function. For example, it enables you to set a default map for the OpenView DTC Manager.

OpenView Draw An OpenView Windows program that is used to draw the network map and to label the components on it.

OpenView DTC Manager An OpenView Windows application that enables you to configure, control, monitor, and troubleshoot the operation of the Distributed Terminal Subsystems on the LAN.

OpenView Run An OpenView Windows program that covers most of the control features used by the DTC Manager, including monitoring and diagnostic functions.

OpenView Windows The set of three programs: OV Admin, OV Draw and OV Run, running on the OpenView workstation under MS Windows, that acts as the platform for all OpenView applications, such as DTC Manager.

OpenView Windows Workstation The personal computer that provides software downloads to enable operation of the Datacommunications and Terminal Controller (DTC). The configuration software that runs on this workstation is called the OpenView DTC Manager software.

OSI model Open Systems Interconnection model. A model of network architecture devised by the International Standards Organization (ISO). The OSI model defines seven layers of a network architecture with each layer performing specified functions.

P

packet A block of data whose maximum length is fixed. The unit of information exchanged by X.25 at Level 3. The types of packets are DATA packets and various control packets. A packet type is identified by the encoding of its header.

Packet Exchange Protocol

PXP. A transport layer protocol used in NS3000/iX links to initially establish communication between nodes when NetIPC socket registry is used.

packet-switched network name The name of a data communication network adhering to the CCITT X.25 recommendation. This can be a PDN or a private network such as the HP PPN.

PAD (packet assembler/disassembler) A device that converts asynchronous character streams into packets that can be transmitted over a packet switching network (PSN).

PAD name A name of up to eight characters that is associated with a configured PAD device. The PAD name is known to both the DTC (defined by the DTC Manager) and the MPE/iX systems (defined by NMMGR) that the device can access.

PAD profile A terminal or printer profile that specifies the configuration characteristics for PAD-connected devices.

PAD support A software module which can be downloaded to an X.25 board in a DTC. The software is used to manage connections with remote PAD devices.

partner gateway half When gateway halves are used, two gateway halves are required in order to provide communication between two networks. Each is the partner of the other.

path name When configuring with NMMGR, you can type a string in the **COMMAND:** field on a screen to branch to another screen. Each screen has a unique path name that corresponds to its location in the hierarchy of configuration screens presented by NMMGR.

PDN Public data network. A data communication network whose services are available to any user willing to pay for them. Most PDNs use packet switching techniques.

point-to-point A link that connects either two nodes in a NS Point-to-Point 3000/iX network or two gateway halves.

port An outlet through which a device can be connected to a computer, consisting of a physical connection point and controlling hardware, controlling software, and configurable port characteristics. Ports can be thought of as data paths through which a device communicates with the computer.

Precision Architecture The hardware design structure for the HP 3000 Series 900 computer family.

printer name A character string of up to 16 characters specified in the DTC Manager configuration (for networks using OpenView Network Management) to define a printer by name. Can be shared by several printers (port pool).

printer profile A set of configuration characteristics that can be associated with one or more printers through the NMMGR configuration. Printer profile specifications include the printer type, line speed, device class assignment, and other values relevant to printers connected through a DTC.

printer type A collection of characteristics that cause a printer connected to an MPE/iX system to act and react in a specified manner. You can configure a printer to use one of the system-supplied printer types, or you can create custom printer types using Workstation Configurator.

privileged mode A capability assigned to accounts, groups, or users allowing unrestricted memory access, access to privileged CPU instructions, and the ability to call privileged procedures.

probe protocol An HP protocol used by NS3000/iX IEEE 802.3 networks to obtain information about other nodes on the network.

probe proxy server A node on an IEEE 802.3 network that possesses a network directory. A probe proxy server can provide a node with information about other nodes on the same or other networks of an internetwork.

profile A method of grouping device connection specifications and characteristics so that the set of characteristics can be easily associated with groups of like devices. See also printer profile, terminal profile.

program captive device *See* **programmatic device**.

Programmable Serial Interface PSI. A hardware card that fits into the backplane of the HP 3000 Series 900 computer. It provides a physical layer interface for NS Point-to-Point 3000/iX Links.

programmatic device A device operating under control of a program running on a computer. Programmatic devices can be used for input, output, or both, depending on the device and how it is opened by the controlling program.

protocol A set of rules that enables two or more data processing entities to exchange information. In networks, protocols are the rules that govern each layer of network architecture. They define which functions are to be performed and how messages are to be exchanged.

PSN Packet-Switching Network. Any data communication network in which data is disassembled into packets at a source interface and reassembled into a data stream at a destination interface. A public PSN offers the service to any paying customer.

PSS Packet Switching System. The national public PSN of the United Kingdom.

PVC Permanent Virtual Circuit. A permanent logical association between two physically separate DTEs that does not require call set-up or clearing procedures.

PXP *See* **Packet Exchange Protocol**.

Q

Q bit Qualified bit. When set in DATA packets the Q bit signifies that the packet's user data is a control signal for the remote device, not a message for its user.

QuickVal A software program that tests whether Network Services are operating correctly between nodes.

R

RDBA Remote Data Base Access. A network service that allows users to access data bases on remote nodes.

reachable network A network that can be accessed (with additional internet hops possibly required) by a particular gateway.

remote connect device An asynchronous device that is indirectly connected to a DTC through a modem and telephone hook-up or through a PAD.

remote node Any network node that is physically separate from the node you are currently using or referring to.

retransmission count (N2) The maximum number of times a frame will be retransmitted following the expiration of the Retransmission Timer, T1.

retransmission timer (T1) The length of time that a transmitter will wait for an acknowledgment from a destination address before

attempting to retransmit a frame. When choosing this value, factors like the line speed and maximum frame size should be taken into account.

RFA Remote File Access. A network service that allows users to access file and devices on remote nodes.

routing The path that packets or fragments of a message take through a network to reach a destination node.

RMP Remote Maintenance Protocol. HP proprietary protocol used in DTC management.

RPM Remote Process Management. A network service that allows a process to programmatically initiate and terminate other processes throughout a network from any node on the network.

RS-232-C The Electronic Industries Association (EIA) Level 1 protocol specification that defines electrical circuit functions for 25 connector pins. HP provides two implementations of this standard: a 3-pin version for direct connections up to a distance of 15 meters (50 feet), and a version which makes use of additional circuits and can be used for either modem or direct connections.

RS-422 The Electronic Industries Association (EIA) Level 1 protocol specification implemented by HP in a 5-pin version which can be used for direct device connection up to a distance of 1500 meters (4000 feet).

S

security string An alphanumeric character string that functions as a password for dial links. The security string is used by the Dial IP protocol.

serial device Any device that is attached to and communicates with a computer by means of a serial transmission interface. Terminals, printers, and plotters are among the devices that communicate serially with MPE/iX computers.

serial transmission A method of transferring data in which characters are transmitted one bit at a time and received one bit at a time in the order of transmission. This transmission scheme is employed by devices connected to the MPE/iX systems via the DTC.

session-accepting device A terminal or personal computer running in terminal emulation mode that is able to establish an interactive (conversational) session with an HP 3000 computer. Also referred to as a logon device.

shared dial A dial link that provides connection to more than one remote system, although to only one at a time.

shared-line access The feature that allows two or more HP 3000 Series 900 hosts to use the same DTC/X.25 Network Access card on a DTC to access an X.25 network.

SIC Serial Interface Card. A card installed in the front of the DTC that acts as an interface between a corresponding Connector Card (CC) and the DTC's processor.

slaved device A device that shares the same DTC port as another device and is connected, to the other device, referred to as its master, by a cable. The actions of the slaved device are controlled by the master device.

SNMP Simple Network Management Protocol. An industry standard, for managing networked computers in a multi-vendor environment.

SNP Synchronous Network Processor card; an alternative name for an X.25 board.

spooled device A printer that is accessed through the MPE/iX spooling facility. The spooling facility allows a nonsharable device to be shared among several users by temporarily storing output data on disk and managing the selection of output spool files destined for the spooled device.

start bit A data bit used to signal the start of a character being transmitted in an asynchronous communication mode.

station address A link level address used by the IEEE 802.3 protocol that is assigned to every node on an IEEE 802.3 network.

stop bit A data bit used to signal the end of a character being transmitted in an asynchronous communication mode.

store-and-forward A technique in which messages are passed from one node to another in a network to reach their destination. Point-to-point networks use the store-and-forward technique to transmit messages.

subnet Another name for a network, especially if the network is part of an internetwork. The word subnet is also a synonym for intranet.

SVC Switched Virtual Circuit. The path through an X.25 network that is established at call set-up time.

switching *See* **DTC switching**.

Switching User Interface The user interface available when DTC switching is enabled that allows terminal users to choose the MPE/iX computer with which they want to establish a communication link.

synchronous A mode of operation or transmission in which a continuous data stream is generated without intervals between characters. The data stream is synchronized by clock signals at the receiver and transmitter. As a result, fast transmission speeds (above 9600 bps) are attainable.

SYSGEN The software program that allows you to configure the operating system on HP 3000 Series 900 computers.

system configuration The method for telling MPE/iX which peripheral I/O devices are attached to the DTC and which parameters are required for system operation.

T

TCP *See* **Transmission Control Protocol**.

Telenet A proprietary public data network in the USA.

Telnet Access Card (TAC) A card that resides in the DTC 48 and provides protocol conversion between Telnet and AFCP.

Telnet Express Box *See* **HP ARPA DTC Telnet Express Box**.

Telnet/iX A direct HP 3000 implementation of the Telnet protocol, which provides both inbound and outbound Telnet connections using only HP 3000 resident code.

Telnet/iX Client An executable program file, telnet.arpa.sys, which enables HP 3000 users to have *outbound* Telnet access to HP 9000, HP 3000 and non-HP systems that support Telnet. Telnet/iX Client is available as of MPE/iX release 5.0, C.50.00.

Telnet/iX Server HP 3000 resident code that comes bundled with MPE/iX (as of release 5.5, C.55.00), which enables users on a remote system running standard Telnet service to logon and run most applications on the HP 3000.

TermDSM Terminal Online Diagnostic System Manager. A utility that provides diagnostic services for DTC connections by means of a series of commands accessible through the SYSDIAG utility. TermDSM is used only when DTCs are managed by an MPE/iX host system.

terminal name A character string of up to 16 characters specified in the OpenView DTC Manager configuration (for networks using OpenView Network Management) to define a terminal by name. It can be shared by several terminals (pool port).

terminal profile A set of configuration characteristics that can be associated with one or more terminals through the NMMGR configuration. Terminal profile specifications include the terminal type, line speed, device

class assignment, and other values relevant to terminals connected through a DTC.

terminal type A collection of characteristics that cause a terminal connected to an MPE/iX system to act and react in a specified manner. You can configure a terminal to use one of the system-supplied terminal types, or you can create custom terminal types using the Workstation Configurator.

ThinLAN 3000/iX LAN that conforms to the IEEE 802.3 Type 10 BASE 2 standard LAN.

throughput class A value assigned to a given virtual circuit that defines how many network resources should be assigned to a given call. It is determined by the access line speed, packet and window sizes, and the local network's internal mechanisms.

throughput class negotiation

One of the Network Subscribed Facilities defined at subscription time. This allows the user to negotiate the Throughput Class at call set-up time.

timer (T3) The length of time that a link can remain in an idle state. After the expiration of the timer, the link is considered to be in a non-active, non-operational state and is automatically reset. The value should be chosen carefully. In particular, it must be sufficiently greater than the Retransmission Timer (T1) so that no doubt exists about the link's state.

topology The physical arrangement of nodes in a network. Some common topologies are bus, star, and ring.

Transmission Control Protocol TCP. A network protocol that establishes and maintains connections between nodes. TCP regulates the flow of data, breaks messages into smaller fragments if necessary (and reassembles the fragments at the destination), detects errors, and retransmits messages if errors have been detected.

Transpac The national public PSN of France.

transparent mode A data transfer scheme in which only a limited number of special characters retain their meaning and are acted on by the system. All other characters are considered to be data and are passed through with no control actions being taken.

transport, network Software that corresponds to layers 3 and 4 of the OSI network architecture model. It sends data out over the communications link, receives incoming data, and routes incoming or outgoing data to the appropriate destination node.

TS8 *See* **HP TS8.**

TTUTIL Also known as the Workstation Configurator. A program, TTUTIL.PUB.SYS, on the HP 3000 that is used to create and modify terminal and printer type files.

Tymnet A proprietary public data network in the USA.

typeahead A facility that allows terminal users to enter data before a read is actually posted to the terminal.

U

unacknowledged frame number (K) The number of frames that can be transmitted without receiving an acknowledgment from the destination address. When this number (K) frame is reached, the same K frames are retransmitted.

unedited mode *See* **transparent mode**.

V

V.24 The CCITT recommendation that defines the function of the interchange circuits between a DTE and a DCE.

validation The process of ascertaining whether the network transport configuration file has been correctly configured. This is accomplished by using the NMMGR Validate Configuration File screen.

VAN Value Added Network. A data communication network that uses and pays for facilities belonging to another carrier. The value-added package is then sold to a user.

VC *See* **virtual circuit**.

virtual circuit A logical association between two physically separate DTEs.

Virtual Terminal A network service that allows a user to establish interactive sessions on a node.

VPLUS Software used to generate screens such as those displayed by NMMGR.

V-Series (V.##) CCITT A set of CCITT recommendations related to data communication over a voice-grade telephone network.

VT *See* **Virtual Terminal**.

W

WAN Wide Area Network. A data communications network of unlimited size, used for connecting localities, cities, and countries.

Workstation Configurator A utility available on MPE/iX systems, TTUTIL.PUB.SYS, that allows users to create customized terminal and printer types by entering data through a series of VPLUS screens.

X

X.3 The protocol that defines which user facilities should be internationally available from a packet assembler/disassembler (PAD) when this is offered by a public data network.

X.21 The protocol that defines the physical interface between a DTE and a DCE of a public data network where the access to the network is made over synchronous digital lines.

X.25 The protocol that defines the interface between a DTE and a DCE for packet mode operation on a Public Data Network (PDN).

X.25 address The X.25 address provided by the network administration if you are connected to a public data network (PDN).

X.25 address key An X.25 address key is a label that maps a node's IP address to its X.25 address and its associated X.25 parameters. You have a combined maximum of 1024 X.25 address keys in the SVC and PVC path tables.

X.25 LUG address X.25 address of a node belonging to a LUG.

X.25 iX System Access The software that works in conjunction with the DTC/X.25 Network Access software to provide MPE/iX access to X.25. The software resides on an HP 3000 host and is configured through use of NMMGR. To configure access to an X.25 network, you must configure two software components: the X.25 iX System Access and the DTC/X.25 Network Access (residing on the Datacommunications and Terminal Controller and configured at the OpenView Windows Workstation). Together, these two components provide a network connection on HP 3000 systems to private and public X.25 packet switched networks (PSNs).

X.29 The protocol that defines the interface for data exchange between a packet mode DTE and a remote Packet Assembly/Disassembly (PAD) facility over a packet-switching network.

XON/XOFF protocol The flow control used by MPE/iX systems to protect against data overruns. XON/XOFF protocol is controlled by the data recipient who sends an XOFF character (ASCII DC3) to the sender if it is unable to continue to receive data. The sender suspends transmission until it receives an XON character (ASCII DC1).

X.Series (X.##) CCITT recommendations A set of recommendations for data communication networks governing their services, facilities, and terminal equipment operation and interfaces.

Index