900 Series HP 3000 Computer Systems

Getting Started as an MPE/iX Programmer Programmer's Guide

HEWLETT PACKARD

HP Part No. 32650-90008
Printed in U.S.A. 1992

Second Edition
E0692

Printing History

The following table lists the printings of this document, together with the respective release dates for each edition. The software version indicates the version of the software product at the time this document was issued. Many product releases do not require changes to the document. Therefore, do not expect a one-to-one correspondence between product releases and document editions.

Edition	Date	Software Version
First Edition	November 1987	A.01.00
Update 1	July 1988	A.10.00
Update 2	December 1988	A.20.00
Second Edition	June 1992	B.40.00

Preface

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 PA-RISC architecture. MPE V software can be run on the PA-RISC (Series 900) HP 3000s in what is known as *compatibility mode*.

Getting Started as an MPE/iX Programmer is a manual designed to introduce a programmer to the Hewlett-Packard MPE/iX operating system, which runs on 900 Series HP 3000 computers. It provides:

- An overview of the operating system architecture, features, and facilities, explaining the concepts necessary for a programmer to obtain a working knowledge of the system.
- References to other manuals for detailed information, syntax, and advanced concepts.

As a prerequisite, you should review the self-paced training course *HP 3000 Series 900 Advanced Skills* (32650-60039).

Getting Started as an MPE/iX Programmer is part of the Programmer's Series. This series consists of the *MPE/iX Intrinsics Reference Manual* (32650-90028) and a set of task-oriented user's guides.

How to Use this Manual

If you are new to the subject of programming on the MPE/iX operating system, you should read chapter 1 first. If you are familiar with MPE/iX, turn directly to the chapter that contains the information you need.

For information on additional programming manuals refer to the *MPE/iX Documentation Guide* (32650-90144).

Organization of this Manual

The guide contains the following chapters:

Chapter 1	**Overview** covers the basics of programming on MPE/iX. It introduces the 900 Series HP 3000, HP-PA architecture, and MPE operating systems. It describes the following topics on a high level, suitable for management and programming purposes: MPE/iX operating system features and fundamentals, user interface, accounting structure, migration from the MPE V/E operating system, and data conversion from MPE V/E.
Chapter 2	**Utilities and Tools** covers programmatic access to the MPE/iX Command Interpreter and many other MPE/iX subsystems and utilities.
Chapter 3	**Program Development** covers writing, compiling, linking, loading, and running a program on MPE/iX. It discusses the multiprogramming environment, error detection, and control codes.
Chapter 4	**Link Editor** covers HP Link Editor/XL, which is a subsystem of MPE/iX used to bring pieces of code together into executable program files and maintain libraries of sharable code.
Chapter 5	**Optimizing a Program** covers the Optimizer subsystem of MPE/iX, which improves program performance.
Chapter 6	**File System** describes the MPE/iX File System, including file and record structure, File System services, file specifications, file domains, data transfer, and file security.
Chapter 7	**Data Management** covers data management concepts and subsystems on MPE/iX, including KSAM, ALLBASE/SQL, TurboIMAGE, and QUERY.

Conventions

`UPPERCASE`
: In a syntax statement, commands and keywords are shown in uppercase characters. The characters must be entered in the order shown; however, you can enter the characters in either uppercase or lowercase. For example:

```
COMMAND
```

can be entered as any of the following:

```
command        Command        COMMAND
```

It cannot, however, be entered as:

```
comm           com_mand       comamnd
```

italics
: In a syntax statement or an example, a word in italics represents a parameter or argument that you must replace with the actual value. In the following example, you must replace *filename* with the name of the file:

`COMMAND` *filename*

bold italics
: In a syntax statement, a word in bold italics represents a parameter that you must replace with the actual value. In the following example, you must replace ***filename*** with the name of the file:

`COMMAND(`***filename***`)`

punctuation
: In a syntax statement, punctuation characters (other than brackets, braces, vertical bars, and ellipses) must be entered exactly as shown. In the following example, the parentheses and colon must be entered:

`(`*filename*`):(`*filename*`)`

$\underline{\text{underlining}}$
: Within an example that contains interactive dialog, user input and user responses to prompts are indicated by underlining. In the following example, $\underline{\text{yes}}$ is the user's response to the prompt:

`Do you want to continue? >>` $\underline{\texttt{yes}}$

`{ }`
: In a syntax statement, braces enclose required elements. When several elements are stacked within braces, you must select one. In the following example, you must select either `ON` or `OFF`:

$$\texttt{COMMAND} \begin{Bmatrix} \texttt{ON} \\ \texttt{OFF} \end{Bmatrix}$$

`[ ]`
: In a syntax statement, brackets enclose optional elements. In the following example, `OPTION` can be omitted:

`COMMAND` *filename* `[OPTION]`

When several elements are stacked within brackets, you can select one or none of the elements. In the following example, you can select `OPTION` or *parameter* or neither. The elements cannot be repeated.

$$\texttt{COMMAND}\ \textit{filename} \begin{bmatrix} \texttt{OPTION} \\ \textit{parameter} \end{bmatrix}$$

Conventions (continued)

`[ ... ]`	In a syntax statement, horizontal ellipses enclosed in brackets indicate that you can repeatedly select the element(s) that appear within the immediately preceding pair of brackets or braces. In the example below, you can select *parameter* zero or more times. Each instance of *parameter* must be preceded by a comma:
	
	`[,`*parameter*`][...]`
	
	In the example below, you only use the comma as a delimiter if *parameter* is repeated; no comma is used before the first occurrence of *parameter*:
	
	`[`*parameter*`][,...]`

`| ... |`	In a syntax statement, horizontal ellipses enclosed in vertical bars indicate that you can select more than one element within the immediately preceding pair of brackets or braces. However, each particular element can only be selected once. In the following example, you must select **`A, AB, BA`**, or **`B`**. The elements cannot be repeated. $\left\{\begin{array}{c}\texttt{A}\\\texttt{B}\end{array}\right\}$ `| ... |`
...	In an example, horizontal or vertical ellipses indicate where portions of an example have been omitted.
Δ	In a syntax statement, the space symbol Δ shows a required blank. In the following example, *parameter* and *parameter* must be separated with a blank: `(`*parameter*`)Δ(`*parameter*`)`
( )	The symbol ( ) indicates a key on the keyboard. For example, (RETURN) represents the carriage return key or (Shift) represents the shift key.
(CTRL)*character*	(CTRL)*character* indicates a control character. For example, (CTRL)`Y` means that you press the control key and the Y key simultaneously.

Contents

Figures

Tables

1 OVERVIEW

This chapter introduces the 900 Series HP 3000 and the MPE/iX operating system, describing how they fit into the HP 3000 family of computers and MPE operating systems, in general. It describes hardware and software naming conventions and provides a brief overview of migration to the 900 Series from MPE V/E-based systems. It describes MPE/iX dual operating modes, features, and major subsystems.

It provides an overview of MPE/iX accounting structure, session and batch modes, how to execute MPE/iX commands, and the considerations necessary to convert files to MPE/iX.

Introduction to the HP 3000

The HP 3000 is a general-purpose multiprogramming machine, designed for the interactive, transaction processing environment of business and industry. The HP 3000 family of computers includes several models of the machine, each with a different series number.

900 Series HP 3000

The newest high performance members of the 900 Series HP 3000 family are based on HP Precision Architecture (HP-PA), a highly flexible computer design that can meet current user requirements and requirements arising during future growth.

HP Precision Architecture (HP-PA)

HP Precision Architecture (HP-PA) is based on Reduced Instruction Set Computer (RISC) concepts with added extensions for a complete system. This increases computer performance by reducing and simplifying the computer instruction set. HP-PA eliminates system overhead associated with conventional computer microcode by directly implementing computer instructions in hardware. The uniformity of HP-PA instructions enhances pipelining, providing higher performance by overlapping execution of multiple instructions. Many technologies can implement HP-PA; highly integrated VLSI designs can be achieved by eliminating the chip space required for microcode.

High performance from HP-PA architecture results from the memory hierarchy design and the use of optimizing compilers. Processor waiting time for memory accesses is minimized due to the following architectural characteristics:

- Frequently used instructions and data are stored in a large number of CPU registers.
- High-speed buffering of code and data occurs.
- Optimizing compilers generate efficient object code, allocate registers, and schedule instruction sequences to maintain efficient pipeline operation.

MPE Operating Systems

All HP 3000 models run under the Multiprogramming Executive (MPE) operating system. It is a disc-based operating system that manages all system resources and coordinates execution of all programs running on the system. The version of MPE used varies from one model to another. Figure 1-1 shows the on-line access capabilities common to all MPE operating systems.

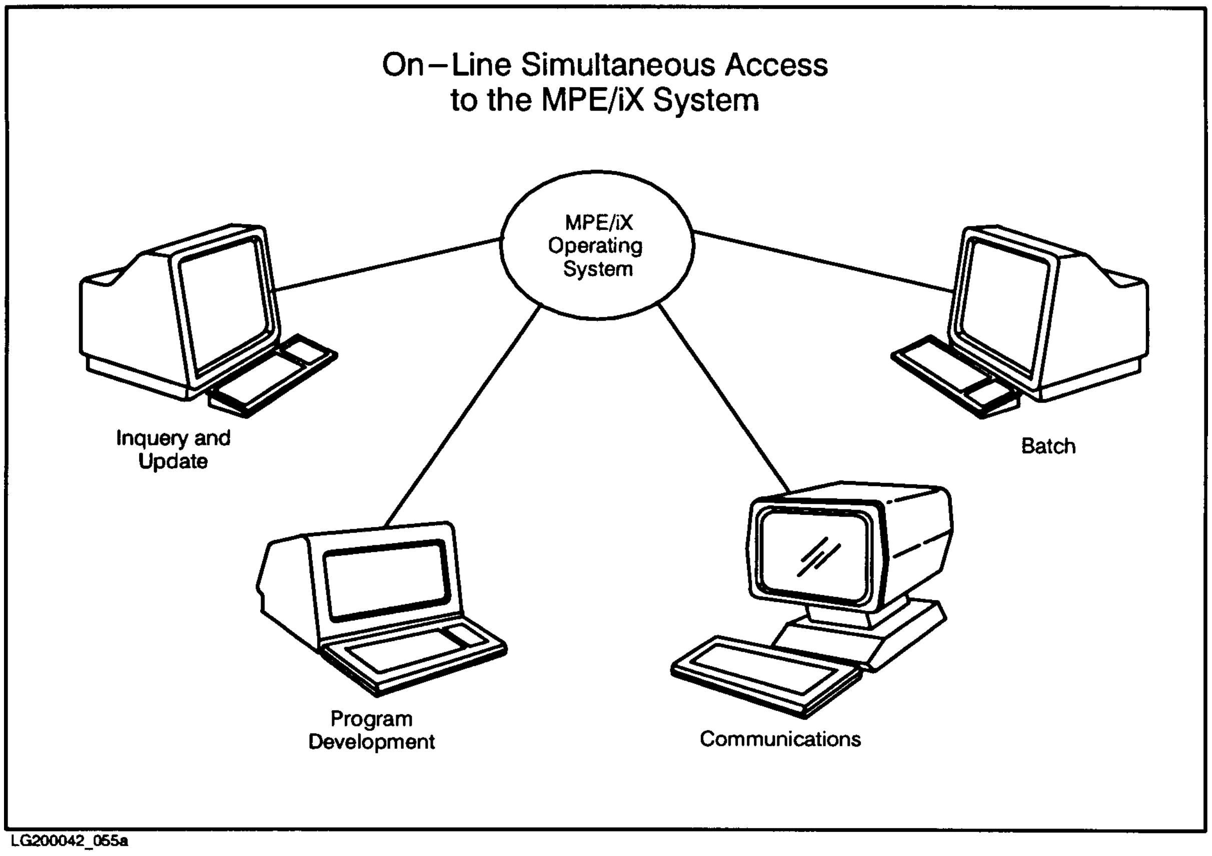


Figure 1-1. MPE/iX On-line Access Capabilities

The system simultaneously performs inquiry and update, program development, batch processing, and communications to other systems. All functions are available on-line (in sessions) or in batch mode (in jobs)submitted to the system. The same programs, files, and commands apply for both types of use. Figure 1-2 shows the high-level tools available on the 900 Series HP 3000.

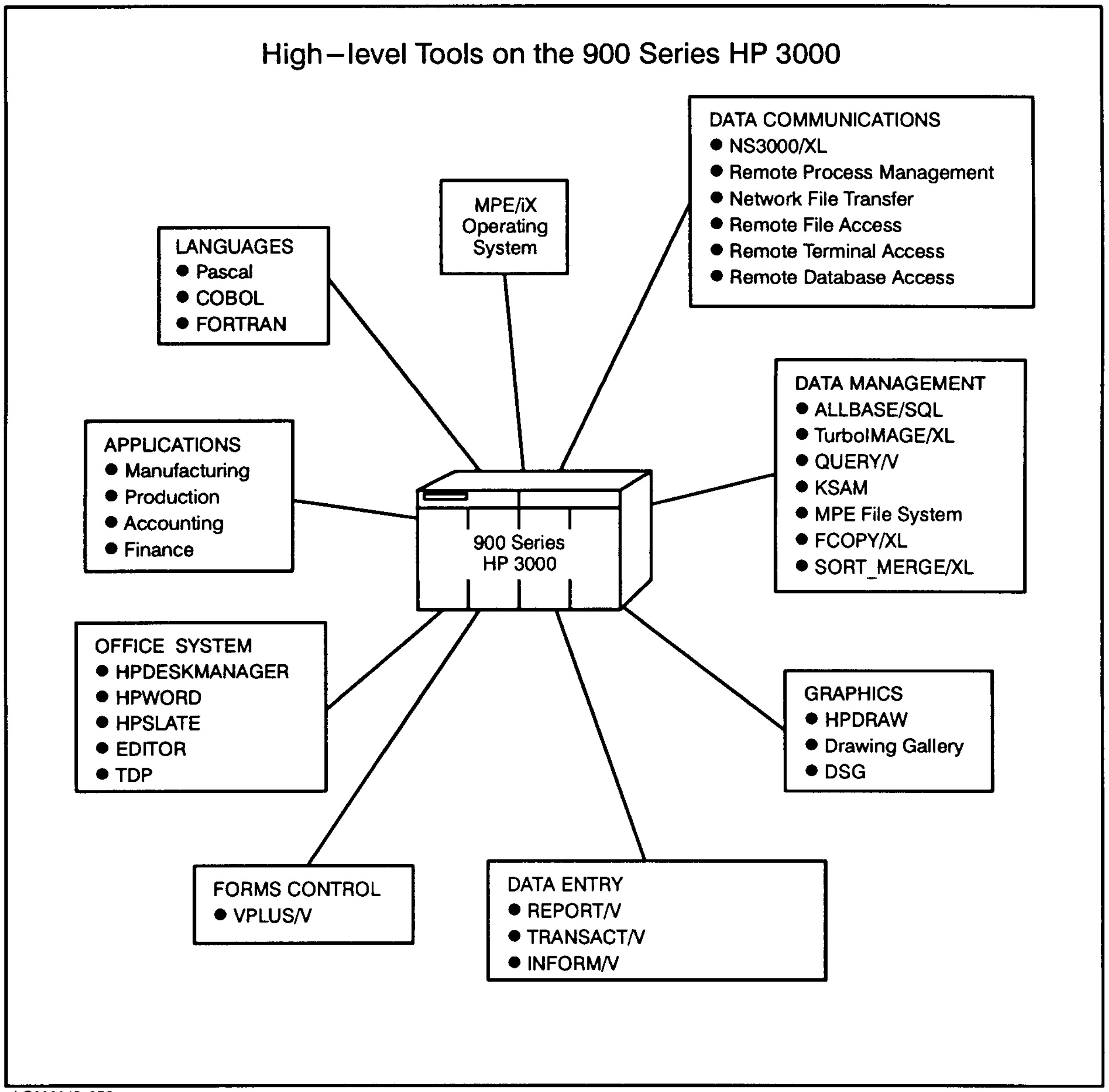


Figure 1-2. 900 Series HP 3000 High-level Tools

MPE/iX and MPE V/E Operating Systems

Since the 900 Series HP 3000 was introduced, Hewlett-Packard supports two types of MPE operating system; MPE V/E and MPE/iX, which stands for MPE with eXtended Large addressing. The "V" in the name MPE V/E is the Roman numeral for "5." MPE V/E is the operating system formerly called MPE. MPE V/E is supported on Series 37 through the Series 70 systems. MPE/iX is the operating system for 900 Series systems. It is designed to take full advantage of HP Precision Architecture (HP-PA). MPE/iX has the performance and ease-of-use of MPE V/E, plus additional functions and capabilities. It is upwardly compatible and has a user interface consistent with MPE V/E. MPE/iX is object code and source code compatible for programs developed on MPE V/E. It has two run-time environments that are transparent to the user:

- Compatibility Mode (CM), which provides object-code compatibility with MPE V/E-based applications.
- Native Mode (NM), which provides full performance benefits and advanced capabilities of HP-PA.

Naming Conventions for HP 3000 Systems and Software

When the 900 Series was added to the HP 3000 family of computers, Hewlett-Packard instituted a new naming convention for HP 3000 software products; the addition of the suffix "/V" or "/XL" to a product name. Software products that previously had the suffix "/3000" now have the suffix "/V." For example, the product named IMAGE/3000 is now named IMAGE/V, and the product named COBOL/3000 is now named HP COBOL/V.

The "/V" suffix indicates that a product is designed for use with the MPE V/E operating system. This product can also be used with the MPE/iX operating system running in Compatibility Mode. A compiler with the suffix "/V" (for example, RPG/V) generates object code that runs under MPE V/E and MPE/iX (in Compatibility Mode).

The "/XL" suffix indicates that a product is designed for use with MPE/iX running in Native Mode. A compiler with the suffix "/XL" (for example, HP COBOL II/XL) generates object code that runs with MPE/iX in Native Mode. Figure 1-3 shows an example of naming conventions.

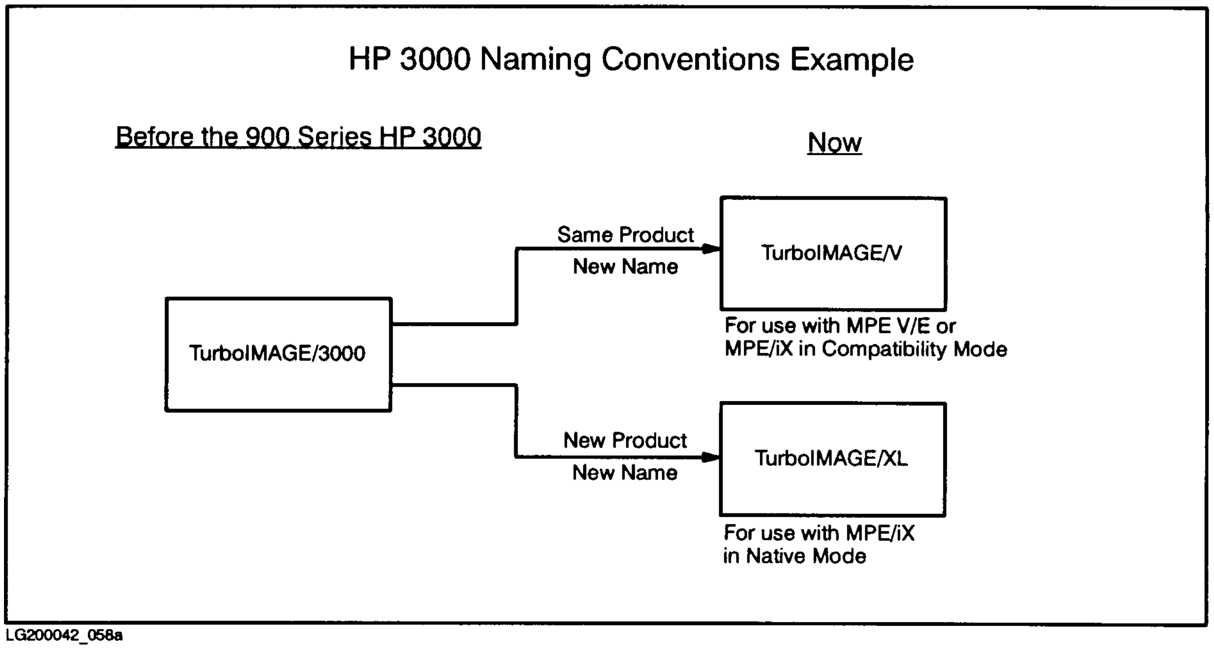

HP 3000 Naming Conventions Example
Before the 900 Series HP 3000
Now
Same Product
New Name
TurboIMAGE/V
For use with MPE V/E or MPE/iX in Compatibility Mode
TurboIMAGE/3000
New Product
New Name
TurboIMAGE/XL
For use with MPE/iX in Native Mode
LG200042_058a

Figure 1-3. HP 3000 Software Naming Conventions Example

Computers in the HP 3000 family that operate under the MPE V/E operating system are frequently called "MPE V/E-based systems." Computers in the HP 3000 family that operate under the MPE/iX operating system in native mode are frequently called "MPE/iX-based systems." This includes 900 Series systems.

Native Mode and Compatibility Mode

MPE/iX provides two run-time execution environments: Native Mode (NM) and Compatibility Mode (CM). MPE/iX dynamically and transparently coordinates and changes modes as required by applications.

NM is the native MPE/iX run-time environment. Source code has been compiled into the 900 Series native instruction set. NM is the preferred environment for the 900 Series and provides the highest performance from the systems through the use of demand paged virtual memory and memory mapped files.

CM provides object code compatibility between MPE V/E based systems and 900 Series computers. CM allows you to move applications and data to 900 Series computers without changes or recompilation.

MPE/iX is a compatible superset of MPE V/E. CM provides a working MPE V/E environment, including MPE V/E code and stack structures and most callable MPE V/E system intrinsics.

MPE/iX Mixed Modes

Applications can run partly in Native Mode (NM) and partly in Compatibility Mode (CM). MPE/iX transparently switches between modes for system routines. MPE/iX has a Switch Subsystem that determines if code is in NM or CM and automatically switches between them, as needed, while the routine is running. When the called routines are in the other mode, users must write their own switching routines.

900 Series Migration

The use of Compatibility Mode (CM) and migration utilities provides smooth, flexible migration to 900 Series systems. The high degree of object code compatibility between MPE V/E and MPE/iX operating in CM allows you to store any MPE V/E based application object code program written in a language supported on MPE V/E, restore it on a 900 Series system, and run it in MPE/iX Compatibility Mode. You can move data bases to a 900 Series system in the same way.

900 Series systems are fully upward compatible with other systems in the HP 3000 family. Migration tools are provided to facilitate upgrading to the 900 Series in stages, as your schedule permits, without interruption of operations. Migration to the 900 Series provides:

- Object code compatibility: a simple store/restore procedure allows you to move MPE V/E applications and data to the 900 Series. You can back up applications and databases on a tape and restore them on a 900 Series system without modification. The applications and databases run on the 900 Series in CM.
- Source code compatibility: you can recompile applications for maximum performance by using Native Mode (NM) optimizing compilers, which improve performance. You can achieve maximum database performance by using software utilities to transform databases into NM.

- Migration flexibility: upgrading to the 900 Series is extremely flexible because applications and databases can access and communicate with each other when they are in different modes (in other words, when one is in CM and the other is in NM).
- Operational flexibility: MPE/iX is a functional superset of MPE V/E. They are nearly identical in terms of user interface, system management, accounting, and security.
- Peripheral compatibility: Because of common I/O mechanisms, the 900 Series supports many of the same peripherals and workstations as other HP 3000 systems do.
- Cross-family development: CM allows you to develop applications on the 900 Series that can also be run on MPE V/E-based systems.

Migration to the 900 Series is flexible because you can move some applications to NM and move others to CM. Both kinds of applications can access the same database, and it can be in CM or NM. You can immediately move all your applications and databases to CM and can migrate applications to NM at convenient times.

Object Code Translator

MPE/iX provides an Object Code Translator (OCT) that can be used to translate MPE V/E object code into native instructions for the 900 Series. This improves performance over that of the MPE V/E object code.

Native Mode Compilers

To take full advantage of 900 Series performance capabilities, you can recompile MPE V/E based applications using Native Mode (NM) compilers for the 900 Series. They provide source code compatibility with the rest of the HP 3000 family of computers. Usually, applications require little or no code modification. The NM compilers available on MPE/iX include:

- HP C/iX
- COBOL II/XL
- HP Pascal/iX
- HP FORTRAN 77/iX

Data Base Manipulations

For improved performance of database manipulations, you can move to ALLBASE/SQL, the Native Mode Database Management System for the 900 Series. Utilities are available for converting to ALLBASE/SQL from TurboIMAGE/XL.

Migration Restrictions

Minor restrictions may apply to migrating MPE V/E-based applications to a 900 Series system. An application may require modification if it uses:

- Undocumented intrinsics
- Privileged machine instructions
- Unsupported hardware
- Architecture-dependent information

Applications written in SPL/V, the systems programming language for MPE V/E-based systems, can run in Compatibility Mode (CM) on a 900 Series system, but cannot migrate to

Native Mode (NM) because SPL/V has a high dependence on the MPE V/E-based HP 3000 architecture. However, you can improve the performance of an SPL/V application running in CM by using the MPE V/E Object Code Translator.

If you require NM performance, these applications should be rewritten in HP C/iX or HP Pascal/iX, which are systems programming languages for MPE/iX. If an application written in a high-level language calls SPL/V procedures, you can recompile it in NM. It runs in NM and uses a user-supplied mode switching procedure to switch to CM to call SPL procedures.

For information on less frequently encountered exceptions, refer to the *Migration Series* of manuals. For information on switching, refer to *Switch Programming Guide* (32650-90014).

Cross-family Application Development

You can develop applications on a 900 Series system for use on HP 3000 systems that use HP Precision Architecture (HP-PA) by using a 900 Series for centralized development. You can compile the source code for programs written to run on a Native Mode compiler on the 900 Series system and compile it to run on MPE V/E-based systems.

Similarly, you can compile source code written to run in Compatibility Mode on the 900 Series and compile it to run on MPE V/E-based systems. The MPE V Segmenter is supplied with MPE/iX to facilitate cross-family development in these languages.

MPE/iX Features

The main features of the MPE/iX operating system on a 900 Series computer include:

- Multiprogramming: concurrent transaction processing, data communications, on-line program development, and batch processing.
- Extended large addressing: 48-bit virtual addressing.
- Demand paged virtual memory, which transparently manages virtual memory and eliminates the need for program segmentation.
- Mapped disc files, which eliminates the need for File System buffering for disc files. This increases system performance for I/O-intensive applications.
- Concurrent multilingual capability, including HP-extended versions of C, COBOL, RPG, FORTRAN, BASIC, and Pascal.
- File System, which includes file backup, user logging, security, and interprocess communication (IPC).
- Access security and complete accounting resources.
- Command interpreter, which includes user-defined commands (UDCs), command files, conditional job control, extensive on-line help facility, and descriptive error messages.
- Device and file independence, which simplifies application development and maintenance.
- I/O System, which provides input/output spooling and a tape label facility.
- Complete, automatic local and remote terminal management.
- Power fail/automatic restart.

- Interactive Debug facility, which provides windows that allow you to simultaneously see the environment of the program being tested. It supports breakpoints, single stepping, calculation of expressions, macros, and command files.

Multiprogramming

MPE/iX supports multiprogramming, the concurrent execution of multiple programs. All system resources are available to you as if you were the only user on the system. While one program is waiting for input, the system shifts control of the CPU to the next highest program in the queue. In this way, activities such as transaction processing, on-line program development, interactive data entry, data communications, and batch processing can be concurrently performed.

MPE/iX is a multiprogramming, multiuser system. On this kind of a system, multiple users can share code. For example, when multiple users access the BASIC/V interpreter, a separate process is created for each one. They all use the same code (because there is only one BASIC interpreter on the system), but each user has a unique environment created by MPE/iX. MPE/iX completely protects one program execution from interfering with another.

Interactive and Batch Processing

MPE/iX provides interactive and batch processing. An interactive process is called a session. A batch process is called a job.

In a session, you enter commands and data at a terminal and receive an immediate response. This is especially useful for data entry and retrieval, program development, text editing, and any application that is expedited by direct dialogue with the computer.

In batch processing, you submit a job to the computer. A job is a single unit composed of commands that request various operations, such as program compilation and execution, file manipulation, or utility functions. While a job is processing, there is no user interaction with the computer unless the job is set up to request information. Jobs can be scheduled to run at lower priorities than interactive sessions and at specific times (for example, when system activities are low).

MPE/iX Information Management

Commercial applications primarily use database applications. The 900 Series provides ALLBASE/SQL, which includes TurboIMAGE/XL, a network database management product, and HP SQL, which has a relational interface to data. ALLBASE/SQL provides a TurboIMAGE/XL cross-development environment. The relational interface, HP SQL, is fully compatible with the version of SQL in most common use. It provides relational access for increased flexibility.

ALLBASE/SQL and other tools form Hewlett-Packard's information management framework. These include:

- Programming languages and tools.
- Reporting and presentation tools that allow access to information without programming.

- A common data dictionary that provides the integration necessary to tie the system together.

The products that meet these needs on the 900 Series are:

- System Dictionary/XL, which provides programmers and system administrators with a single source for documenting all aspects of the system, from data definitions to configuration information. This central source of information aids in developing and maintaining applications and effectively managing system resources. The System Dictionary/XL has programmatic interfaces for easy integration with other software. You can customize System Dictionary/XL to meet your needs.
- Optimizing compilers, which use HP Precision Architecture (HP-PA) to allow all programming on the 900 Series computers to be done in high-level languages. The compilers are integrated with MPE/iX to provide convenient access to ALLBASE/SQL and other information managements subsystems.
- VPLUS/V, which is a forms design and screen-handling tool for programmers.
- Toolset/XL, which includes facilities for full screen editing, symbolic source-level debugging, and version management of source code. It provides a high-productivity, integrated environment for application development.
- Transact/XL, which is a procedural, high-level programming language for transaction processing applications. It provides the functions of a high-level language, such as COBOL, combined with a comprehensive set of powerful verbs that can perform several functions in a single call.

Self-adjusting System Tables

Most system tables in MPE/iX are self-adjusting. MPE/iX continuously monitors and adjusts tables to fit the workload. Usually, the system manager can do this without shutting down the system.

On-line Diagnostics for Peripherals

Hewlett-Packard Customer Engineers (CEs) are equipped with on-line diagnostic tools for many HP peripheral devices. A CE can remotely run diagnostics without shutting down the system.

Disc Failure Tolerance

MPE/iX allows any system disc not critical to the functioning of the operating system to go off-line without affecting the system. Users cannot access files on off-line discs.

Automatic Power Fail Recovery

MPE/iX and the 900 Series hardware provide automatic power fail recovery. When a power failure starts, the system initiates a power failure procedure to preserve the operating environment before the complete loss of power. A battery pack ensures the validity of main memory for a minimum of 15 minutes. If power is restored within 15 minutes, the system automatically resumes processing from the point at which the power failure occurred, and jobs continue from the point of interrupt.

MPE/iX Transaction Management Facility

The MPE/iX Transaction Management Facility provides the following functions for transaction-oriented applications:

- Automatic transaction locking
- Automatic transaction logging
- Automatic rollback recovery from "soft" failures
- Semi-automatic rollforward recovery from "hard" failures

These functions are described in detail in the subsections below.

Transaction Locking

A transaction is a series of data updates that must be entirely completed to obtain logical consistency; either all or none of the updates in the series must be done. To the user, a transaction may appear to be a single change. However, internal to the system, it may require many changes to records in several files or data sets.

For example, if you enter a customer order on the system with a simple command, it may internally require updating several files containing data on orders, customers, and inventory requirements. The multiple changes, all of which are required for logical consistency, form one transaction.

The MPE/iX Transaction Management Facility ensures data consistency and integrity by providing automatic transaction locking. Transaction locking meets two criteria:

- Only one transaction at a time is allowed to update a given portion of data.
- All changes that are part of a transaction must be completed before the changes are committed to disc (permanently recorded). When a transaction is abnormally terminated before it is completed, the changes made up to that point are not committed.

MPE/iX transaction locking is based on page-level protection of the system architecture and does not require significant CPU overhead.

Transaction Logging and Recovery

The Transaction Management Facility automatically generates and maintains a transaction log file that records all transaction updates. Maintaining a log file facilitates recovery from the abnormal end of transactions and system failures. In these cases, files can be restored to a consistent state by copying the contents of the log file into the data file. This "undoes" the actions of partially committed transactions.

System failures are either hard or soft. When a soft failure occurs, data is not altered or lost, but some incomplete transactions may exist. In this case, the Recovery Manager portion of the Transaction Management Facility is automatically invoked to perform recovery of the data file when the system is restarted. Files are restored to their original state by copying the "before image" of data from the log file for incomplete transactions. This is called rollback recovery. Recovery from a soft failure is efficient and takes a maximum of only a few minutes. Rollback recovery is automatically performed for abnormally ended transactions.

When a hard failure occurs (for example, a disc media failure), data is lost. At system restart, you must mount a backup tape and issue the MPE/iX command `:RECOVER`. The

Transaction Management Facility returns data to a consistent state by reapplying all committed transactions in the log file to the checkpoint presented by the backup tape. This is called "rollforward recovery."

Transaction logging requires little CPU overhead because it is designed as an integral part of the MPE/iX File System, utilizing the addressing and protection features of HP Precision Architecture (HP-PA).

Simplifying a Program

The Transaction Management Facility simplifies development and maintenance of transaction-oriented applications by:

- Maintaining recovery routines, so a programmer does not have to develop and maintain custom ones.
- Simplifying the programming task of ensuring data integrity and increasing its efficiency.
- Providing locking and logging, thus requiring the programmer to mark only the beginning and end of transactions.

DEBUG

DEBUG is a low-level assembly language debugger, requiring some knowledge and familiarity in the following areas to utilize:

- Assembly code
- Procedure calling conventions
- Parameter passing conventions
- HP3000 and HP Precision Architecture (HP-PA)

Hewlett-Packard offers two source level, symbolic debuggers, SYMBOLIC DEBUG/XL and Toolset/XL, which you can use if you do not require assembly language debugger features.

DEBUG is an intrinsic procedure, providing privileged and non-privileged users with an interactive debugging facility for checking out their operating environments. Using DEBUG, it is possible to:

- Set, delete, and list breakpoints in a program. The program executes until a breakpoint is reached, then stops and passes control to the user. When you set breakpoints, you can specify a list of commands that will automatically be executed when the breakpoint is hit.
- Single step (multiple steps) through a program.
- Display and/or modify the contents of memory locations. A full set of addressing modes is offered, including:

  - absolute CM memory
  - code segment relative
  - data segment relative
  - S relative
  - Q relative
  - DB relative
  - HP-PA virtual addresses
  - HP-PA real memory addresses

- Display a symbolic procedure stack trace, optionally displaying interleaved Native Mode (NM) and Compatibility Mode (CM) calls. You can also temporarily set the current debug environment back to the environment that existed at any marker in the stack.
- Calculate the value of expressions to determine the correct values of variables at a given point in a program. Values can be custom formatted in several bases.
- Use full screen displays (windows) that allow you to inspect registers, program code, the current stack frame, and the top of stack. Groups of custom user windows can be aimed at important data blocks to dynamically monitor changing values.
- Display on-line help for all commands, predefined functions, and environment variables.
- Create and reference user-defined variables.
- Define powerful, parameterized macros. You can invoke macros as new commands to perform sequences of commands or as functions within expressions that return single values.
- Define aliases for command and macro names.
- Execute commands from a file, record all user input to a logfile, and record all DEBUG output to a listfile.

MPE/iX User Interface

MPE/iX has a user interface that includes a command language, on-line HELP facility, user-defined commands (UDCs), and command files.

The MPE/iX command language is processed by the Command Interpreter. It contains all necessary commands to direct and control the system.

You can enter identical MPE/iX commands during a session or through a job; MPE/iX has no separate control language for jobs. You can also issue MPE/iX commands in a command file.

Actions MPE/iX commands can perform include:

- Initiate and terminate jobs and sessions.
- Run system programs and utilities.
- Compile, link, load, run, and debug programs.
- Create, maintain, and delete files.
- Display file information.
- Display job, session, or device status.
- Transmit messages.
- Establish communication with local and remote computers.
- Control and manage system resources.

If the command interpreter (CI) detects an error in command syntax in interactive mode (during a session), it provides a descriptive error message specifying the erroneous parameter and prompts you to correctly reenter the command. If it detects a syntax error while running, it lists the error on the output device and halts the job.

You can use the command language to create batch files (also called stream jobs) that contain control statements and variables. Execution of the commands in the file can be altered at execution time by using these control statements.

MPE/iX System Performance

MPE/iX provides efficient performance through use of the mapped file technique and concurrent directories described in the following subsections.

Mapped Files

MPE/iX employs the mapped file technique for performing file access. It is an improved version of the disc caching-capability of MPE V/E. File access efficiency is improved when code and data portions of files required for processing reside in memory. Accessing memory is faster than performing physical disc I/O operations. The mapped file technique eliminates file system buffering and optimizes global system memory management.

File mapping is based on MPE/iX demand paged virtual memory, which uses to advantage the large amount of virtual memory on the system. When a file is opened, it is logically mapped into virtual memory. An open file and its contents are referenced by virtual addresses. Each byte of each opened file has a unique virtual address.

File mapping improves I/O performance without imposing additional CPU overhead or sacrificing data integrity and protection. Traditional disc caching schemes for increasing I/O performance impose a CPU overhead penalty. The 900 Series hardware and system architecture allow MPE/iX to perform file mapping without incurring this penalty. System hardware performs the virtual to physical address translations for locating portions of the mapped files, thus eliminating CPU overhead for this function.

If the required pages are not in memory, the MPE/iX Memory Manager fetches them directly from disc and places them in the user's area in memory. This eliminates File System buffering. Pages are "prefetched" to reduce the amount of physical disc I/O. Prefetching means that the page specified for fetching and the group of pages surrounding it are a fetched all at once. This improves efficiency because the processor is likely to require pages that are located near each other. Two benefits of this are:

- Eliminating unnecessary data movement in memory improves system performance.
- Memory space usage is optimized.

MPE/iX file system access intrinsics are built on the mapped file technique. Programs using file access methods supported by MPE file types and intrinsics obtain the benefits of file mapping without requiring changes.

You can directly access mapped files when programming in languages with pointers. For example, you can obtain the advantage of File System naming and data protection for accessing array type structures and developing specialized access methods.

You can write programs that address files through virtual memory, instead of calling File System intrinsics for disc reading and writing. The file interface provides opening and closing of user mapped files with normal naming and security, but with improved `LOAD` and `STORE` speed on file references.

Directory Entries

On operating systems that have the system directory centralized on one disc, access to directory services for files on any disc on the system requires serial access to the system directory. In other words, they must occur one at a time. At peak usage times, this creates a bottleneck due to physical contention for one disc and logical contention for one directory user.

The space for the directory structure is spread across the volume set, not necessarily located on one volume. The locking mechanism allows multiple readers and ensures that the proper locks for specific changes in the directory structure, such as file name insertions and deletions.

MPE/iX uses directory entries that are spread across all members of the system volume set to speed up file access and eliminate the physical or logical serialization imposed by a centralized directory.

With MPE/iX, each disc in a system volume set has a directory of files located on it. Thus, user requests automatically go through the directories and arrive at the disc containing the requested files without going through a centralized directory. Multiple users can simultaneously access a system directory.

On nonsystem volume sets, the directory is restricted to the MASTER volume of the set, so that it is not necessary to mount the entire set at one time.

Native Mode System Components

This section describes the MPE/iX operating system subsystems important for programming only in Native Mode (NM). For programming information on MPE/iX operating in Compatibility Mode (CM), refer to the Migration Series of manuals. Fundamental Operating System (FOS) is the name for the MPE/iX operating system and associated software included in a standard 900 Series HP 3000 installation. The software that automatically comes with MPE/iX is a collection of high-level tools that make the system easier to use. In addition, other high-level tools are optionally available for use with MPE/iX. Collectively, these tools include:

- MPE/iX operating system
- Languages:

  HP C/iX
  COBOL II/XL
  HP Pascal/iX
  HP FORTRAN 77/iX
  HP Business BASIC/V (CM only)
  SPL/V (CM only)
  RPG/V (CM only)
- Forms design and screen handling:

  - forms control
  - VPLUS/V (formatted data entry application)

- Office systems:

  - TDP (text editor and formatting application)
  - HPSLATE
  - HPDESKMANAGER (electronic mail application)
  - HPWORD
  - Editor (text editor application)

- Development aids:

  - System Dictionary/XL
  - Transact/V
  - Inform/V
  - Report/V

- Data Management:

  - ALLBASE/SQL
  - TurboIMAGE/XL
  - Query/V (subsystem for verifying and modifying data)
  - Keyed Sequential Access Management (KSAM)
  - MPE/iX File System
  - SORT-MERGE/XL (utility for ordering records in a file and merging records in sorted files. It can sort any character sequence using any data type.)

  MPE/iX also provides special purpose utilities for system administration tasks.

  For example, the MPE/iX Tape Labeling Facility allows you to place labels on magnetic tapes for identification and protection.

  MPE/iX provides utilities to facilitate migration of applications and databases to the 900 Series systems.

Terminal Keyboard Layouts

Hewlett-Packard terminals contain keys for the entire ASCII character set. These characters can be displayed on the CRT screen and printed on a printer. HP terminals have three modes of operation:

- Local (not connected to the HP 3000 system)
- Remote character mode
- Remote block mode

Non-printable keys include the SHIFT, BACKSPACE, RETURN, and TAB keys. The ESC (escape) and CTRL (control) keys generate special codes for terminal operation. The most frequently used codes are the following key combinations (not case sensitive):

CTRL-Y	stop subsystem activity
CTRL-S	suspend output
CTRL-Q	resume output

Terminals come with various memory amounts. This affects the amount of information you can access at one time. When each line of the screen is filled, the top line scrolls up to make room for a new line at the bottom. This continues until terminal memory is filled. At this point, information will be lost from view when you enter another line because the line scrolled up at the top of the memory buffer is no longer available.

Giving Commands to MPE/iX

There are several ways to command MPE/iX: commands, command files, and user-defined commands (UDCs). These alternatives are described in the subsections below and shown in Figure 1-4.

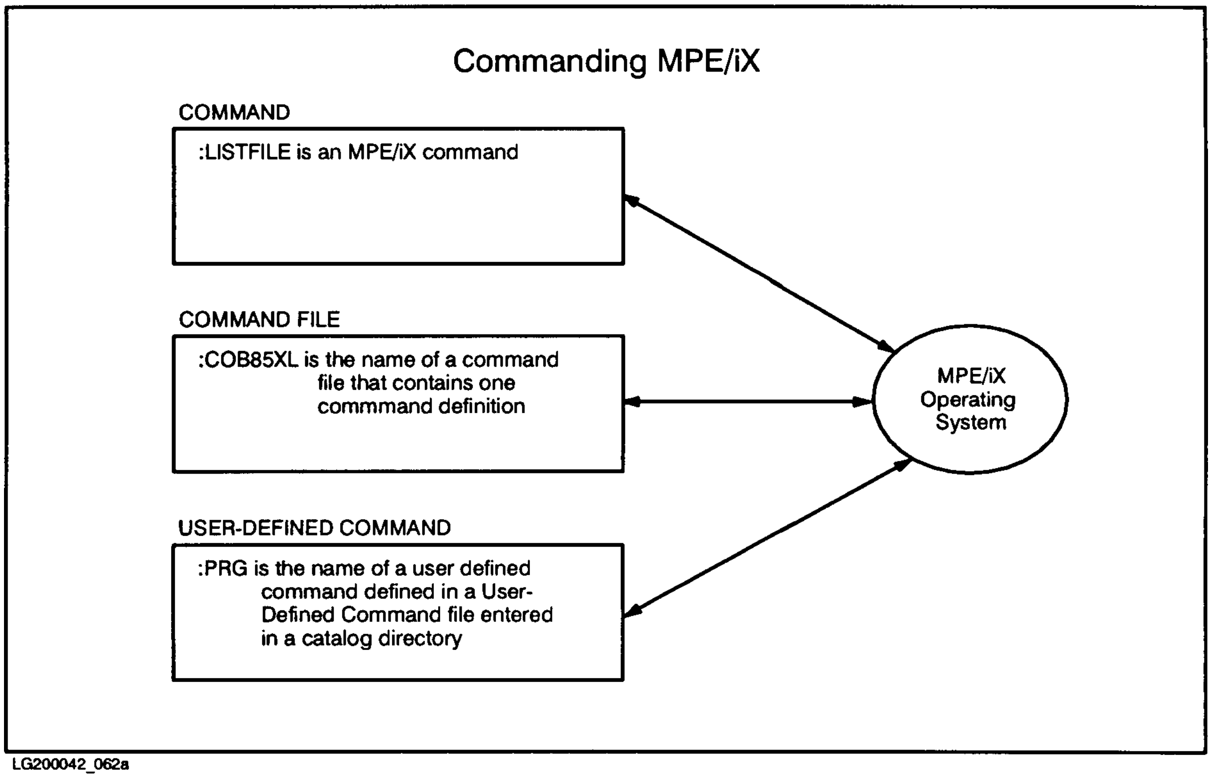

Commanding MPE/iX
COMMAND
:LISTFILE is an MPE/iX command
COMMAND FILE
:COB85XL is the name of a command file that contains one commmand definition
USER-DEFINED COMMAND
:PRG is the name of a user defined command defined in a User-Defined Command file entered in a catalog directory
MPE/iX Operating System
LG200042_062a

Figure 1-4. Commanding MPE/iX

MPE/iX Commands

MPE/iX commands perform many different functions: managing files, compiling programs, executing programs, and so on. Many commands actually invoke subsystems, causing other programs to run. Some commands require that you have capabilities on the system beyond that of the normal user. For example, they may require Account Manager (AM), System Manager (SM), or System Supervisor (OP) capability.

In a session, the command interpreter uses a leading colon (:) as a prompt character to indicate that it is expecting you to enter an MPE/iX command. In jobs, you enter the leading colon before an MPE/iX command to identify it. In both cases, the system disregards blanks between the leading colon and the MPE/iX command.

The list below shows some common MPE/iX commands grouped according to similar function.

```
     :HELLO
:BYE

     :BUILD
:FILE
:LINK
:RUN

     :COPY
:RENAME
:LISTF
:PURGE

     :SHOWJOB
:SHOWME

     :HELP
:REDO

     :TELL
:TELLOP
:SETMSG

     :CCXL
:COB85XL
:PASXL
:FTNXL

     :RESUME
:ABORT

BREAK key
```

For detailed information, refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364).

MPE/iX Command Parameters

Some commands require or can accept parameters. Parameters given in command syntax can have two components:

- Literal information that you must enter exactly as shown in documentation. For example, in the command

  ```
  :SHOWJOB JOB=@J
  ```

  `JOB=` is literal information.

- User-supplied variable information that is specific for the current invocation of the command. For example, in the command above, `@J` is a user-supplied variable.

For detailed information on the conventions for documenting command syntax (including parameters), refer to "Notation Conventions" in the front of this manual.

MPE commands can have positional or keyword parameter lists. The characteristics of a positional parameter list are:

- Location of a parameter in the list is significant.
- Each parameter is separated by a comma (,).
- If a parameter is omitted from the list, a place-holding comma must replace it, unless the parameter is at the end of the list.

For example:

```
        :COBOLII SOURCE,,LISTOUT
```

The characteristics of a keyword parameter list are:

- Location of a parameter in the list is not significant.
- Each parameter is preceded by a semicolon (;).
- It can have a positional subparameter list.

For example,

```
        :FILE OUTFILE;DEV=LP;CCTL
```

is the same as

```
        :FILE OUTFILE;CCTL;DEV=LP
```

Continuing an MPE/iX Command to Another Line

The maximum number of characters in a command is 256. You can continue a command on more than one line by entering an ampersand (&) as the last nonblank character on the line to be continued. Enter the ampersand immediately before or after a delimiter (**, ; . = /** or blank).

For example:

```
        :BUILD MYFILE;DEV=DISC &
        :REC=-80
```

On-line Help in Using an MPE/iX Command

MPE/iX provides an on-line help facility to aid you in using MPE/iX commands. You can easily experiment with it by typing `HELP` at the colon (:) prompt.

Correcting or Modifying an MPE/iX Command

MPE/iX provides a correction facility to correct the previous MPE/iX command or modify it to use again. The `:REDO` command allows you to modify the previously displayed command without having to retype the entire command. It does not cancel any action performed by the previously displayed command.

Referring to Several Files at Once

Several generic characters can be used like wildcards in specifying a file set or volume set. This is especially useful in listing operations, to avoid entering exact names when you are performing the operation on many files with similar names. Generic characters allow you to process an entire set of files in one MPE/iX command, by specifying a string or character that is common to all members of the set. You can use generic characters with some MPE/iX commands.

The generic characters (or portions of names) are described as follows:

@	All strings up to eight characters long, including a null string.
?	A single alphanumeric character.
#	A single numeric character.

Examples of wildcard use are:

`@A@` All filenames (in the logon account and group) that include an `A`. For example, it could include the filenames:

```
AARDVARK
DELTA
BACCUS
```

`@.@.PRR` All filenames in all groups of the `PROG` account. For example, it could include the filenames:

```
CONTENTS.PUB.PROG
CONTENTS.MKT.PROG
CONTENTS.ENG.PROG
```

`ABC?` All filenames (in the logon account and group) starting with `ABC` and ending with any single alphanumeric character. For example, it could include the filenames:

```
ABCX
ABCY
ABCZ
```

`XYZ#@` All filenames (in the logon account and group) starting with `XYZ`, followed by a single numeric character, and possibly ending in other characters. For example, it could include the filenames:

```
XYZ1
XYZ23
XYZ2AAA
```

Command Files and User-defined Command Files

Command files and user-defined commands (UDCs) are files that allow programmers to customize their environment. MPE/iX accepts numbered and unnumbered files composed of commands. The commands can consist of any number of MPE/iX commands. Each command line can be a maximum of 279 characters long. To continue a line, place an ampersand (`&`) at the end of the line, after the last nonblank character. A command line can be continued up to a maximum of ten lines, not exceeding a total of 279 characters. The maximum number of characters on a line is 80.

A Command File is a file that contains a single command definition. It is executed by specifying its file name. A Command File does not have a command name and is not entered in a catalog directory. For example, entering `:COB85XL`, followed by a source file name, executes a command definition that invokes the COBOL II/XL compiler.

A User-Defined Command (UDC) file is a text file that contains one or more command definitions and with a name for each definition. Each definition is a UDC. You can use a UDC to perform several MPE/iX commands in succession by issuing only the name of the UDC. You can also use a UDC to disable an MPE/iX command. You can create the UDC file by using Editor or TDP.

Each UDC file command definition in a UDC has the following components:

- Head: name of the UDC (required) and parameters and defaults (optional).
- Options (this portion is optional):

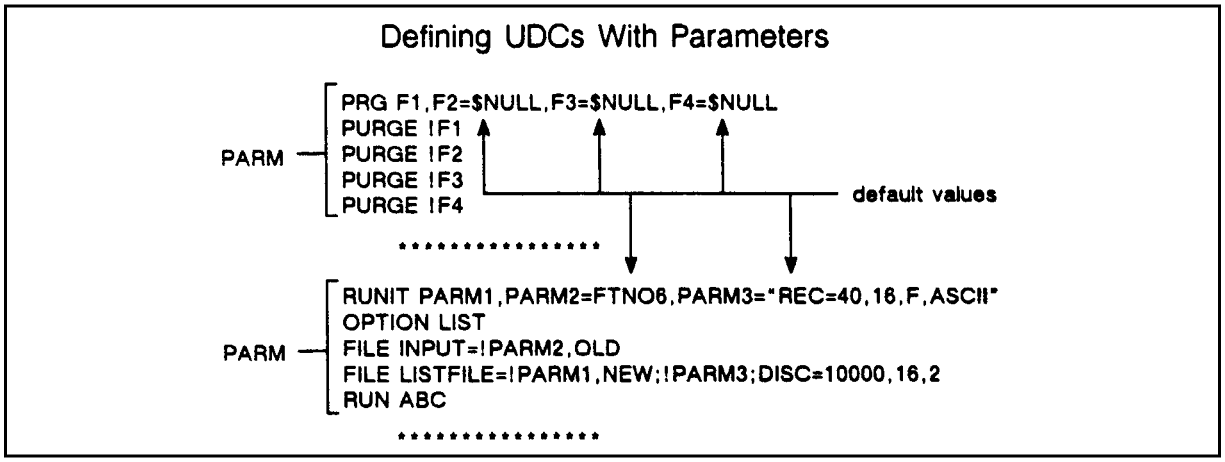
  `LIST`, which lists all commands executable when the UDC is invoked.

  `LOGON`, which immediately executes the UDC at log on.

  `NOBREAK`, which disables the BREAK key for UDC execution.

  `NOHELP`, which disables `:HELP` for UDC execution (normally `:HELP` lists the entire UDC file).
- Body: one or more MPE/iX commands contained in this UDC.
- Separator: one or more asterisks (*) alone on a line, separating command definitions.

Figure 1-5 shows an example of a UDC file containing two UDCs.



```
PRG F1,F2=$NULL,F3=$NULL,F4=$NULL
PURGE !F1
PURGE !F2
PURGE !F3
PURGE !F4
****************
RUNIT PARM1,PARM2=FTN06,PARM3="REC=40,16,F,ASCII"
OPTION LIST
FILE INPUT=!PARM2,OLD
FILE LISTFILE=!PARM1,NEW;!PARM3;DISC=10000,16,2
RUN ABC
****************
```

Figure 1-5. UDC File Example

This UDC file defines two UDCs that accept parameters when executed. The first UDC purges one or more files. The second one runs a program. The following rules apply to parameter specifications:

- Parameters can have any name that starts with an alphabetic character and has no special characters, such as $ or %.
- A parameter without a default value is required to execute the UDC.
- An exclamation point (`!`) indicates that the following string is a parameter. If no exclamation point appears, MPE/iX processes the string as part of the command.
- If the default value specified for a parameter contains a special character other than `$` or `#`, the default must be delimited by double quotation marks (").

- If a UDC calls another UDC, the called UDC must be defined in the UDC catalog (see below) after the calling UDC unless recursion is specifically enabled.

To activate the UDCs in a UDC file, the file must be identified to MPE/iX as a catalog. Use the `:SETCATALOG` command to catalog a UDC file. Each time you execute `:SETCATALOG`, the specified file becomes the only enabled UDC file, unless you specify that more files should be appended. An enabled UDC file is frequently called a UDC catalog.

Executing `:SETCATALOG` without any file names disables all UDC catalogs. You must first disable the UDC catalog to save a new version of a it to the same file name. If you do not want to disable the UDC catalog, you can save a new version of it under a different file name. Executing `:SHOWCATALOG` shows a list of all UDC catalogs (enabled UDC files) and the UDCs within them.

The file named `COMMAND.PUB.SYS` contains a table of UDC users and catalogs. Purging or putting a lockword on this file disables all UDCs.

When you use a UDC, catalogs are searched for the specified command name in the following order of catalogs set at:

- User level (Us)
- Account level (Ac)
- System level (Sy)

The order in which UDC catalogs are searched within a level is determined by the order in which they were specified in the `:SETCATALOG` command. Command definitions are sequentially searched for execution in order of appearance in a UDC catalog.

System resources are required to manage UDCs for each session in which they are enabled. UDCs that are automatically executed at log on cause an increase in the time required to complete the log on. A situation where many users have several UDC catalogs (enabled UDC files) can have a severe negative impact on system performance.

For detailed information on command files and UDCs, refer to *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

Break Mode

When a process is in control, it reads user input and acts according to its own rules and conventions. It is possible to temporarily interrupt the process to go back to the command interpreter (CI) without exiting the process and having to restart it. When this occurs, the CI is executing in break mode.

Activating the BREAK key interrupts the data communication function of the terminal. This can suspend or terminate the process currently executing (for example, a subsystem or user program). The program currently running is interrupted, not the actual MPE/iX command that started execution of the program; this command may have already completed execution. In break mode, you can enter MPE/iX commands and user-defined commands (UDCs). When you press the BREAK key, the colon prompt (:) appears, indicating that your session is in break mode.

The following MPE/iX commands can be interrupted by pressing the BREAK key. This action aborts the output of the command, not its action:

```
:BYE
:HELLO
:HELP
:LISTF
:LISTVS
:REDO
:REMOTE HELLO
:REPORT
:SHOWDEV
:SHOWIN
:SHOWJCW
:SHOWJOB
:SHOWME
:SHOWOUT
:STREAM
```

The following MPE/iX commands can by interrupted by pressing the BREAK key. This action temporarily suspends execution of the command; execution can be resumed by entering `:RESUME`.

```
:CCXL
:CCXLG
:CCXLK
:COB74XL
:COB74XLG
:COB74XLK
:COB85XL
:COB85XLG
:COB85XLK
:EDITOR
:FTNXL
:FTNXLGO
:FTNXLLK
:LINK
:LINKEDIT
:PASXL
:PASXLGO
:PASXLLK
:RESTORE
:RUN
:STORE
```

Echo On/Off

Normally, everything you type on the keyboard is shown on the screen. In other words, the screen echoes your typing. However, in some situations, you may not want to see your typing appear. For example, when you are typing a password, someone looking over your shoulder may see it or observe it on the screen if you leave the terminal. It provides better security if the password does not appear on the screen. You can turn off the echo feature, so your typing does not appear, by entering

```
:SET ECHO=OFF
```

You can turn on the echo feature, so you can see what you type, by entering

```
:SET ECHO=ON
```

Accounting Structure Overview

For programming on the 900 Series HP 3000, it is important that you understand the accounting structure. Its major components are:

- Accounts
- Users
- Groups
- Files

The accounting structure on the HP 3000 family of computers is designed for business and industrial purposes. The ability to account for system use on a department level is a fundamental element in business accounting. HP 3000 systems record on an account basis the amount of CPU time, elapsed time (connect time), and disc space used. An HP 3000 computer must have a system account named `SYS`. This is used to store the information necessary to running the operating system. You can add more accounts for other purposes.

A user name identifies a valid user for an account. A group is a partition in an account for storing files that are logically related.

Logon and Logoff

If you are not familiar with the HP 3000 accounting structure, you can still log on and off the system for simple use while you are learning about the accounting structure. Simple instructions for logging on the system are provided in To log off the system, simply enter the MPE/iX command `:BYE`.

Account

An account on the 900 Series HP 3000 provides a billable entity for accounting purposes. Accounts can be added to or removed from a system, as needed. A system always has a `SYS` account, containing operating system software, subsystem software, and system data. A `SUPPORT` account contains troubleshooting information useful for providing information for proper system support. The `SYS` and `SUPPORT` accounts are part of the system when it is delivered. On an account basis, the system can limit:

- Amount of CPU time
- Elapsed connect time
- Disc space used

An account name can be from one to eight alphanumeric characters long, starting with an alphabetic character. Following is an example of a logon for user John in the account PROG.

```
:HELLO JOHN.PROG
```

As a security provision, you can assign an account password to an account. When an account has a password, MPE/iX prompts you to enter it after you enter the `:HELLO` command. When you enter a password in response to this prompt, it is not echoed on the screen as you type it.

As a short cut, you can enter the account password in the `:HELLO` command instead of waiting for MPE/iX to prompt you for it. To do this, type the account password after the account name, separated by a slash (/). When you enter a password in this way, as part of the log on line, it is echoed on the screen as you type it and may reduce system security because it is visible. For example, if the account PROG has an account password of `XYZ` you can either wait for the password prompt or log on by entering:

```
:HELLO JOHN.PROG/XYZ
```

Figure 1-6 shows an example of the types of accounts a company might have on a system.

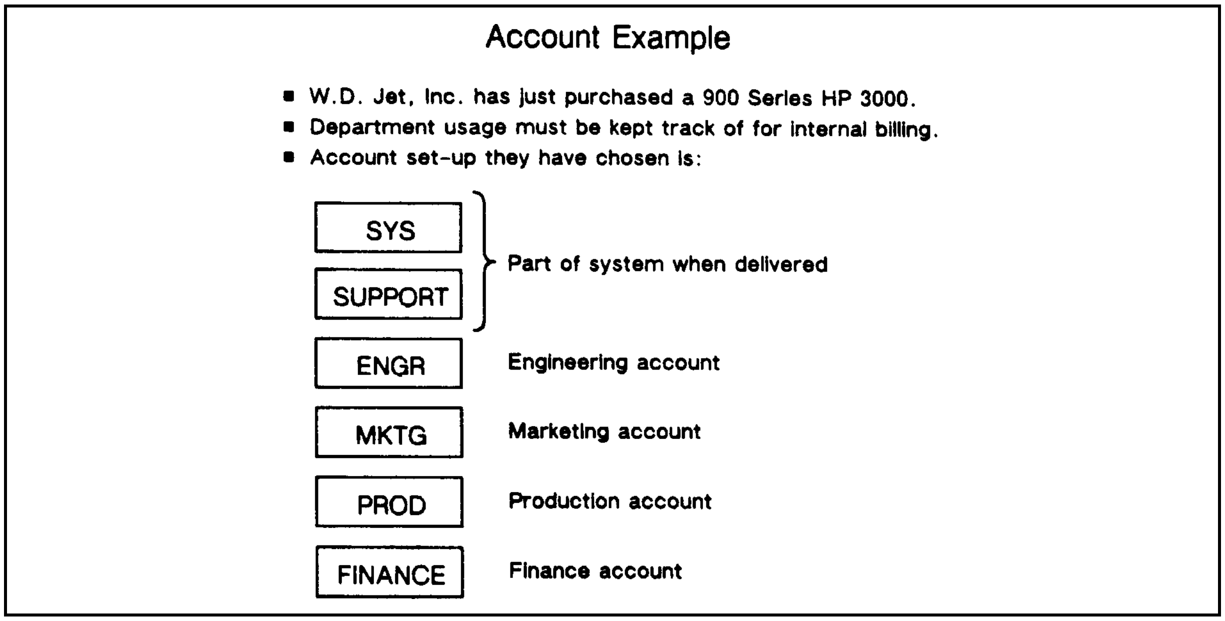

Account Example
W.D. Jet, Inc. has just purchased a 900 Series HP 3000.
Department usage must be kept track of for internal billing.
Account set-up they have chosen is:
SYS
SUPPORT
Part of system when delivered
ENGR
Engineering account
MKTG
Marketing account
PROD
Production account
FINANCE
Finance account

Figure 1-6. Accounts Example

User

Each account can have many users. A valid user name is required to log onto an account. At least one user must be designated for an account in order to log onto it. The `SYS` account automatically comes with the user name `MANAGER`. User names can be added to or removed from an account, as needed. An example of a logon for a user MARY in the FINANCE account follows:

```
:HELLO MARY.FINANCE
```

As a security provision, you can assign a user password to a user. When a user has a password, MPE/iX prompts you to enter it after you enter the :HELLO command and the account password, if one exists. When you enter a password in response to this prompt, it is not echoed on the screen as you type it.

As a short cut, you can enter the user password in the :HELLO command instead of waiting for MPE/iX to prompt you for it. To do this, type the user password after the user name, separated by a slash (/). When you enter a password in this way, as part of the log on line, it is echoed on the screen as you type it and may reduce system security because it is visible. For example, if the user JOHN has a user password of BDATA you can either wait for the password prompt or log on as follows:

```
:HELLO JOHN/BDATA.PROG/XYZ
```

Figure 1-7 shows an example of several users on an account.

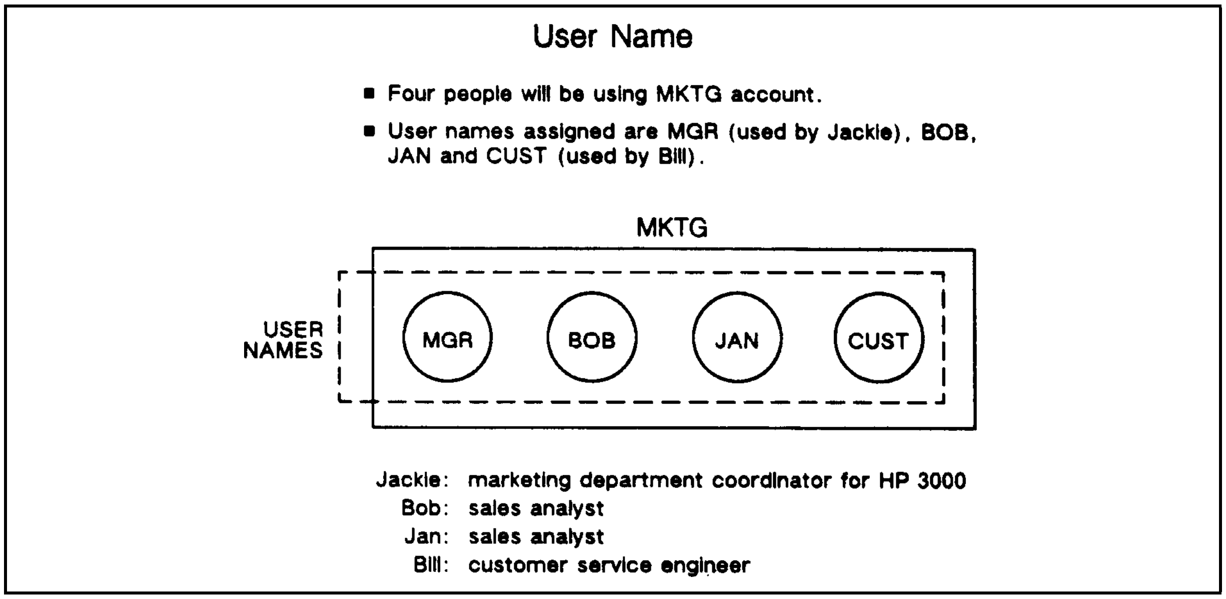

User Name
Four people will be using MKTG account.
User names assigned are MGR (used by Jackie), BOB, JAN and CUST (used by Bill).
MKTG
USER NAMES
MGR
BOB
JAN
CUST
Jackie: marketing department coordinator for HP 3000
Bob: sales analyst
Jan: sales analyst
Bill: customer service engineer

Figure 1-7. User Example

Group

A group in an account allows you to store sets of files that have something in common. You can think of a group as a file folder holding any files you wish to keep together. Groups partition sets of files in an account. You can execute the MPE/iX command :REPORT to see a list of all groups in the logged on account.

When an account is created, it automatically has a group named PUB. Groups can be added to or removed from an account, as needed. As a convenience, a user can be set up to have a home group. If you have a home group, you need not specify the group when you log on. Otherwise, you must specify the group at log on, or you will be logged onto the PUB group, by default.

Following is an example of a logon to the group PROJ1:

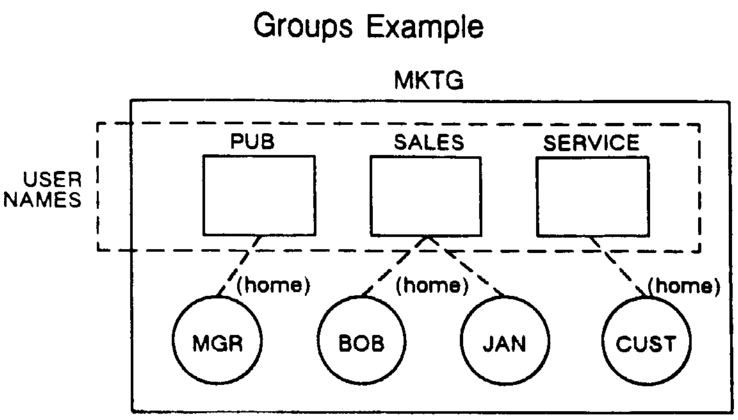
```
:HELLO JOHN.PROG,PROJ1
```

As a security provision, you can assign a user password to a group. When a group has a password, MPE/iX prompts you to enter it after you enter the :HELLO command and the account and user passwords, if they exist. When you enter a password in response to this prompt, it is not echoed on the screen as you type it.

As a short cut, you can enter the group password in the `:HELLO` command instead of waiting for MPE/iX to prompt you for it. To do this, type the group password after the group name, separated by a slash (/). When you enter a password in this way, as part of the log on line, it is echoed on the screen as you type it and may reduce system security because it is visible. For example, if the `PROJ1` group shown above has a group password of `MINE` you can either wait for the password prompt or log on by entering the password as part of your logon:

```
:HELLO JOHN.PROG,PROJ1/MINE
```

The following figure shows an example of several groups and several users on an account.



```
home group logons:    :HELLO MGR.MKTG
                      :HELLO BOB.MKTG
                      :HELLO JAN.MKTG
                      :HELLO CUST.MKTG

User BOB wants to log onto customer service:

                      :HELLO BOB.MKTG, SERVICE
```

Figure 1-8. Groups Example

File Specifications

You must use a standard file reference format to refer to a file. If you are logged into the account and group where the file resides, only the file name and lockword (if it has one) are required. A lockword is an optional, additional security provision that the MPE/iX File System provides for individual files. file level. If a file has a lockword, you must specify the file name and lockword to refer to the file, using the following format:

filename/lockword

where *filename* is a valid file name (refer to Chapter 7) and *lockword* is the lockword associated with the file. For example, you can refer to a file named `STATUS` with the lockword `MY` by specifying:

```
STATUS/MY
```

Referring to a File in a Different Group

To refer to a file in a different group in the account you are logged onto, use the following standard file reference format:

filename/lockword.groupname

where *groupname* is the name of the group where the file resides. For example, you can refer to the file described in the example above, residing in a group named `SEPT`, by specifying:

```
STATUS/MY.SEPT
```

Referring to a File in a Different Account

To refer to a file in a different account than the one you are logged onto, use the following standard file reference format:

filename/lockword.groupname.accountname

where *accountname* is the name of the account where the file resides. For example, you can refer to the file described in the example above, residing in an account named `MANU`, by specifying:

```
STATUS/MY.SEPT.MANU
```

Session and Batch Modes

This section describes how to use sessions and jobs on a 900 Series HP 3000. A detailed description of the steps involved in program development are given in Chapter 3 "Program Development."

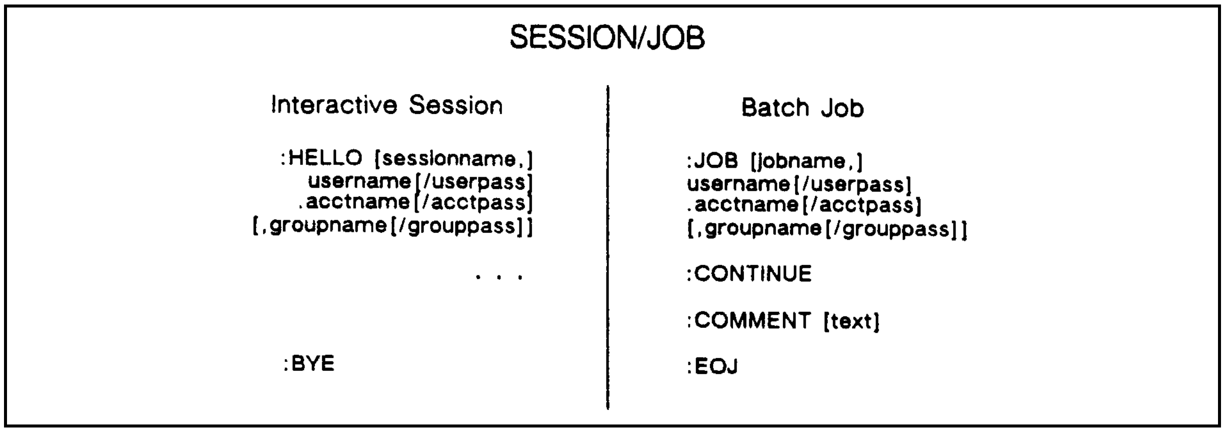
The two ways to perform tasks on a Series 900 HP 3000 system are session mode and batch mode. Session mode is interactive. In other words, you log on to the system, it prompts you with information displayed on the screen to determine what you want to do (or tell you what it has done), you enter information telling it what you want to do, and the system executes your commands. Session mode is dynamic; you can submit commands that can alter the outcome of your task, as you go.

Batch mode requires that you set up a job that you can submit all at once to the system for processing. The job contains all the information necessary for the system to perform your task: a log on, a list of commands to execute, and an end-of-job signal. When you submit a job in batch mode, you cannot alter the outcome of the task. The job will run to completion (assuming it does not end abnormally) without any opportunity for you to change the commands in the job.

You can use the MPE/iX command `:STREAM` to initiate a job, once you have placed all the instructions for the job in a file. This job stream is independent of the session or job that originated the file. You can use it to initiate a job directly from a current session or from a disc file.

Almost all MPE/iX commands execute in either mode. Some give slightly different information depending on the mode in which they are executed, and some are totally interactive and are ignored in batch mode.

Figure 1-9 shows a comparison of the commands that begin and end a session and job.

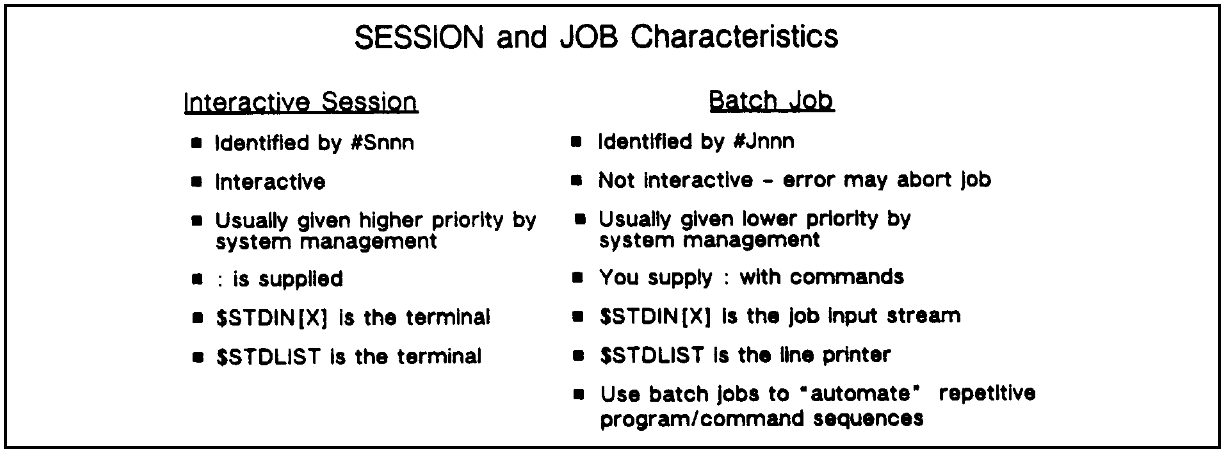


SESSION/JOB

Interactive Session	Batch Job
:HELLO [sessionname,] username[/userpass] .acctname[/acctpass] [,groupname[/grouppass]]	:JOB [jobname,] username[/userpass] .acctname[/acctpass] [,groupname[/grouppass]]
. . .	:CONTINUE
	:COMMENT [text]
:BYE	:EOJ

Figure 1-9. Session and Job Commands

Figure 1-10 shows a comparison of characteristics of a session and a job. You may not yet be familiar with all of the terms used, but you can get a general idea of the similarities and differences and refer to this figure again when you are more familiar with the 900 Series HP 3000.



SESSION and JOB Characteristics

Interactive Session	Batch Job
■ Identified by #Snnn	■ Identified by #Jnnn
■ Interactive	■ Not interactive - error may abort job
■ Usually given higher priority by system management	■ Usually given lower priority by system management
■ : is supplied	■ You supply : with commands
■ $STDIN[X] is the terminal	■ $STDIN[X] is the job input stream
■ $STDLIST is the terminal	■ $STDLIST is the line printer
	■ Use batch jobs to "automate" repetitive program/command sequences

Figure 1-10. Session and Job Characteristics

You can identify sessions and jobs currently running on the system by using the MPE/iX command `:SHOWJOB`. This lists all sessions and jobs, identifies them with a unique number (sessions start with `#S`, jobs start with `#J`), gives their state of execution, input priority, and other pertinent information.

Converting Data Files from MPE V/E to MPE/iX

When converting files from MPE V/E to MPE/iX, you must consider several differences in data storage, including:

- Floating point differences
- Floating point, single precision differences
- Floating point, double precision differences
- Floating point conversion intrinsic
- HP FORTRAN 77/iX native alignment
- HP Pascal/iX allocation alignment of independent variables
- COBOL II/XL native alignment

MPE/iX has many data type differences from MPE V/E, in terms of concepts of data storage, differences in data storage techniques, and implications for the programmer in handling them. For an overview of data conversion, refer to *Introduction to MPE/iX for MPE V Programmers* (30367-90005). For detailed information on MPE/iX data types, refer to *Data Types Conversion Programmer's Guide* (32650-90015). It describes the utilities for converting MPE V/E binary data files to MPE/iX native mode format.

Differences integral to the conversion of MPE V/E data files to MPE/iX are:

- MPE V/E and MPE/iX data variables and data structures are different. MPE V/E has a 16-bit native word length and MPE/iX has a 32-bit word length.
- MPE V/E and MPE/iX floating point formats differ.

MPE/iX is the same as MPE V/E for most data conversions, except for floating-point decimal numbers. An intrinsic called `HPFPCONVERT`, included in MPE/iX, converts floating-point decimal number formats.

Data alignment and real number storage formats differ in MPE/iX from their MPE V/E implementations.

Data Alignment Differences

MPE V/E and MPE/iX each have a different word size; MPE/iX has a 32-bit word size, and MPE V/E has a 16-bit word size. Therefore, applications to be compiled on MPE/iX to run in Native Mode using MPE V/E-compatible data files, may have to select the HP3000_16 compiler directive that is provided with Native Mode compilers. This option causes the compiler to:

- Align data in records on 16-bit boundaries (as in MPE V/E), instead of 32-bit boundaries.
- Selects the MPE V/E representation mode for real numbers.

Many data structures that are aligned on 16-bit boundaries on MPE V/E are aligned on 32-bit boundaries on MPE/iX. On MPE/iX, 32-bit data types are aligned on 32-bit boundaries, by default, to improve performance. Figure 1-11 shows the differences in data alignment between MPE V/E and MPE/iX with an HP FORTRAN 77/iX example.

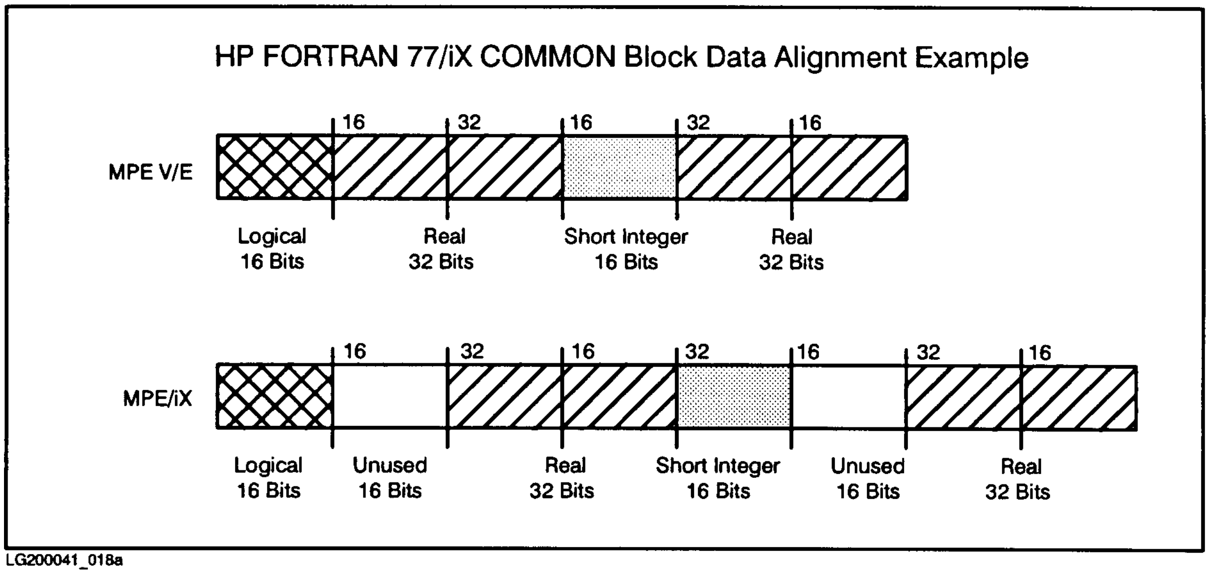


Figure 1-11. HP FORTRAN 77/iX COMMON Block Data Alignment Example

Native alignment for some HP Pascal/iX, COBOL II/XL, and HP FORTRAN 77/iX data types is different than that used by languages running on MPE V/E. For tables comparing alignment by data type, refer to *Introduction to MPE/iX for MPE V Programmers* (30367-90005).

If an application uses both native aligned data files and MPE V/E aligned data files, an alignment directive should be specified in the program record definitions to force MPE/iX or MPE V/E-aligned records on a structure-by-structure basis. MPE/iX compilers offer two directives to specify MPE/iX or MPE V/E alignment, HP3000_32 and HP3000_16. For more information on these directives, refer to the programmer's guide for the appropriate language in the *Language Series*.

Note

The HP3000_16 compiler directive maintains data alignment and format compatibility with MPE V/E and impacts the ability to use Native Mode (NM) data structures. Unless specified otherwise, all data elements are in the mode of the program or as specified by the compiler options in effect.

For example, to maintain Compatibility Mode (CM) data alignment and format and create NM data structures, you should explicitly define the individual structures to be in NM format as HP3000_32 while operating under the HP3000_16 compiler directive. The other alternative is to create a program that will read data in one mode and write it in the other.

Native Mode (NM) words and integers are different from Compatibility Mode (CM) words. Table 1-1 shows the proper conversions. In this manual, NM words are implied, unless a CM prefix is added to a term.

Table 1-1. Word and Integer Conversions

Bits	Native Mode (NM)	Compatibility Mode (CM)
16	1 halfword or shortint	1 CM word or integer
32	1 word or integer	2 CM words, double integers, or double words
64	2 words	4 CM words

Integers in MPE/iX can be 16 or 32 bits long, signed, or unsigned. Signed integers are in twos complement form. Halfword integers (16-bit) are stored in memory at even byte addresses, and word integers (32-bit) at addresses divisible by four.

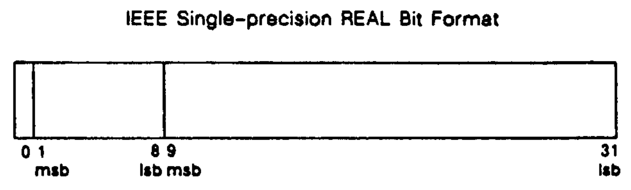
Real numbers in MPE/iX are stored in one of two floating-point formats. The two methods of storing floating-point numbers in the MPE/iX environment are: HP 3000 format, and according to IEEE Task P854. The default MPE/iX Native Mode (NM) real number representation conforms to IEEE Task P854. This standard specifies a storage format different than that used in MPE V/E-based HP 3000 systems. The real number format for a particular application can be forced to the HP 3000 representation by specifying the HP3000_16 compiler directive. Since this compiler directive selects both MPE V/E alignment and real number format, native aligned data must then be aligned on a per record basis.

A single source module can use only one real number format, but the HPFPCONVERT intrinsic converts real numbers between the various formats. Separate (external) procedures may use different formats. The formats have different precisions and ranges for both single-precision and double-precision real numbers.

Because of accuracy differences between IEEE and HP 3000 double real numbers, the least significant digit may be lost in 16-digit real numbers. Since the HP 3000 double precision range is smaller than the IEEE double real range, conversion from a very large or small one in IEEE to one in HP 3000 may cause an overflow or underflow.

For a comparison of the real number representation on MPE V/E and MPE/iX, refer to *Introduction to MPE/iX for MPE V Programmers* (30367-90005).

Figure 1-12 and Figure 1-13 show the internal representation for floating-point numbers on MPE/iX.

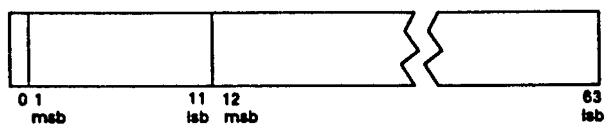


where:

Bit 0	is the sign field
Bits (1:8)	is the exponent field
Bits (9:23)	is the mantissa field
Bit 1	is the most significant bit (msb) in the exponent field
Bit 8	is the least significant bit (lsb) in the exponent field
Bit 9	is the most significant bit (msb) in the mantissa field
Bit 31	is the least significant bit (lsb) in the mantissa field

Figure 1-12. IEEE Single-precision Real Number Format

IEEE Double-precision REAL Bit Format



where:

Bit 0	is the sign field
Bits (1:11)	is the exponent field
Bits (12:52)	is the mantissa field
Bit 1	is the most significant bit (msb) in the exponent field
Bit 11	is the least significant bit (lsb) in the exponent field
Bit 12	is the most significant bit (msb) in the mantissa field
Bit 63	is the least significant bit (lsb) in the mantissa field

Figure 1-13. IEEE Double-precision Real Number Format

Conversion from HP 3000 format to IEEE format for a single-precision real number can present a range problem, because the IEEE range is smaller. Thus, overflow can occur in performing either of the following conversions:

- From an HP 3000 single-precision real number to an IEEE single-precision real number.
- From an IEEE double-precision real number to an IEEE single-precision real number.

You may have to develop new error handling code to prevent overflow.

The mantissa of an HP 3000 double-precision real number provides enough bits for 16 digits of accuracy. The mantissa of an IEEE double-precision real number provides for 15.9 digits of accuracy. Thus, converting double-precision real numbers from HP 3000 to IEEE format can incur an extremely small loss of numeric precision. However, if the requirements of an application depend on the ASCII representation of floating-point results, the effect of this accuracy difference can be important.

For example, if a program assumes 16-digit accuracy and requests 16 digits for formatting output, with trailing zero suppression, the number 64.4 appears as 64.4 when the system is running in Compatibility Mode (CM) and 64.40000000000001 when the system is running in Native Mode (NM).

Rounding is frequently necessary when formatting output. In HP 3000 format, a number equidistant from two adjacent integers rounds to the integer of greater magnitude. For example, 1.5 rounds to 2, and 2.5 rounds to 3. In IEEE format, a number equidistant from two adjacent integers rounds to the integer that has a least significant bit of zero (in other words, the even integer). For example, 1.5 rounds to 2, and 2.5 also rounds to 2.

Converting Files

General procedures for converting data files from MPE V/E to MPE/iX are described below. For detailed information on conversion, refer to *Introduction to MPE/iX for MPE V Programmers* (30367-90005) or the appropriate language manual in the *Migration Series*.

The procedure for converting HP FORTRAN 77 binary files from MPE V/E to MPE/iX format is as follows:

1. Read data from file in a subroutine with `$HP3000_16 ON`.
2. Pass the data to a subroutine that has `$HP3000_16 OFF`.

3. Call the intrinsic `HPFPCONVERT` to convert `REALs` from MPE V/E to MPE/iX floating point format.
4. Write the data out to a new file.

Note A subroutine that has `$HP3000_16 ON` cannot call the `HPFPCONVERT` intrinsic.

Pascal on MPE V/E and MPE/iX has the following incompatibilities due to data alignment:

- MPE V/E and IEEE floating point format
- Data alignment of simple variables and record elements
- String format
- Pointers

COBOL II on MPE V/E and MPE/iX has incompatible indexed and synchronized data items.

Data Communications

Hewlett-Packard's networked, data communications and data management products are called HP AdvanceNet. AdvanceNet provides network services (NS) software products, including interactive and programmatic services. NS enables Hewlett-Packard and multivendor computer systems to communicate with each other and share resources. For detailed information, refer to *NS3000/XL User/Programmer Reference Manual* (36920-90001).

Network File Transfer (NFT)

Network File Transfer (NFT) is the network service that copies disc files from one computer system in a network to another. NFT can transfer a file between any two systems in a local area network. NFT can transfer files between two systems remote from your own or perform local transfers on a single HP 3000. You can use NFT interactively or programmatically.

Remote Process Management (RPM)

Remote Process Management (RPM) provides intrinsics that allow a process to create and kill other processes (that is, initiate and terminate their execution). A created process may or may not be dependent on the creator. If it is independent, it can continue to execute after the creator has expired. RPM permits a process to create a process and send information to it in the same intrinsic call. You can use RPM in conjunction with Network Interprocess Communication (NetIPC) to manage distributed applications. For detailed information on NetIPC, refer to *NetIPC 3000/XL Programmers Reference Manual* (5958-8600).

Local Area Network (LAN)

NS 3000/XL is Hewlett-Packard's local area network (LAN) software services for linking multivendor computer equipment, including MPE/iX based HP 3000 processors. These network services (NS) run in conjunction with either of the HP AdvanceNet link products ThinLAN3000/XL Link and StarLAN/3000 Link. The LAN link supports the NS product and consists of both hardware and software components. NS3000/XL software services and one of these link products combine to form a high-speed, shared-access, IEEE 802.3 LAN.

Remote File Access

The Remote File Access service (RFA) allows access to remote files and devices. Using RFA, you can create, open, read, write, close, and perform other manipulations on a file residing on a remote HP 3000 system. Since a file can be a peripheral device, you can, for example, read from a tape mounted on a remote system or print local data on a remote printer.

The RFA uses the same MPE/iX File System intrinsics used on a local system. They are sent to the remote environment and executed there. A local program can call them explicitly or use the I/O procedures specific to the language in which the program is written.

You can interactively access a remote file or device if you have previously issued a `:FILE` command that specifies the remote location of the file. You cannot directly indicate the location in the MPE/iX or subsystem command that accesses the file.

You can programmatically access a remote file from a local application program once you have established an environment on the remote node by doing one of the following:

- Calling standard MPE/iX File System intrinsics.
- Using I/O procedures specific to the language in which the program is written. If you have issued a `:FILE` command specifying a formal file designator for a remote file or device, then an `HPFOPEN` or `FOPEN` call in a local program can use this formal file designator in the *formaldesignator* parameter.

Remote Terminal Access

You can use the Virtual Terminal (VT) and Remote File Access (RFA) services to access remote terminals. Use a `:FILE` command or an `HPFOPEN` or `FOPEN` call to indicate that the file you wish to access is actually a remote terminal. The remote terminal functions as a non-session I/O device.

Remote Data Base Access

TurboIMAGE/XL is a Hewlett-Packard database management system. You can use TurboIMAGE/XL intrinsics and utilities to access a TurboIMAGE/XL database residing on a remote HP 3000. TurboIMAGE/XL intrinsics are sent to the remote node and executed in the remote environment. The database should reside on an HP 3000, since other TurboIMAGE products are not fully compatible with TurboIMAGE/XL. The database must be located entirely on a single node.

You can obtain the information required to open a remote TurboIMAGE/XL database in a program in three ways:

- Identify the database as a remote file in a previously executed `:FILE` command.
- Use the `COMMAND` intrinsic to include `:FILE` information in a program.
- Create a database-access file to supply `:FILE`, `:DSLINE`, and `:REMOTE HELLO` commands.

2

Utilities and Tools

This section describes the MPE/iX user interface to show how user programs can programmatically access MPE/iX features and provides an introduction to utilities and tools. The information is presented as an overview on a conceptual level. For detailed information on programmatic access, refer to *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

Programmatic access to MPE/iX allows you to use command interpreter (CI) features in a program without having to enter break mode. With the available techniques, you can develop sophisticated, automated subsystems to run under MPE/iX. The techniques include:

- Using MPE/iX programmatic interfaces, command files, and user-defined commands (UDCs).
- Performing variable dereferencing, concatenation, string substitution, recursive dereferencing, and expression evaluation.
- Using Job Control Words (JCWs) and variables.
- Using the MPE/iX Help facility for commands, command files, and UDCs.

Programmatic Access to the Command Interpreter

Three intrinsics, `COMMAND`, `HPCICOMMAND`, and `MYCOMMAND`, allow programmatic use of the MPE/iX command interpreter (CI). They allow a program other than the CI to:

- Perform system services (for example, building and purging files).
- Use the entire MPE/iX CI command set, command files, and UDCs.
- Parse MPE/iX commands.

Some interactive subsystems can programmatically perform these functions without first having to exit their specific subsystem environment (that is, enter break mode).

The intrinsics for programmatic use of the CI are:

- `COMMAND`, which provides access to the MPE V/E subset of CI commands.
- `HPCICOMMAND`, which provides programmatic access to most of the MPE/iX CI command set, command files, and UDCs.
- `MYCOMMAND`, which provides a programmatic ability to parse a line and return one or more of its parameters.

Command files and user-defined commands (UDCs) are files that allow programmers to develop their own environment. These are discussed in Chapter 1. For detailed information on command files and UDCs, refer to *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

Concatenating Strings and String Substitution

You can concatenate a string with a prefix or suffix. String substitution in a Command File or a UDC parameter takes precedence over substitution of Command Interpreter-level variables. For example,

```
       :SETVAR x 1
:WHILE  x < 10 DO
     :PURGE ACCT!x
     :SETVAR x x+1
:ENDWHILE
:DELETEVAR x
```

purges files ACCT1, ACCT2, ... , ACCT9.

Expression Evaluation

You can specify an expression in a Command File or a UDC parameter and an appropriate value will be assigned to it.

Using Job Control Words (JCWs)

A Job Control Word (JCW) is one type of predefined variable. On MPE/iX, JCWs are a subset of session-level variables, restricted by values and naming conventions. MPE/iX stores JCWs as integers in the session-level variable table. You can manipulate JCWs with

- JCW commands and intrinsics
- Variable commands and intrinsics
- IF command

The commands SETJCW and SHOWJCW correspond to the commands SETVAR and SHOWVAR used on standard variables. DELETEVAR is used to delete JCWs. The intrinsics GETJCW and PUTJCW correspond to the intrinsics HPCIGETVAR and HPCIPUTVAR used on CI variables. MPE/iX distinguishes between the table entries created by the SETJCW command (or the PUTJCW intrinsic) and those created by the SETVAR command (or the HPCIPUTVAR intrinsic).

JCW commands and intrinsics usually function like the corresponding variable commands and intrinsics, with a few exceptions, described below.

Job Control Word Name and Type

A JCW name can be a maximum of 255 characters long. The first character must be alphabetic. Other characters can be alphabetic or numeric. A JCW name cannot contain the underscore (_) or the wildcard parameters (#, ?, [, and]). A JCW must be of type integer, with a value in the range 0 to 65535. This is the only difference between JCW names and variable names. For detailed information on naming and dereferencing variables, refer to *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

Changing the Value of a JCW

Once created, you can change the value of a JCW by using variable or JCW commands and intrinsics. If you assign a value to a JCW that is outside the valid JCW range, then MPE/iX reclassifies it as a standard variable when you assign such a string value to it. You can no longer use JCW commands and intrinsics on it; you must use variable commands and intrinsics to display, change, or delete it. You can force reclassification of a JCW by assigning it a large numeric value. For example,

```
SETJCW X 99
```

defines a variable named `X` as a JCW with value 99.

The statement:

```
SETVAR X 400
```

gives `X` the new value of 400, but `X` remains a JCW. However, when you assign a string value `ABC` to `X` with the statement:

```
SETVAR X "ABC"
```

then `X` becomes a variable and not a JCW. MPE/iX informs you of the change with the message:

```
JCW HAS BEEN RE-CLASSIFIED AS A STANDARD VARIABLE
```

JCW and CIERROR

Two special session-level JCWs named `JCW` and `CIERROR` are also system JCWs. Therefore, they cannot be reclassified as session-level variables or deleted. You can change their values with the `SETVAR` and `HPCIPUTVAR` commands, but only to assign a new value in the legal range for JCWs. If you attempt to assign a value outside the range, MPE/iX issues the message:

```
THE VALUE MUST BE AN INTEGER IN THE RANGE 0...65535
```

and the initial value remains unchanged.

Reserved-word Prefixes

JCWs with reserved-word prefixes and numeric values are:

```
SYSTEM     49152
FATAL      32768
WARNING    16384
OK         0
```

Help Facility

The MPE/iX Help facility is available for commands, User-Defined Commands (UDCs), command files, and program files. Command files and UDCs have two special options for the Help facility: `HELP` and `NOHELP`. `NOHELP` disables the Help facility to provide increased security for UDCs and command files. When `NOHELP` is active, you can execute a UDC or Command File, but cannot display its contents. `HELP` is the default option. For detailed information on this topic, refer to *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

Toolset/XL

The Toolset/XL Program Development System is a product that provides a uniform programming environment for use with COBOL II/XL, HP FORTRAN 77/iX, and HP Pascal/iX on the 900 Series HP 3000. It combines an integrated set of programming tools to facilitate all phases of program development, from source code creation and modification to program compilation, execution, and testing for COBOL II/XL and HP Pascal/iX. For HP FORTRAN 77/iX, it provides programming tools for program testing on the symbolic debugging level.

The major features of Toolset/XL are:

- User interface provides easy access to all programming tools.
- Workspace File Manager allows you to generate versions of source files requiring a minimum of disc utilization. It stores source files using data compression to eliminate unnecessary trailing blanks.
- Full screen editor allows you to enter and modify source code on the screen.
- Program Translation Management simplifies program compilation and execution. The latest compile listing is saved on-line, compile time errors are located in the compile listing by using a “point and push” technique, and the source file can be immediately edited by accessing the full screen editor.
- Symbolic Debug significantly reduces time required to locate run-time errors. You can symbolically reference variables and locations using names in the program. You can set and clear program breakpoints with optional frequency and proceed counts, edit source file at a breakpoint, display values and move new values to any data item, trace and retrace program flow, and trace changes in data item values between paragraphs.
- COBOL II/XL COPY Library Editing and Management provides a reusable code feature.
- COBOL II/XL and HP Pascal/iX Data Definition Extractor allow you to extract definitions from Dictionary/V and use them to generate data declarations.

In addition to the features described above, Toolset/XL provides an on-line help facility, a recovery feature for system failure, and the ability to directly access many MPE/iX commands.

Useful Tools

Several useful programming tools are available to facilitate tasks common to system administrators, programmers, and general users. You can reduce the tedious aspects of many repetitious tasks by using Hewlett-Packard tools to expedite forms design, screen handling, report generation, and system dictionaries, as described below.

Forms Design and Screen Handling Tools

VPLUS/V is a comprehensive software system that implements and controls source data entry. It also provides an interface between a terminal and any transaction processing program.

Source Data Entry

As a source data entry system, VPLUS/V provides easy forms design with data editing and validation built into the forms. It also provides a ready-to-use data entry program called ENTRY that you can use to enter data without any programming effort. ENTRY allows you to browse the entered data and modify it as it is entered.

Thus, source data entry through VPLUS/V can be done without programming by simply using ENTRY. However, if you need additional or different capabilities, you can write your own application incorporating VPLUS/V intrinsics.

Transaction Processing

As an interface to transaction processing applications, VPLUS/V provides a set of intrinsics that allows you to control forms and data on a terminal from an application program. These intrinsics are available to programs written in any of the supported programming languages.

VPLUS/V also provides a reformatting capability. You can enter specifications to control how entered data is to be reformatted and then run a program to actually reformat the data.

VPLUS/V intrinsics and the reformatting capability, either singly or in combination, provide a "front end" to existing transaction processing applications. Thus, VPLUS/V allows you to concentrate on processing problems rather than on editing data or controlling the terminal.

VPLUS/V Features

The main features of the VPLUS/V system are:

- Forms design program called FORMSPEC that allows quick and easy forms design by using menus at a terminal.
- Batch mode management of forms files (through FORMSPEC) that allows a forms file to be updated, compiled, and listed without tying up a terminal.
- Advanced forms design (through FORMSPEC) that edits, formats, moves, and computes data when the form is executed. It uses the user's native language for alphabetic information and the local customs for numeric and date information.
- Ready-to-run data entry program called ENTRY that provides immediate data entry and modification with no programming effort.
- Flexible data reformatting design program called REFSPEC that specifies reformatting of entered data.

- Batch program called REFORMAT that reformats the data according to the REFSPEC formatting specifications and writes it to a file for use by an application.
- Set of intrinsics that provides a powerful programming language interface to terminals, using FORMSPEC definitions, from applications written in any of the supported programming languages.

Report Generation Tools

Several useful report generation tools are available. These are described in the subsections below.

Business Report Writer/XL

Business Report Writer/XL facilitates report development and execution. It is a menu-driven report writing system designed to streamline the development and maintenance of large production reports.

Its capabilities include on-line report formatting, rotational views of data, multipass reporting, advanced calculations, and performance tuning. It provides access to TurboIMAGE/XL, databases, MPE files, and KSAM files. Business Report Writer/XL, together with System Dictionary/XL or Dictionary/V transparently resolve data definitions, structures, and access paths.

Business Report Writer/XL execution programs are part of the MPE XL Fundamental Operating System (FOS). Reports developed on Business Report Writer/XL can be run on other HP 3000 systems without access to Business Report Writer/XL or a dictionary. By using intrinsics with user-specified parameter values, you can run reports from applications written in other languages. Business Report Writer/XL contains a conversion utility to automatically translate Report/V programs into Business Report Writer/XL. For detailed information, refer to *Business Report Writer/XL Reference Manual.*

Report/V

Report/V is a non-procedural report writing language operative only when MPE/iX is in Compatibility Mode. It includes a high-level language that creates reports, a compiler, and a processor to execute the reports. It provides access to TurboIMAGE/XL databases, MPE files, and KSAM files. More advanced reporting capabilities are available in Native Mode through Business Report Writer/XL, which can automatically translate Report/V programs. For detailed information, refer to *Report V Reference Manual* (32245-90001).

HP Visor

HP Visor is a terminal-based interface to HP SQL/V and ALLBASE/XL (HP SQL) databases. It allows end users or database administrators to perform queries and generate their own reports, without involving a programmer. It provides facilities for programmers to use in preconfiguring complex tasks or to improve productivity when working with HP SQL databases.

HP Visor has a forms-based interface that allows the user to mark boxes and press function keys to step through tasks such as queries or custom report generation. It also provides additional capabilities for experienced users to directly enter HP Visor or SQL commands.

Three modules are integrated with the HP Visor interface.

- EZAccess, which enables new or infrequent users of the database to generate queries.
- SQLAccess, which enables users familiar with HP SQL to use a screen-based editor to formulate SQL queries and perform many other database operations.
- EZReport, which formats and displays query results from EZAccess or SQLAccess and allows users to customize report formats and calculate additional statistcs using the data retrieved by the query.

System Dictionary/XL

System Dictionary/XL provides programmers and system administrators with a single source for documenting all aspects of the system, including:

- Data definitions.
- Descriptions of databases, application systems, accounting structures.
- Configuration information.

It ensures that the information is entered in a standard format, stored in an organized structure, and easily available to users and programs. This central source of information aids in developing and maintaining applications and effectively managing system resources. System Dictionary/XL has a programmatic interface for easy integration with other software. You can customize it to meet your needs. Major System Dictionary/XL features are:

- Entity-relationship model provides a documentation structure closely matching the user's conceptual model.
- Extensibility allows customized documentation structure.
- Programmatic access for automatic access and update.
- Domains eliminate name conflicts.
- Separate test, production, and archival versions.
- Uses synonyms and aliases to provide alternative names in the dictionary from those used by external systems.
- Interactive or batch mode interface.
- Accepts macros and include files.
- Reports on dictionary contents.
- Input, output, and all dictionary names can be converted to local languages.
- Security provides restricted access to domains and dictionary objects and defines user capabilities.
- Automates conversion from Dictionary/V to System Dictionary.
- Creates IMAGE schemas and root files from dictionary definitions and produces dictionary definitions from an IMAGE root file.
- Loads information about VPLUS/V forms files into the dictionary.
- Generates definitions for HP COBOL II/XL COPY libraries from System Dictionary definitions (this is part of a separate product called COBOL II/XL Definition Extractor).

A data dictionary is not an alternative to a database management system (DBMS). A dictionary manages information about data on the system, while a DBMS manages the data itself. A DBMS schema may contain some information on the format and usage of the data. However, dictionary information is more extensive and easily usable.

In addition to documenting IMAGE databases, System Dictionary can describe MPE files, KSAM files, VPLUS/V forms files, programs, network device configurations, and other system components. It can also document the relationships among components.

For detailed information, refer to one of the following manuals in the Tools Series:

- *HP System Dictionary/XL COBOL Definition Extractor Reference Manual* (32256-90001).
- *HP System Dictionary/XL General Reference Manual (Volumes 1 and 2)* (32256-90004 and 32256-90005).
- *HP System Dictionary/XL Intrinsics Reference Manual* (32256-90002).
- *HP System Dictionary/XL SDMAIN Reference Manual* (32256-90001).
- *HP System Dictionary/XL Utilities Reference Manual* (32256-90003).

Editor

The Editor is an HP 3000 subsystem that runs on MPE operating systems. It is a line-oriented text editor used to create and manipulate ASCII files. Files can be source programs, job streams, or text material.

You enter commands and lines of text through an input file. The Editor sends messages and prompts in an interactive session by writing to an output file. You issue Editor commands that operate on an Editor work file, which is a temporary file especially created for this purpose. Each operation performed in the Editor manipulates the work file. A permanent text file is created as the result of saving the work file. Until you keep the contents of the work file, no permanent text file exists. If you are updating an existing text file, it remains unchanged while you make the changes on the temporary work file copy. The original permanent text file is overwritten only when you save the work file.

The work file is created as a temporary file and is deleted when you make a normal, orderly exit from the Editor. If your edit session ends abnormally, the temporary work file is saved as a specially named file and is still available. This is called a "K" file; the file name has the form

```
Kdddhhmm
```

where `ddd` is the Julian day, `hh` is the hour, and `mm` is the minute at which the session abnormally ended.

You can use the Editor command `VERIFY` to determine all aspects of the Editor's operating environment, including the location of the work file pointer. For example, you can determine the increment for line numbers, page margins, record length, total length of work file, format, tabs, and other attributes of the environment. Editor is described in the *EDIT/3000 Reference Manual* (03000-90012).

SORT-MERGE/XL

SORT-MERGE/XL is an MPE/iX subsystem that allows you to sort one or more files or merge several sorted files to form one file in a specified sequence. SORT-MERGE/XL changes the order of the records from the input file according to your specifications and writes them to the output file.

You can use SORT-MERGE/XL interactively or programmatically. Use the `SORT-MERGE/XL` intrinsics for programmatic use. You must specify the input and output files and the sorting (or merging) keys. The collating sequence defaults to ASCII unless you specify otherwise. For detailed information on SORT-MERGE/XL, refer to *SORT-MERGE/XL Programmer's Guide* (32650-90080).

Key

A key is the section of the record that SORT-MERGE/XL uses to determine the order in which input records are to be rearranged for output. It is a record field you specify by stating the position of the first byte and the number of bytes in the field. The key applies to the same portion of a record for each record in a file. The data format for that portion must be of the same type in all records.

You can specify multiple keys. The first one you enter becomes the major key. SORT-MERGE/XL uses the major key to rearrange the records. If the content of two records is the same in a key field, SORT-MERGE/XL uses the content of the next specified key to determine which is written to the output file first. If the content of all the key fields for two records is identical, then SORT-MERGE/XL preserves the order found in the input file when it writes to the output file.

Ordering Sequence

SORT-MERGE/XL arranges records in the output file according to an ordering sequence based on the value of data in the keys. The individual bytes in the key definitions determine these values, based on their positions in a collating sequence.

Collating Sequence

The collating sequence can be ASCII, EBCDIC, a native language, or user-defined. ASCII and EBCDIC are the basic collating sequences. Native language collating sequences apply to keys of type `CHARACTER`. You can specify the rearrangement to be in ascending or descending order, based on the appropriate collating sequence.

DISCFREE

DISCFREE is a utility that reduces the amount of fragmentation on a disc. Fragmentation is a term used to describe the extent to which files are physically divided up on a disc. As disc space becomes fragmented, smaller and smaller amounts of contiguous space are available, and files must be broken into smaller and smaller pieces to fit. Fragmentation has a negative effect on system performance and is highly undesirable.

You can use DISCFREE to:

- Determine free space on the disc.
- Perform a `START NORECOVERY` to correct fragmentation.
- Obtain a detailed format of free space (HISTOGRAM) or free space allocation (ALLOCATION) on discs.

For more information on using DISCFREE, refer to *Volume Management* (32650-90045).

FCOPY/XL

FCOPY/XL is a utility program on MPE/iX that allows you to copy data from one file to another. You can copy the content of an entire file or a selected portion of a file from any supported input device to any supported output device. Supported input devices are disc, magnetic tape, terminal, and tape cartridge. Supported output devices are disc, magnetic tape, terminal, tape cartridge, and line printer.

FCOPY/XL's basic functions include:

- Making multiple copies of files
- Making account-independent magnetic tape copies of disc files
- Transferring programs or data from one medium to another (for example, from magnetic tape to disc)

FCOPY/XL has many features that expedite file manipulation. These include:

- FCOPY/XL code translating function, which converts data from one computer code system to another as part of a copying operation. FCOPY/XL can translate files between ASCII and EBCDIC or BCDIC formats. You can use this feature to prepare magnetic tapes to be used at different computer installations or to incorporate magnetic tapes from other installations into your 900 Series HP 3000 system. FCOPY/XL can translate between uppercase and lowercase alphabetic characters. This is useful for a peripheral that has only an uppercase character set.
- FCOPY/XL subset function, which allows you to perform operations using a specified portion of a file. You can designate the file subset as a range of contiguous records or as all records containing a specified data item in specified columns. This is useful for separating a file into several separate files or copying or displaying only specific records of a file.
- FCOPY/XL skip end-of-file function, which allows you to position magnetic tape to the exact file where you want to read or write.

- FCOPY/XL display functions, which show numeric codes and corresponding character symbols, thus allowing you to examine the contents of files at a terminal or on a printer list.

You can also operate on multiple files and multiple tapes. You can interactively invoke FCOPY/XL. For detailed information on using FCOPY/XL and syntax, refer to *FCOPY Reference Manual* (03000-90064).

3

Program Development

The required elements for running a program are:

- Data space (for input, output, and computations)
- Instructions (machine readable code and constants)
- System routines (for example, input and output)

Program development is a term for taking a program design, on paper, to the point where it is machine readable and functions reliably.

Program components are data and code. When a program is running, instructions and data are fetched from main memory to the CPU; data may be stored back into main memory for later use. Code and data must be in main memory when required for execution. CPU registers keep track of the location of such information as:

- Next instruction to execute
- Program status
- Data calculations

The major steps for developing a program are:

1. Writing: design the program and enter the source code in a text file.
2. Compiling: translate source code to machine readable instructions.
3. Linking: bind all resources necessary for the program's code to run the program.
4. Run: execute the program.

These steps are described in detail in the subsections below. Figure 3-1 shows a summary of these steps.

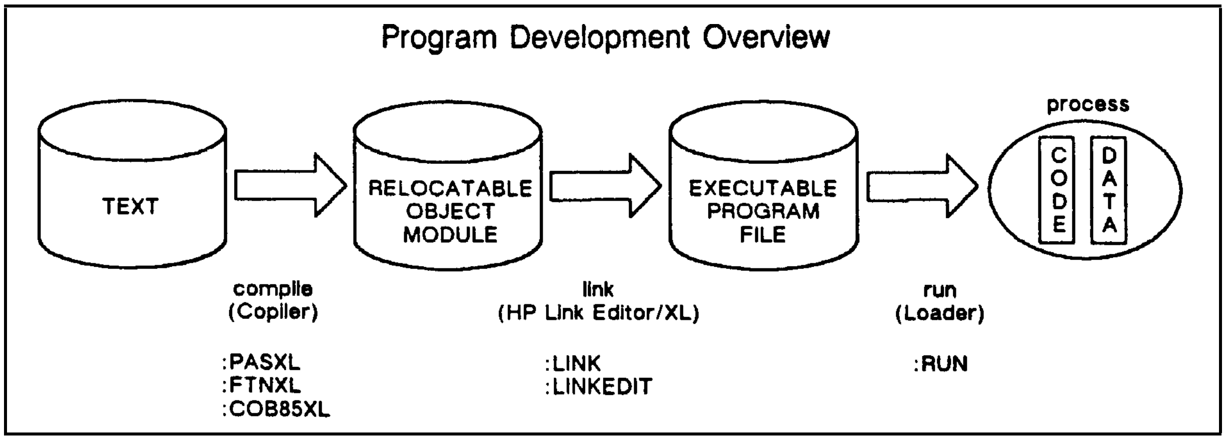


Figure 3-1. MPE/iX Program Development

Writing a Program

You can write a program with any text editor or word processor. MPE/iX provides two facilities for entering text that you can use to create a source file for a program: Editor and Toolset/XL. For an overview of Editor and Toolset/XL, refer to Chapter 2, "Utilities and Tools." Since these facilities can be used for writing documents, memos, and programs, the output is called text files. Although the source code for a program is a text file, this guide refers to it as a source file for program development purposes.

How to Use Intrinsics

Many programs use low-level, system supplied procedures or subroutines to handle recurring tasks. On MPE/iX, these are performed through a set of procedures called intrinsics, which are an integral part of the operating system. Intrinsics are always available to any process on the system and allow a program to gain access to system services. Tasks that intrinsics provide include:

- Accessing and alteration of files (for example, writing to a file)
- Requesting of utility functions (for example, perform ASCII/binary number conversion)
- Access to system resources (for example, obtain system timer information)

You can manipulate and manage processes and system resources by means of intrinsics, provided you have the required execution privileges. Many intrinsics return values to the caller. Most do this through parameters; some, through functional returns. Most intrinsics are coded in HP Pascal/iX (one of the systems programming languages for the 900 Series HP 3000) and are defined by a procedure declaration consisting of:

- Procedure header, containing the procedure name and type, procedure definitions, and other information about the procedure.
- Procedure body, containing executable statements and declarations local to this procedure.

Intrinsics work like user-written procedures, except that the details of performing the task are invisible to you.

Note Most intrinsics are callable from any language supported on MPE/iX.

The MPE/iX intrinsic mechanism provides flexible and convenient access to intrinsic routines from various languages. In some programming languages, you need not (or cannot) give descriptions for procedures that are external to your program. When you designate that an external routine is an intrinsic, the compiler uses the Intrinsic Mechanism to correctly invoke the routine by:

- Providing a consistent intrinsic interface
- Ensuring proper data type conversion
- Generating proper reference parameter addresses
- Ensuring that the intrinsic is properly called

Although intrinsics usually refer to system routines, you can define routines that you want to access as if they were intrinsics and then place them in new or existing intrinsic files and libraries.

You invoke an intrinsic by calling it from within a program. In HP C/iX, HP Pascal/iX, COBOL II/XL, and HP FORTRAN 77/iX programs, you explicitly call an intrinsic. The Intrinsics Mechanism facilitates the declaration of system intrinsics. All MPE/iX intrinsics are processed as external procedures by user programs.

Before you can call an intrinsic from a program, you must declare it in all languages by using an intrinsic declaration statement. The format varies depending on the language. Refer to the appropriate language programming guides for details on how to call intrinsics. For detailed information on intrinsics and intrinsic declarations, refer to *MPE/iX Intrinsics Reference Manual* (32650-90028).

Compiling a Program

When you have saved the program source code in a file, it must be compiled; translated into machine readable instructions in a program file. The MPE/iX Native Mode language compilers available for this purpose are HP C/iX, HP Pascal/iX, HP COBOL II/XL, and HP FORTRAN 77/iX. Commands are available to:

- Compile, link, and execute a program, in one command
- Compile and link a program, in one command
- Compile a program

An overview of these commands is given later in this section. For detailed information on them, refer to :

- *HP C/iX Reference Manual Supplement* (31506-90001).
- *HP Pascal Programmer's Guide* (31502-90002).
- *HP COBOL II/XL Programmer's Guide* (31500-90002).
- *HP FORTRAN 77/iX Programmer's Guide Supplement* (31501-90002).

Compiler Input

Follow the appropriate instructions for the language you are using; HP C/iX, HP Pascal/iX, HP COBOL II/XL, or HP FORTRAN 77/iX. For example, to compile a program named `MYPROG` in HP Pascal/iX, enter:

```
:PASXL MYPROG
```

For further instructions on compiling, refer to:

- *HP C/iX Reference Manual Supplement* (31506-90001).
- *HP Pascal Programmer's Guide* (31502-90002).
- *HP COBOL II/XL Programmer's Guide* (31500-90002).
- *HP FORTRAN 77/iX Programmer's Guide Supplement* (31501-90002).
- *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364).

Compiler Output

The compiler can write compiled code in a relocatable object file, generate a source code listing, and generate an error list. Unless you specify differently, the compiler writes the object file to the standard file `$NEWPASS`, which is renamed `$OLDPASS` when the compile is completed and `$NEWPASS` is closed. The compiler writes the source code and error list to `$STDLIST`.

Compiler Operation

The compiler reads a source file as input, performs a translation to object code, and writes the resulting compiled code in relocatable object modules. The number of modules in a file is compiler dependent. MPE/iX compilers usually produce one relocatable object module in a relocatable object file. The internal structure of a relocatable object module is common to all compilers that generate Hewlett-Packard Precision Architecture (HP-PA) code. A relocatable object module contains a combination of code and data for all procedures in the source file that was the compilation unit. A relocatable object module is the smallest unit that a compiler can produce or the Link Editor can manipulate.

A source file with several procedures in it compiles all of them into one relocatable object module. The procedures within the relocatable object module cannot be replaced or purged individually.

When a large program is divided into several source files, each one can refer to external procedures (subroutines or variables that are defined in another file). Because MPE/iX compilers process only one source file at a time, external procedure references cannot be resolved at compile time. They are resolved at link time, when all of the program components are brought together. The compiler simply assigns a fix-up request (frequently called a relocatable address) to each external reference, indicating the relative position of each subroutine or variable in the relocatable object module.

Note A relocatable object module on MPE/iX corresponds to a relocatable binary module (RBM) on MPE V/E, with the following exceptions:

- A relocatable object module can contain zero, one, or several procedures, while an RBM represents only one.
- A relocatable object module is complete in itself and can stand alone as an independent file, while an RBM can exist only as part of an MPE V/E user subprogram library (USL) or a relocatable library (RL).

To separate relocatable object modules for two procedures into different relocatable object files, you must put the procedures in separate source files and compile them individually. You can gather them together at link time. Figure 3-2 shows an MPE/iX Native Mode compiler producing a relocatable object module.

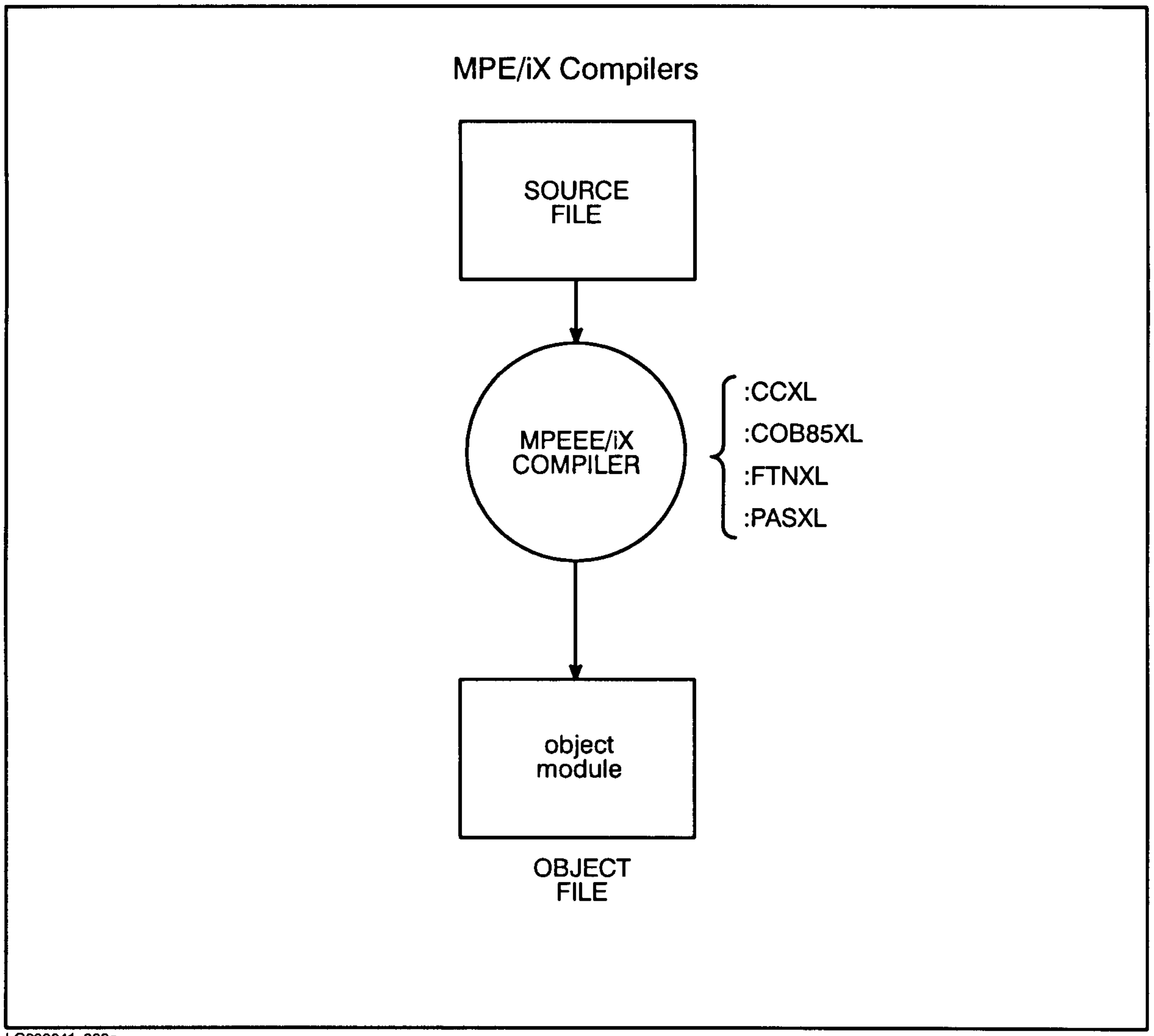


Figure 3-2. Compiler Producing Relocatable Object Modules

A relocatable object module is an independent file. It may not require resources such as relocatable libraries (RLs) and links to other relocatable object modules, although these are made, when necessary, at link time. External resources (usually system routines) are acquired at run time.

A relocatable object module contains the following information, described in detail in the subsections below:

- Compiled code (machine instructions) for all procedures in the source file
- Information on data variables to be used
- External references (resources required to run the program)

The compiler generates a symbol table in each relocatable object module. It lists all the procedures and variables that are defined in the module and may be required by other modules at link time and all those that are referenced in the module, but not defined there.

The compiler assigns a fix-up address to each defined subroutine or variable symbol, indicating its position relative to the beginning of the module.

Compiled Code

A compiler segregates code and data into separate areas in a relocatable object module. The individual compiler determines how compiled code is organized within it and the content is compiler dependent. Data constants are stored in the relocatable object module along with code and, thus, are non-modifiable. To change compiled code, you must recompile the entire relocatable object module.

Data Variables Information

A relocatable object module contains information on data requirements. Usually, it describes data types and required data space. The exact type of information is compiler dependent. It is used when initialization of variables is requested or when the data space has a common variable, which is a variable that several pieces of code can share.

Unresolved References

Because a compiler processes one source file at a time, it cannot resolve external references, determine parameter compatibility with them, or analyze actual data. This must be resolved at link time or run time. The compiler simply lists external references required for execution in a symbol table located in the relocatable object module, allowing you to compile a program in several pieces by separately compiling several source files. The symbol table lists all subroutines and variable names that are defined by the relocatable object module.

Compiler Libraries

Compiler libraries are used at run time by every program. On MPE/iX, they are stored in a library named `XL.PUB.SYS`.

Command to Compile Only

The command to compile a program (without linking, loading, and running it) is the command, followed by an optional list of file names. The commands are:

:CCXL
:COB85XL
:FTNXL
:PASXL

If you omit a file from the list, a standard default file is used. The standard default files are:

`$STDIN`	textfile
`$NEWPASS`	objectfile
`$STDLIST`	listfile

If you fail to supply a relocatable object module, a compiler opens a new file named `$NEWPASS` and designates it to be a relocatable object module. At the end of the compilation, `$NEWPASS` is automatically renamed `$OLDPASS` and saved. Unless you designate differently, the compiler listing produced by the compilation is output to your job or session list device, so you can see any errors.

You can obtain two types of output from a compiler, aside from the compiler listing of errors:

- Source code translation of the output to a relocatable object module.
- Source listing, with various map and table options.

Compiler Control

If you wish only to compile a program, without linking and running it, you can use the "compile only" command for the appropriate language. These commands allow you to provide several optional parameters that specify the name of the textfile (source file), objectfile (relocatable object module), and listfile. The text file contains the source code for the program. The relocatable object module will contain the output from the compile. The listfile, which is usually on a terminal or printer, will reflect the progress of the compile.

When source code has been successfully compiled and you have an error-free relocatable object module, you are ready to link a program file from information in one or more relocatable object modules. A software product called Link Editor performs this operation.

Linking a Program

Many executable programs are originally generated from more than one source file. On MPE/iX, each source file is compiled separately to produce a relocatable object module. The linking phase of program development makes a relocatable object file into an executable program file. It can also bring together separately compiled relocatable object modules into an executable program file.

Use the MPE/iX command `:LINK` to automatically access Link Editor to create an executable program file from one or more relocatable object modules.

Note Link Editor on MPE/iX replaces Segmenter on MPE V/E. You can still use Segmenter in MPE/iX Compatibility Mode. For detailed information, refer to *MPE Segmenter Reference Manual* (30000-90011). For an explanation of the differences between Link Editor and Segmenter, refer to *Link Editor/XL Reference Manual* (32650-90030). `:LINK` is the MPE/iX counterpart to `:PREP` on MPE V/E systems. However, it operates differently and has different options. For an explanation of the differences between `:LINK` and `:PREP`, refer to *Link Editor/XL Reference Manual* (32650-90030). Figure 3-3 shows the role of `:LINK` and Link Editor in MPE/iX.

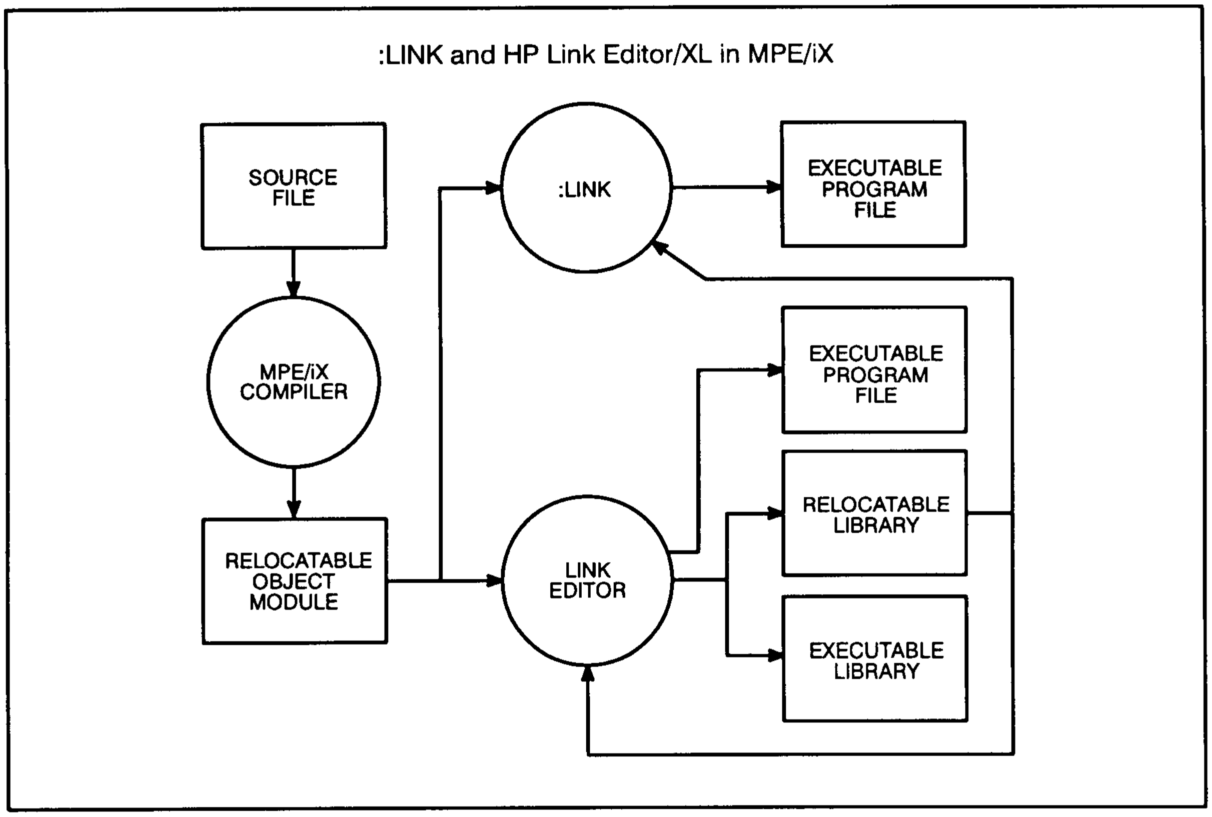


Figure 3-3. Linking and Using User Libraries

`:LINK` operates on one or more relocatable object modules created by a native compiler to perform the following tasks:

- Search one or more relocatable libraries (RLs) and resolve references to them in any of the relocatable object modules being linked.
- Resolve references among the relocatable object modules.
- Merge all the relocatable object modules associated with one program to create an executable program file.

If you use language commands that compile, link, and execute in one command (for example, `:PASXLGO` and `:COB85XLG`) or compile and link in one command (for example, `:PASXLLK` and `:FTNXLLK`), then linking is automatic. When you compile separately, you can use the MPE/iX command `:LINK` to produce an executable program file. Use this method when you want to use values different from standard Link Editor defaults. `:LINK` invokes Link Editor and passes the specified parameters to it. (An analogous `LINK` command exists in Link Editor; the MPE/iX command `:LINK` simply provides a short cut.)

Creating Executable Program Files

The MPE/iX command `:LINK` creates a load module (an executable program file). The `:LINK` command invokes Link Editor, which is an MPE/iX subsystem that prepares compiled programs for execution and maintains libraries. When invoked by using `:LINK`, Link Editor resolves external references in relocatable object modules by merging relocatable object modules to produce an executable program file containing all of the code and data that was in the relocatable object modules.

It assigns final addresses to each symbol (for a variable or procedure) to ensure that none overlap each other in the executable program file. These addresses are called virtual addresses, because they are still subject to a final relocation when the program is loaded into physical memory. Once virtual addresses are determined, Link Editor can resolve many of the references that could not be resolved at compile time, because the symbol tables from all of the relocatable object modules have been merged to one table. You can specify the relocatable libraries to search for unresolved references at link time. Any remaining external calls must be resolved at load time.

`:LINK` parameters can specify indirect files. An indirect file is a file list contained in an ASCII file. For example, in the `:LINK` command, an indirect file can be used to specify one of the following lists:

- Files to link
- Relocatable libraries (RLs) to merge
- Executable libraries (XLs) names to put in a program header

For an overview of Link Editor, refer to Chapter 4. For detailed information on `:LINK` and Link Editor, refer to *Link Editor/XL Reference Manual* (32650-90030).

You can request options at link time to:

- Produce a program file map.
- Search specified relocatable libraries (RLs) for any required code not found in the relocatable object module.
- Give the executable program file certain privileges and capabilities.

Symbol Listing

Example 3-1 shows a source file named `EX1SRC`. An executable program file has been created for `EX1SRC` named `EX1PROG`.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EX1
ENVIRONMENT DIVISION.
INPUT-OUTPUT DIVISION.
FILE-CONTROL.
     SELECT IFILE             ASSIGN "IFILE".
     SELECT PFILE             ASSIGN "PFILE".
DATA DIVISION.
FILE SECTION.
FD   IFILE.
01   IREC.
     05  NAME                 PIC X(30).
     05  SOC-SEC              PIC X(9).
     05  HIRE-DATE.
         10 MO                PIC XX.
         10 DA                PIC XX.
         10 YR                PIC XX.
     05  SALARY               PIC S9(6).
     05                       PIC X(29).
FD  PFILE.
01  PREC.
     05  SOC-SEC              PIC X(9).
     05                       PIC XX.
     05  NAME                 PIC X(30).
     05                       PIC XX.
     05  HIRE-DATE.
         10 MO                PIC XX.
         10                   PIC X.
         10 DA                PIC XX.
         10                   PIC X.
         10 YR                PIC XX.
     05                       PIC X(81).
01   HREC.
     05  HSOC-SEC             PIC X(11).
     05  HNAME                PIC X(32).
     05  HHIRE-DATE           PIC X(89).
```

Example 3-1. Source File Example (page 1 of 2)

```
WORKING STORAGE SECTION.
01  LNCNT                    PIC S9(4) BINARY VALUE 60.
01  W-DATE.
     05  WYR                 PIC XX.
     05                      PIC X(4).
PROCEDURE DIVISION.
P1.
     ACCEPT W-DATE FROM DATE.
     OPEN INPUT IFILE OUTPUT PFILE.
     PERFORM WITH TEST AFTER UNTIL SOC-SEC OF IREC = ALL "9"
          READ IFILE
               AT END MOVE ALL "9" TO SOC-SEC OF IREC
               NOT AT END
                    IF WYR = YR OF IREC THEN
                         ADD 1 TO LNCNT
                         IF LNCNT > 50 PERFORM HEADINGS END-IF
                         MOVE SPACES TO PREC
                         MOVE CORR IREC TO PREC
                         WRITE PREC AFTER ADVANCING 1 LINE
                    END-IF
               END-READ
          END PERFORM
     CLOSE IFILE PFILE
     STOP RUN.
HEADINGS.
     MOVE "SOC-SEC" TO HSOC-SEC.
     MOVE "NAME" TO HNAME.
     MOVE "HIRE DATE" TO HHIRE-DATE.
     WRITE PREC AFTER ADVANCING PAGE.
     MOVE 0 TO LNCNT.
```

Example 3-1. (page 2 of 2)

The commands

```
:LINKEDIT
LinkEd> LISTPROG EX1PROG
```

invoke HP Link Editor/XL and create a program map displaying the symbols in `EX1PROG`, as shown in the following example:

```
PROGRAM        : EX1PROG
XL LIST
CAPABILITIES   :  BA, IA
HEAP SIZE      :
STACK SIZE     :
VERSION        : 85082112

Sym             C H X P Sym    Sym          Sym       Lset
Name                    Type  Scope        Value      Name
----            - - - - ----  -----        -----      ----
$START          0   3 3 sec_p univ        000059B7
_start          0   3 3 sec_p univ        00005A07
ex1             0   3 3 pri_p univ        000059EB
M$1             0       data  local    dp+00000000
```

In this program map, the portion preceding the symbol table is the header, which provides general information about the executable program file:

- `PROGRAM` names the executable program file.
- `XL LIST` shows the names of executable libraries specified in the `XL` parameter of the `LINK` command.
- `CAPABILITIES` shows the capabilities assigned to the program in the `CAP` parameter of the `LINK` command.
- `HEAP SIZE` shows the value specified in the `NMHEAP` parameter of the `LINK` command.
- `STACK SIZE` shows the value specified in the `NMSTACK` parameter of the `LINK` command.
- `VERSION` shows the executable program file format version.

The header information is followed by a list of symbols in the executable program file. For information on understanding the symbol listing, refer to *Link Editor/XL Reference Manual* (32650-90030).

Loading and Running a Program

The loader performs the following tasks:

- Loads the executable module into virtual memory locations.
- Updates addresses contained in an executable object module with the actual memory addresses.
- Resolves unsatisfied external references to executable libraries. If any unsatisfied external references remain, the loader aborts.

The loader loads executable libraries to resolve external references by linking the import requests of the unresolved reference to an export definition contained in the executable library. The actual link is made by pointer reference to shared code. No object module in an executable library can have access to the caller's data without passing parameters.

The first step in running a program is to load it. The MPE/iX Loader performs the final step in preparing a file for execution in Native Mode and Compatibility Mode. When operating in Compatibility Mode, the loader simulates the MPE V/E Loader, with only the changes necessary to make it run.

MPE/iX Loader input consists of an executable program file and an optional set of executable libraries.

The loader initializes code and data to create table entries needed to execute or access the code and data and creates links to connect program references to executable libraries.

A program load performs the following tasks:

- Maps executable module into virtual memory by converting the code relative addresses to absolute addresses. (Code relative addresses are those which are relative to the beginning of the executable program file.)
- Changes the access rights of pages that contain code. Code is executed without copying it; it is mapped to virtual memory. Write access to program code is not necessary on MPE/iX; only read and execute access is granted to code pages.
- Creates global data areas for the program and each module in referenced executable libraries.
- Copies global data initialization information into process data space and sets the appropriate register to point to it.
- Generates external reference list and attempts to locate all entries in the list.

Program Auxiliary Header

The MPE/iX program auxiliary header resides in an executable program file, is generated by HP Link Editor/XL, and is used by the loader. It specifies the following information:

- Primary entry point name
- `UNSAT` procedure name
- `XL LIST` (the list of XLs specified at link time)
- Program capabilities
- Maximum stack and heap sizes

MPE/iX allows you to change the specifications listed above at load time, except the primary entry point name and program capabilities. Any information you specify at run time overrides the specifications given by HP Link Editor/XL.

If the program auxiliary header is not specified, the loader uses the default values for the `:RUN` command. Libraries specified at run time take precedence over those specified at link time.

Using Executable Libraries

On MPE/iX, you can run a Native Mode program and specify the use of multiple libraries with different names, rather than just one executable library (XL) at a time.

The syntax for running a program that utilizes one or more executable libraries is as follows:

`:RUN prog; XL = "`*xlib1*[,*xlib2*][,*xlib3*] ... `"`

In this syntax, *xlib1*, *xlib2*, *xlib3*, ... are replaced by the names of the libraries you want to execute and `prog` is a Native Mode program name.

You can specify any number of XLs, up to a maximum limit of 288 characters for the complete command. You can use any valid file name for an XL. The system library is added to the end of the list by default. However, if you specify it, you must make it the last library listed. XLs that you specify at run time override those specified at link time.

Searching Executable Libraries

The following guidelines apply to searching executable libraries (XLs):

- The library list can be specified at Link time, but is not actually used until run time.
- Run-time libraries can appear only once in the `:RUN` command.
- You must list libraries in order of increasing privilege level.
- If the system library is not specified, the system automatically adds `XL.PUB.SYS` and `NL.PUB.SYS` as the last libraries to search.

UNSAT Procedure

You can load a program that has one or more referenced external procedures that cannot be located by specifying an UNSAT procedure. This is a dummy procedure specified in the `UNSAT` parameter of the `:LINK` command. This procedure is substituted for the missing library routine. You can set up the procedure to contain statements that facilitate program execution in the absence of the real routine. For example, the execution of the UNSAT procedure could print a statement informing you that it was called.

Note The content of an UNSAT procedure determines whether or not a program can continue to run after the UNSAT procedure is called.

When the loader encounters an UNSAT procedure, it uses it to resolve all remaining unresolved references. The UNSAT procedure iself may be written in any language; however, it must be compiled and put into one of the executable libraries that is specified at run time. Figure 3-4 shows an example of using an `UNSAT` procedure in a program.

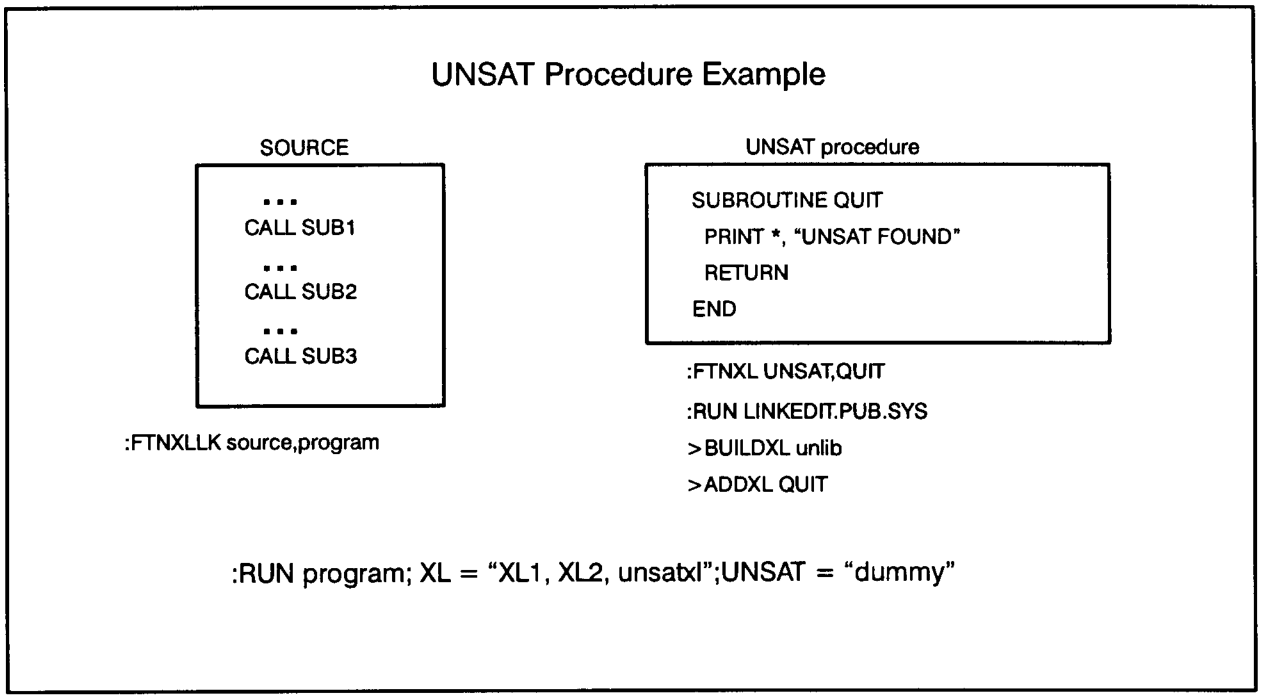


Figure 3-4. UNSAT Procedure Example

System Libraries

System library characteristics include the following:

- They are loaded in system space.
- Processes using the same object module share code, global data, and binding.
- Object modules can reference only themselves or other system library object modules.
- Cannot use `UNSAT` procedures.
- System libraries are the only libraries that can contain both system and nonsystem object modules. (A nonsystem object module shares only the code with other processes. Its data and binding are unique to each process that references it. A nonsystem object module is loaded once for each process that uses it.) The Native Mode system library is named `NL.PUB.SYS`. The Compatibility Mode system library is named `SL.PUB.SYS`.

Mixing Execution Modes

While the operating system is executing for a process, the process may switch execution modes. It can alternate repeatedly between Native Mode (NM) and Compatibility Mode (CM). MPE/iX provides switch stubs to allow NM programs to access CM intrinsics. The operating system intrinsic call determines when to use the NM Executable Library (NL) or the CM Segmented Library (SL).

The NM intrinsic file is `SYSINTR.PUB.SYS`; the CM intrinsic file is `SPLINTR.PUB.SYS`.

You can set up a Native mode program to call procedures that are in a CM Segmented Library (SL) by using the switch intrinsics. This requires that the program specify the switch stub. The switch intrinsic uses the `LOADPROC` procedure to find the CM procedure. For detailed information on the switch subsystem and programmatic access through switch stubs, refer to *Switch Programming Guide* (32650-90014).

Virtual Memory and Demand Paging

When a program is running, only part of it is needed at any one time. To save space in main memory, MPE/iX brings pieces of a program in, as needed for current execution. It divides a program into fixed-length pieces called pages (for a description of pages, refer to Chapter 1). The remainder of the program can be stored on a high-speed device (for example, a disc) that can act as an extension of real memory. This extension is called virtual memory. The use of virtual memory reduces to a minimum the problem of application program size compared to available memory size, because it eliminates the requirement for loading all code and data into main memory at once.

LMAP: Load MAP

An `LMAP` facilitates code management by listing a loaded program. It describes the spaces loaded for a process and the linkages used to connect the external references of the process for the program and each library specified by the user. To produce a load map (map of a loaded program), specify the `LMAP` parameter of the `:RUN` command. To print an `LMAP`, use `:FILE` to define the file with the formal file designator `LOADLIST` as a line printer device file. For detailed information refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364). A load map can also help to determine if you have achieved good localization. An `LMAP` shows:

- File name and type of file (program or library)
- Number of external references a program file makes
- Type of library from which each call is resolved

Load-time Binding Sequence

The loader assigns virtual addresses and makes an executable program file into a runnable process.

Running a Program

When a program has been successfully linked, you can load it and execute the `:RUN` command. At this stage of operation, the program is called a process. A process is a unique execution of a particular program by a particular user at a particular time. If you change even one element of a program, it becomes a different process than it was before.

When you execute a program, a piece of software called the LOADER starts the process on the system. The loader must resolve any external references in the program by using executable libraries.

MPE/iX is a multiprogramming operating system, which has many processes competing for CPU time. Only one process can execute at a given instant in time. When a program has had external references resolved it is ready to execute. It can now compete with other processes for the resources of main memory and CPU time.

The `:RUN` command requires you to specify the filename of the program and allows you to specify many other options, such as obtaining a program load map. You can specify values for process stack and heap size to override values specified at the link stage. There are facilities for passing information to a program at run time and determining which libraries are searched to resolve external references.

Multi-programming Environment

MPE/iX is a multiprogramming environment in which one process has control of the CPU at any given instant. All other processes are suspended to allow time for this to occur, and the processes take turns. The frequency and duration of a turn is determined by the MPE/iX Dispatcher and is dependent on the priority assigned to the process. Processes can voluntarily suspend for many reasons. For example, a process may suspend because it is waiting for input or output. The code and data for suspended processes is retained in memory until space is needed by the executing process.

Priority Levels

A priority number identifies the priority level for a process. The lower the number, the higher the priority. Priorities range from 1 to 255. MPE/iX uses a mechanism of subqueues to determine when to change the priority level of a process, if at all. Operating system processes are given the highest priority to promptly service requests. The AS subqueue (used for most operating system processes) receives the highest priorities, followed by the BS subqueue (usually used for special user processes). The CS, DS, and ES subqueues (used for the Command Interpreter, Editor, compilers, and standard user processes), as a group receive the lowest priorities.

Operating system processes are given the highest priority to promptly service requests. At the user level, processes are queued and given CPU control based on rules for the types of queue used at the particular level of the process. The operating system always runs the highest priority process that is ready to execute. A system manager can assign to users and accounts privileges or restrictions of one subqueue over another. A system supervisor has some control over the priority levels in CS, DS, and ES.

Linear Subqueues

The AS and BS subqueues are linear queues. The characteristics of a linear queue are:

- MPE/iX does not change the priority level of a process.
- Processes at a given priority level execute on a first come, first served basis.
- An executing process controls the CPU until it voluntarily suspends or receives a higher priority interrupt.

The AS subqueue should be reserved for operating system processes. A user process running in AS can severely impair the operating system's ability to function. The BS subqueue is for special high priority user processes. It is a recommended practice to run a special user process at a priority of 140 or lower to avoid preempting important MPE/iX functions, such as spooler operation.

Circular Subqueues

The CS, DS, and ES subqueues are circular queues. The characteristics of a circular queue are:

- Priority levels are cycled.
- An executing process controls the CPU until it voluntarily suspends, receives a higher priority interrupt, or reaches the end of its quantum (time slice).

A circular queue lowers the priority of a process at the end of its quantum. After the drop, the process eventually gets another opportunity to execute. The CS, DS, and ES subqueues differ in how low a priority can be dropped.

Error Detection

The types of errors that can occur on the MPE/iX operating system are in the categories given below, along with suggested actions:

Command Interpreter Errors

Errors in using Command Interpreter (CI) are usually the simplest errors to detect. They include typing errors and syntax mistakes. For correcting typing errors, use the :REDO command. For syntax errors or misunderstandings in how the command works, use the `:HELP` command or refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364).

File System Errors

File System errors are common at all levels of user interface because file access affects almost every kind of operation. When you are unable to open, access, or close a file, consider the rules governing the file's domain, access, and security. For detailed information, refer to *Accessing Files Programmer's Guide* (32650-90017).

Compiler, Link Editor, and Loader Errors

You may have exceeded a configured system limit with program files; ask the system manager for information. If a program load fails, it is probably marked unrunnable, in which case you must link it again or restore the file.

Unresolved externals do not mark the file unrunnable. If this problem occurs, check for errors in the subprogram names and `XL=` names. Determine if the routines you requested actually reside in the Executable Libraries you searched.

For errors reported by a compiler, refer to compiler information in the appropriate language manual.

Run-time Errors

Run-time errors present a challenging problem, frequently causing the program to end abnormally (abort). To determine the answer to a run-time error, you must consider the following aspects of the problem:

- What is the cause or source?
- What is the mechanism of the abort?
- What information is provided to help trace and correct the problem?
- What are typical abort situations?

The decision to abort a process can be made by:

- User program
- Library routine
- MPE/iX intrinsic
- MPE/iX operating system

A program's design can cause it to abort if bad input or some unrecoverable situation occurs. The program can call the `QUIT` intrinsic or an equivalent compiler-determined statement. In this case, the unrecoverable situation is by design; the programmer must provide code to handle the problem or abort the program.

A process can abort because a subsystem library routine encountered a problem. For example, if an HP FORTRAN 77/iX program performs a `READ` statement, an HP FORTRAN 77/iX library routine is called to perform the necessary I/O. If an end-of-file condition is encountered, a data format problem occurs. If the library routine cannot complete the operation, it may abort the program.

An MPE/iX intrinsic can abort a program, depending on the individual intrinsic's requirements. For example, it may encounter a missing parameter, an unusable parameter value, or an unusable address. The intrinsic might require that the program file have a special capability to use the intrinsic.

900 Series HP 3000 hardware or software can detect errors. For example, if an arithmetic operation exceeds a data value maximum, computer hardware detects an overflow problem. It may also detect a bad instruction or invalid address for code or data. MPE/iX aborts a program when a stack requires more room than is available. This alerts you to a potential problem with recursive calls and loops.

Abort Message Information

When MPE/iX carries out an abort for any reason, the process must terminate cleanly (complete the operations it would have completed if it had ended normally). However, because the termination is unexpected, information may be lost. For example, NEW files may not be retained. A message indicating the type of abort is printed before termination. After termination, Command Interpreter prints the final message line. The information in the message helps to identify the cause and guide you toward a solution to the problem.

An abort message appears in a standard format providing:

- Program file name
- Location at which the abort occurred in the program file
- Message text describing the type of abort

If the abort occurred during execution of an intrinsic, a subsystem library routine, or a user library routine, then the location is given within the appropriate routine and the program file location indicates the location from which the routine was called in the user's program.

For detailed information on reading an abort message, refer to *MPE/iX Intrinsics Reference Manual* (32650-90028)| or the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364).

Typical Causes of Program Aborts

Two causes of abort that occur frequently are data memory protection traps and aborts in library routines.

A data memory protection trap occurs when a program attempts to address outside the bounds of the code/data area. This can be caused by a faulty address or index used for a subscript. For example, a parameter address may be passed as a byte address, rather than a word-aligned address expected by the called routine. Address corruption can occur if an array subscript goes out of the bounds of the array, but is still inside the bounds of the stack.

When an abort occurs in a library routine, the problem is most frequently in the user program. Examine the program for file equations that you may have overlooked. It is possible, although considerably less frequent, that a bug actually exists in the library routine itself.

File Information Display (Tombstone)

It is frequently necessary to obtain status information on a file to determine the cause of an error. A file information display is frequently called a tombstone. It provides:

- Actual physical and operational file characteristics.
- Current file information, pertaining to end of file, record pointer, and logical and physical transfer count.
- Information on the last error for the file and the last `HPFOPEN` or `FOPEN` error.

When a file is opened, the final characteristics may be different from those originally requested because of defaults, overrides, `:FILE` commands, and the file label.

You can use the `PRINTFILEINFO` intrinsic to print a tombstone. It requires that you specify the file number returned when the file is opened by `HPFOPEN or FOPEN`. The tombstone can display either a full or short format. If the file is open, it provides a full display. Otherwise, it provides a short display. Calling this intrinsic does not automatically abort the program.

You can call the `PRINTFILEINFO` intrinsic from programs written in COBOL II/XL and HP FORTRAN 77/iX. When calling from COBOL II/XL, use the FD filename. You can call the name `PRINTFILEINFO` directly from HP FORTRAN 77/iX programs. You can obtain the required file number by using the `FNUM` intrinsic.

Control Codes (JCWs)

A Job Control Word (JCW) is one word (16 bits) of memory used to pass information between job steps and to determine the status of the previous job step. Figure 3-5 shows the structure of a JCW. It is composed of a 2-bit type and a 14-bit modifier.

The space allowed for JCWs is determined by the space allowed for your session or job after file equations, temporary files, and data space is taken. There is always one system JCW and one user-defined JCW named `CIERROR` that is actually defined by Command Interpreter (CI). You can define as many user-defined JCWs as remaining space allows.

You can use the MPE/iX command `:SHOWJCW` to see the value of JCWs. If you have not defined any JCWs of your own, this shows only the value of the system JCW and `CIERROR`.

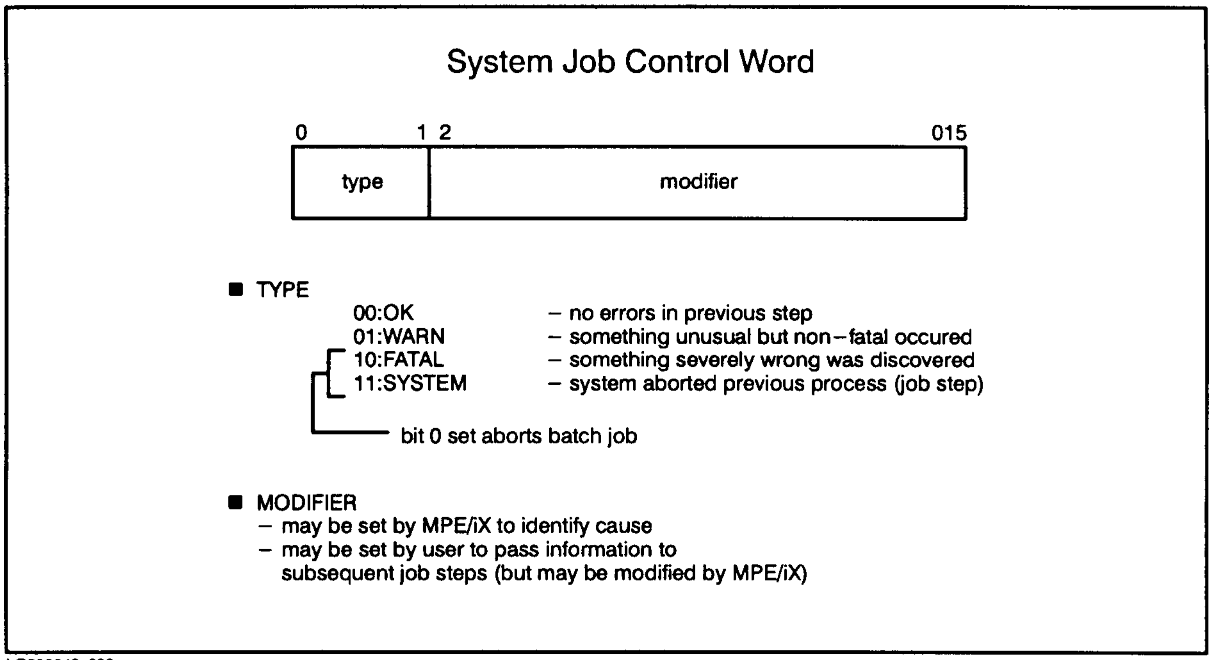

System Job Control Word
0
1 2
015
type
modifier
TYPE
00:OK – no errors in previous step
01:WARN – something unusual but non–fatal occured
10:FATAL – something severely wrong was discovered
11:SYSTEM – system aborted previous process (job step)
bit 0 set aborts batch job
MODIFIER
– may be set by MPE/iX to identify cause
– may be set by user to pass information to subsequent job steps (but may be modified by MPE/iX)
LG200042_033a

Figure 3-5. Job Control Word (JCW) Structure

System JCW

The system JCW is set by MPE/iX and some subsystems. You can set the system JCW programmatically by using intrinsics or interactively at the Command Interpreter (CI) level by using the `:SETJCW` command.

MPE/iX checks the system JCW before each step (process) in a session or job is executed. Based on the system JCW value, MPE/iX may abort the session or job. You can interactively check the system JCW at the CI level by using the `:SHOWJCW` command. You can programmatically check by intrinsics (calling `:FINDJCW`) or by using an `:IF/THEN`, `:ELSE`, `:ENDIF` construct.

The system JCW type provides information on the severity of an error. The modifier can be set to identify the cause or to pass information to subsequent job steps. However, information placed there by the user may not be preserved, because MPE/iX can change it. Thus it is frequently expedient to place information that you want preserved in a user-defined JCW instead of in the system JCW. When set by the user, the modifier may be modified by MPE/iX. The types are the bit settings given below, accompanied by their keywords, and descriptions.

`00:OK`	No error occurred in the previous step.
`01:WARN`	An unusual event occurred, but not necessarily fatal.
`10:FATAL`	Program aborts, under its own control.
`11:SYSTEM`	System aborts user process due to a problem outside the process's direct control.

The modifier can be set to any number from 0 to 65535, inclusive. These numbers are divided up for use by each of the four types given above:

```
OK      0-16383
WARN    16384-32767
FATAL   32768-49151
SYSTEM  49152-65535
```

JCW Notation

A JCW value can be described with three kinds of notation:

- A keyword, followed by a constant. The constant is the decimal equivalent of the octal number in the modifier.
- Modifier appended to the type to form one constant.
- The numerical value of the modifier.

Figure 3-6 shows examples of JCW notations.

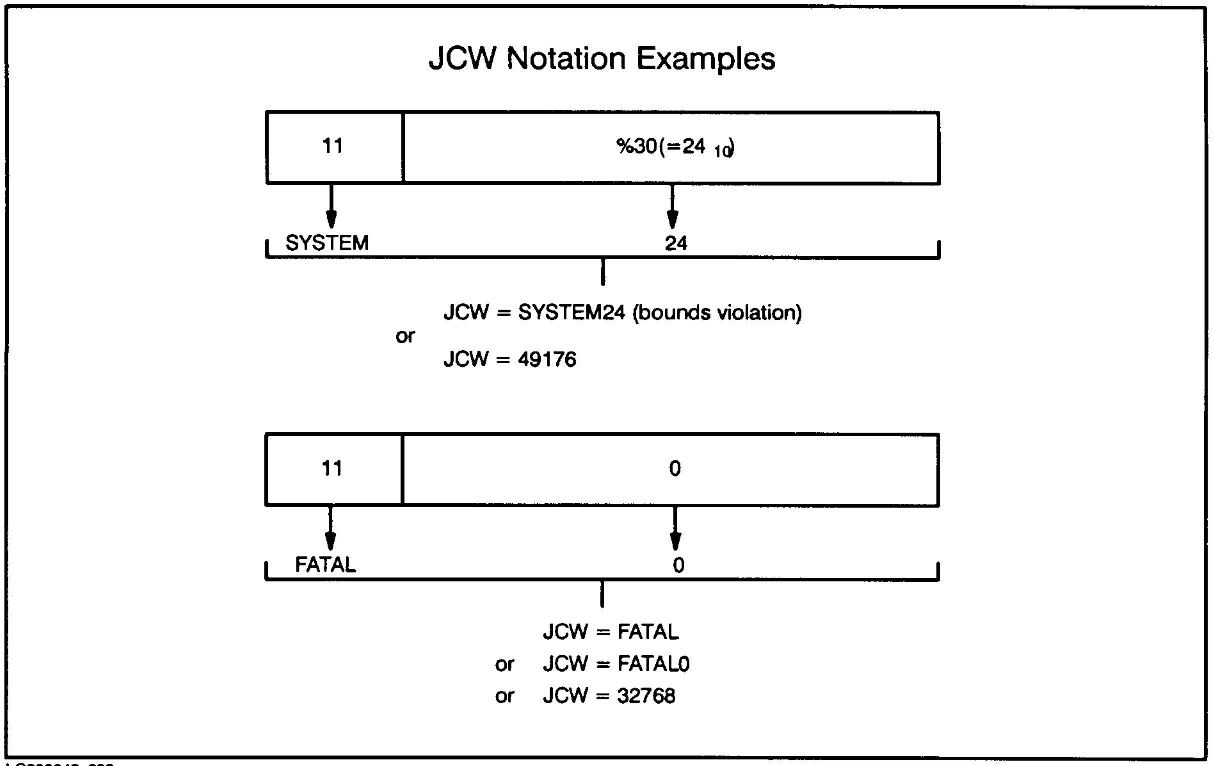


Figure 3-6. JCW Notation Examples

Using a System JCW

The following job shows an example of using a system JCW:

```
:JOB STUDENT.INTRO/PASSWORD
:CONTINUE
:FTNXL ABC
:IF   JCW<FATAL THEN
:      LINK $OLDPASS,ABCPROG
:      SAVE ABCPROG
:      RUN ABCPROG
:ELSE
:      SAVE $OLDPASS,ABCOBJ
:ENDIF
:SHOWJCW
:EOJ
```

This job compiles an HP FORTRAN 77/iX program. If it works, you want to link the object file to your program file and run the program. If a problem occurs during the compile, you want to save the file and end the job.

If an error occurs, HP FORTRAN 77/iX sets `JCW = FATAL`, so the job tests the value of the JCW against the value `FATAL` (`FATAL` is the same as `FATAL0` or `32768`). The `:CONTINUE` command causes the job to proceed even if the next step produces an error. The :IF/THEN, :ELSE, :ENDIF structure allows you to make a decision based on the JCW value following the :FTNXL job step. The system-defined JCW that CI creates is called CIERROR. CI sets the JCW modifier to the last CI error number. Sometimes, CI aborts a job based on the value of the CIERROR modifier.

User-defined JCWs

User-defined JCWs allow you to supply input to programs by using Command Interpreter (CI) and get status from programs that can be examined in CI. Characteristics of user-defined JCWs include:

- Available to each process in the same session or job.
- Have names different from JCW or CIERROR.
- Are created and set interactively with :SETJCW or programmatically with the PUTJCW intrinsic.
- Can be examined interactively with :SHOWJCW or programmatically with the FINDJCW intrinsic.

MPE/iX does not examine user-defined JCWs. The value of a user-defined JCW has meaning only to the user. Keywords and definitions assigned to the type and modifier can be identical to those assigned to system JCWs or different. You determine what action to take based on your own definitions.

Using a User-defined JCW

The following job shows an example of using a user-defined JCW.

```
:JOB STUDENT.INTRO/PASSWORD
:SETJCW UPDATE=OK
:RUN UPDATEDB
:IF UPDATE=OK THEN
:    RUN REPORT
:ELSE
:    SHOWJCW
:    TELLOP REPORT NOT RUN
:    TELLOP TOO MANY INPUT ERRORS
:ENDIF
:EOJ
```

In this example, the job stream begins by defining and setting a user-defined JCW called UPDATE to the value OK (OK is the same as OK0 or 0). Running the program named UPDATEDB edits a database. If too many errors occur, UPDATEDB changes the value of UPDATE by using the PUTJCW intrinsic. The :IF/THEN, :ELSE structure tests the value of UPDATE. If the program is successful (in other words, if there are not too many errors), it runs another program called REPORT. If too many errors did occur in the UPDATEDB program, the job shows the JCW values, sends two messages to the console operator, and ends.

4

HP Link Editor/XL

HP Link Editor/XL is a software tool that prepares compiled programs for execution on Series 900 HP 3000 computers and allows you to create and maintain libraries containing subprograms that you frequently use.

Most MPE/iX compilers let you compile, link, and execute a program, all in one step, or just compile and link in one step. In these cases, you do not directly execute HP Link Editor/XL to perform the linking function; it is executed automatically.

This chapter describes the basic function of HP Link Editor/XL. For detailed information on how to use HP Link Editor/XL, refer to *Link Editor/XL Reference Manual* (32650-90030).

HP Link Editor/XL uses one or more relocatable object modules produced by one or more native compilers to create one of the following files:

- Relocatable library (RL), which is a collection of relocatable object modules that can be used by many programs at link time.
- Executable library (XL), which is a collection of executable modules that can be shared by many programs at run time.
- Executable program file, which is the result of merging all relocatable object modules associated with one program.

It can also operate on an existing RL or XL to update it. Figure 4-1 shows the role of `:LINK` and HP Link Editor/XL on the MPE/iX operating system.

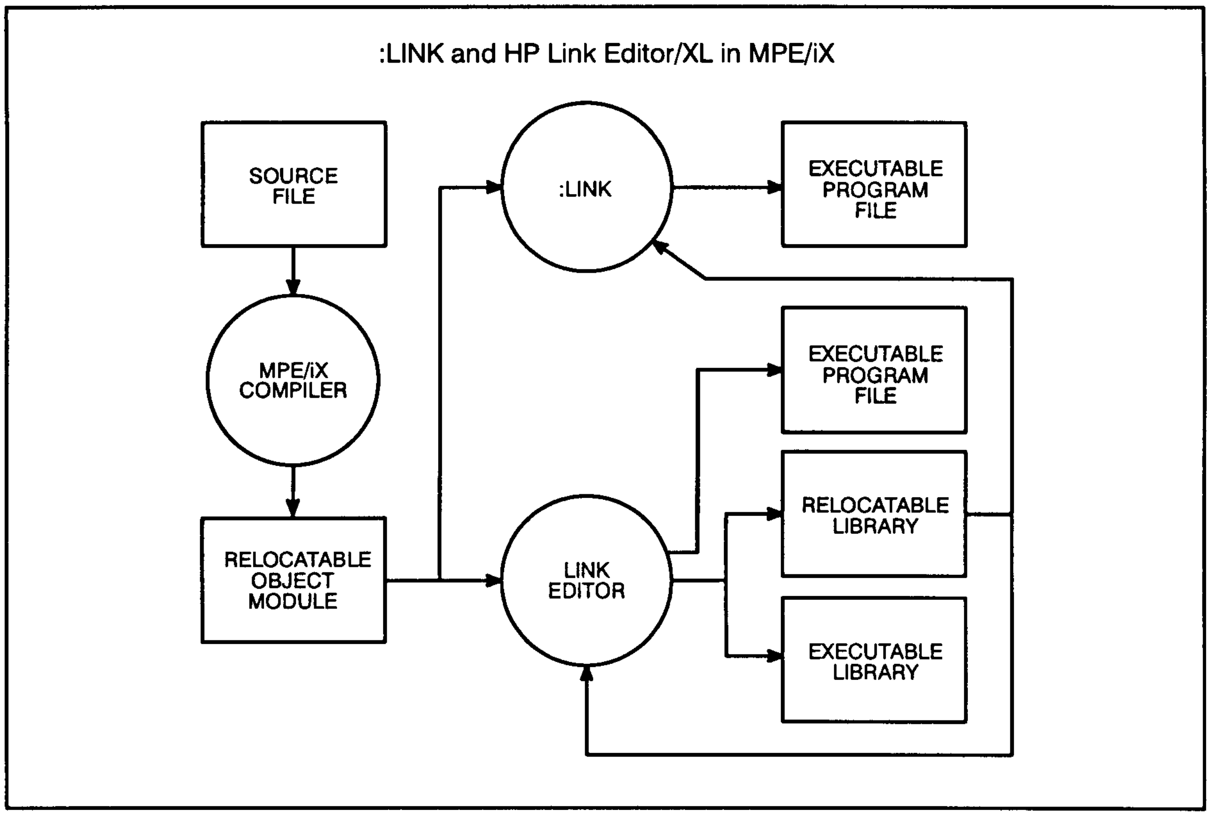


Figure 4-1. :LINK and HP Link Editor/XL on MPE/iX

You automatically enter HP Link Editor/XL for the purpose of manipulating relocatable object modules into an executable program file by using the MPE/iX command `:LINK`. This command is especially useful in program development, when you must link a large program that calls many separately compiled routines or when you want to use values different from standard HP Link Editor/XL defaults for this process. For a `:LINK` overview, refer to the "Linking a Program" subsection of Chapter 3. For detailed information on linking, refer to *Link Editor/XL Reference Manual* (32650-90030).

You can enter HP Link Editor/XL to:

- Create an executable program file (the executable form of a program), which includes several different modules that have been compiled separately.
- Change one or more of the default parameters associated with the program. For example, you may need to change the execution stack size.
- Use one or more library routines in your program. HP Link Editor/XL creates and maintains two kinds of libraries: relocatable libraries and executable libraries. Routines in relocatable libraries are in their compiled format. Routines in executable libraries are in executable form. Libraries minimize duplication of programming effort, promote consistency and standardization in a programming organization, and help to produce easily maintained programs.

Common Uses of HP Link Editor/XL

The most common ways to use HP Link Editor/XL are to:

- Link one or more relocatable object files.
- Link relocatable object modules from a relocatable library.
- Name executable modules from an executable library to be searched at load time.
- Obtain program file information by using the `LISTPROG` command.

Linking a Relocatable Object File

When you want to use file names or run-time defaults that vary from the defaults provided by the compiler, link a relocatable object file yourself, rather than have it automatically linked. (Run-time defaults include type checking levels, capability-class attributes, stack size, and heap size.)

For example, you can compile, link, and execute a COBOL II/XL program named `EX1SRC`, by using the command

```
        :COB85XLG EX1SRC
```

This command is equivalent to invoking the following three commands:

```
        :COB85XL EX1SRC
   :LINK
   :RUN $OLDPASS
```

Both of the methods given above use `$OLDPASS` for the relocatable object file and for the executable program file.

The example below shows how to compile and link the COBOL II/XL source file, EX1SRC, using a different relocatable object file and a different executable program file. The compiler creates the relocatable object file named `EX1OBJ`, which the HP Link Editor/XL command `LINK` uses to create the executable program file named `EX1PROG`. The second line of this example is the HP Link Editor/XL command line.

```
        :COB85XL EX1SRC,EX1OBJ
   :LINK FROM=EX1OBJ;TO=EX1PROG
```

For a listing of the source file named `EX1PROG` and examples of linking several relocatable object files at once, refer to *Link Editor/XL Reference Manual* (32650-90030).

Comparison of HP Link Editor/XL and MPE V/E Segmenter

HP Link Editor/XL on MPE/iX corresponds to Segmenter on an MPE V/E operating system, with a few differences. The primary differences are:

- A source file must be recompiled to change any part of a relocatable object module.
- You can create a relocatable library (RL) that contains one large relocatable object module; or you can create RLs that contain several relocatable object modules, which can be independently added, copied, or purged.
- An executable library (XL) is similar to a segmented library (SL), except that it does not require segmenting as it grows.
- You can create an XL that contains one large executable object module; or you can create XLs that contain several executable object modules, which can be independently added, copied, or purged.

For a detailed description of the differences between HP Link Editor/XL on MPE/iX and Segmenter on MPE V/E, refer to *Link Editor/XL Reference Manual* (32650-90030).

How HP Link Editor/XL Works

HP Link Editor/XL processes object code produced by high-level language compilers, such as COBOL II/XL. Object code is saved in relocatable object files. HP Link Editor/XL links relocatable object files for execution by assigning actual memory locations to them and to any external subroutines that they use.

In addition to creating executable program files, you can use HP Link Editor/XL to create and maintain relocatable and executable libraries.

When you invoke the MPE/iX command `:LINK`, HP Link Editor/XL is automatically entered, linking is completed, and HP Link Editor/XL is exited. Information required by HP Link Editor/XL is passed to it as `:LINK` parameters.

Files Used by HP Link Editor/XL

HP Link Editor/XL can use three types of input files:

- Relocatable object files.
- Command input file, containing commands for processing.
- Relocatable library (RL) files to be searched or manipulated

The command input file is the standard file `$STDINX`. For sessions, `$STDINX` is the terminal. For batch jobs, `$STDINX` is the job file. You can redirect `$STDINX` to another file by using the `STDIN` option of the MPE/iX command `:RUN`. For example, to use the file named `SCRIPT` as the standard input file, enter the command

```
:RUN LINKEDIT.PUB.SYS;STDIN=SCRIPT
```

You can send one input command through the "INFO=*string*" parameter of the MPE/iX command `:RUN` instead of using `$STDINX`. This starts HP Link Editor/XL, executes the command you specified, and exits HP Link Editor/XL. Figure 4-2 shows the files used by HP Link Editor/XL.

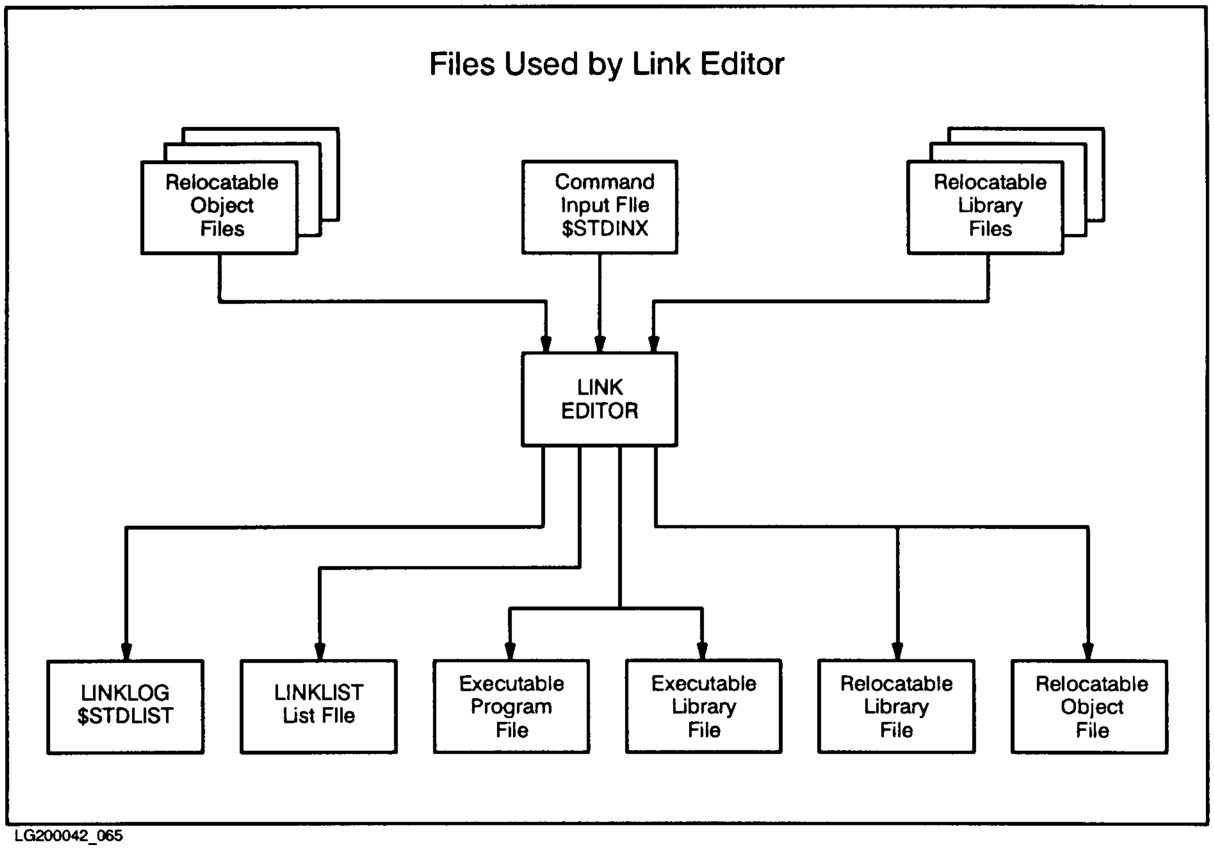


Figure 4-2. Files Used by HP Link Editor/XL

Note A relocatable object file produced by one of the MPE/iX compilers contains one relocatable object module.

Types of HP Link Editor/XL output files are:

- `$STDLIST`, which is the terminal during a session and the output spool file from a batch job, unless you redirect it, as discussed below.
- `LINKLIST`, which is a list file.
- An executable program file.
- An executable library file (XL).
- A relocatable library file (RL).
- A relocatable object file.

You can redirect $STDLIST to another device by using the STDLIST option of the MPE/iX command :RUN. For example, the following commands send HP Link Editor/XL output to the printer and runs HP Link Editor/XL:

```
        :FILE LINKOUT;DEV=LP
  :RUN LINKEDIT.PUB.SYS;STDLIST=*LINKOUT
```

LINKLIST information is sent to the $STDLIST device by default.

LINKLIST contains:

- Symbol map produced by the MAP option of the LINK command.
- Listing produced by the LISTPROG command.
- Listing produced by the LISTOBJ command.
- Listing produced by the LISTRL command.
- Listing produced by the MAP option of the ADDXL command.
- Listing produced by the LISTXL command.

HP Link Editor/XL creates a list file named LINKLIST for commands that generate large amounts of information. It usually writes the information to the $STDLIST device, but you can redirect LINKLIST to a different device by using the DEV option of the MPE/iX command :FILE. For example, the following commands send the listing of a relocatable library named LIBRL to a line printer:

```
         :FILE LINKLIST;DEV=LP
   :LINKEDIT
   :LinkEd> LISTRL RL=LIBRL
   :LinkEd> EXIT
```

HP Link Editor/XL Commands

HP Link Editor/XL commands that create and display executable program files and display symbols in a relocatable object file are:

- LINK, which creates an executable program file.
- LISTOBJ, which displays symbols in a relocatable object file.
- LISTPROG, which displays symbols in an executable program file.

Other HP Link Editor/XL commands manage relocatable and executable libraries. They allow you to add relocatable object modules to an RL or XL, purge modules from an RL or XL, copy modules between two RLs or two XLs. Some frequently used HP Link Editor/XL commands are:

- COPYRL, which copies modules from one relocatable library to another.
- EXTRACTRL, which extracts a relocatable object module or a group of object modules from a relocatable library.
- SHOWRL and SHOWXL, which display the name of the current (or working) library.
- LISTPROG, which displays a symbol listing on LINKLIST.

For detailed information, refer to *Link Editor/XL Reference Manual* (32650-90030).

Case Sensitivity

HP Link Editor/XL reads commands from the standard input file `$STDINX`. It is not case sensitive in commands or file names. However, entry point (procedure) names are case sensitive. For example, the `COPY`, `EXTRACT`, and `PURGE` commands allow you to specify an `ENTRY` name, which is case sensitive. This name must correspond, character for character, to the entry name in the symbol table, which indicates the name used in the relocatable object module. Usually, a case-insensitive language compiler converts a procedure name to lowercase. For relocatable object modules compiled in these languages, specify lowercase entry point names.

Keyword and Positional Parameters

Most HP Link Editor/XL commands have one or more keyword or positional parameters. These types of parameters are briefly discussed in Chapter 1, "Overview." For a complete discussion of keyword and positional parameters in HP Link Editor/XL commands, refer to *Link Editor/XL Reference Manual* (32650-90030).

Note When specifying positional parameters, use only one file name rather than a file name list, even if the command syntax permits a list. (For example, the `ADDRL` command syntax allows a file name list in the `FROM=` parameter.) To obtain a file name list, you must either specify keyword parameters or use an indirect file.

Using an Indirect File

An indirect file is an ASCII file containing a list of names. You can use indirect file names in HP Link Editor/XL commands instead of individually entering each name contained in the file. You can also mix indirect and regular file names in commands. Indirect files are a convenient way to enter a long list of names for commands that you use frequently. You can use indirect files only with the following commands in the specific parameters given in parentheses.

```
LINK        (FROM=, RL=, and XL=)
ADDRL       (FROM= and RL=)
COPYRL      (ENTRY=, MODULE=, and LSET=)
EXTRACTRL   (ENTRY=, MODULE=, and LSET=)
LISTRL      (ENTRY=, MODULE=, and LSET=)
PURGERL     (ENTRY=, MODULE=, and LSET=)
ADDXL       (FROM=, RL=, ENTRY=, MODULE=, and LSET=)
COPYXL      (ENTRY=, MODULE=, and LSET=)
LISTXL      (ENTRY=, MODULE=, and LSET=)
PURGEXL     (ENTRY=, MODULE=, and LSET=)
```

When you create an indirect file, enter one or more names on each line, using as many lines as necessary. Use a space or a comma to separate each name on a line. Make sure that HP Link Editor/XL has read access to the file.

To use an indirect file in a command, precede its name by a caret.

For example, if an ASCII file named `OBJLIST` contained the lines

```
        LIB1OBJ
        LIB2OBJ
        LIB3OBJ
        LIB4OBJ
        LIB5OBJ
```

you can use the indirect file `OBJLIST` in the following commands to add the five relocatable object files named in `OBJLIST` to the relocatable library named `LIBRL`:

```
        :LINKEDIT
   LinkEd> BUILDRL LIBRL
   LinkEd> ADDRL FROM=JLIST
   LinkEd> EXIT
```

Starting and Ending HP Link Editor/XL

You automatically use HP Link Editor/XL when you execute one of the MPE/iX commands that compiles and links a program in one step or compiles, links, and executes a program in one step. You can directly start HP Link Editor/XL in one of the following ways:

- Enter the `:LINKEDIT` command at the MPE/iX prompt:

```
             :LINKEDIT
```

HP Link Editor/XL displays its command line prompt `LinkEd>`, and waits for you to enter a command. Each time you enter a link editor command, it is executed and you are prompted to enter another. This continues until you end HP Link Editor/XL with the `EXIT` command.

- Enter a `:LINK` command at the MPE/iX prompt:

```
             :LINK FROM=EX1OBJ;TO=EX1PROG;RL=LIBRL
```

The link editor performs the link operation, then ends. The `:LINK` command is discussed in Chapter 3 and has the same syntax when used at the MPE/iX command level as when entered at the link editor prompt.

- Enter a `:RUN` or a `:LINKEDIT` command, with an `INFO` string, at the MPE/iX prompt. Enter an `INFO` string that contains one link editor command:

```
             :RUN LINKEDIT.PUB.SYS;INFO="LISTRL;RL=LIBRL"
```

Or you can use the short form:

```
             :LINKEDIT "LISTRL;RL=LIBRL"
```

The command in the `INFO` string is executed and HP Link Editor/XL ends. You can execute any link editor command in this manner.

There are three situations that terminate HP Link Editor/XL:

- When you explicitly end HP Link Editor/XL by entering the `EXIT` command:

```
LinkEd> EXIT
```

You can abbreviate the `EXIT` command as `E`, `EX`, or `EXI`. The commands `QUIT`, `Q`, and `BYE` also terminate HP Link Editor/XL.

- When end-of-file in `$STDINX` is encountered.
- When an error occurs in a batch job. An error message is printed, the system Job Control Word (JCW) is set to indicate a fatal error, and HP Link Editor/XL ends.

Creating an Executable Program File

HP Link Editor/XL creates executable program files from relocatable object files and relocatable libraries in the following way. First, it merges selected relocatable object files and libraries into one module and resolves inter-module references. Then, it searches selected relocatable libraries, resolving external references to symbols undefined after the merge operation. When a relocatable object module in the library resolves an external reference, the module is merged into the executable program file that is being built. In the last step, HP Link Editor/XL assigns virtual addresses to all symbols, binds references to the known symbols within each relocatable object module, and puts the resulting executable program in a form that the loader can process. Figure 4-3 shows the files HP Link Editor/XL uses when it creates an executable program file.

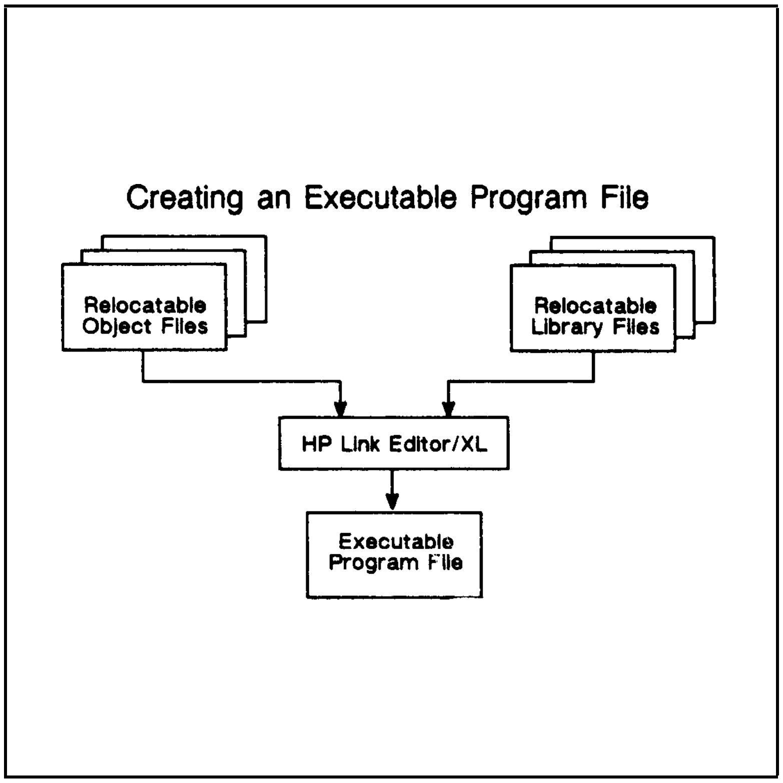


Figure 4-3. Creating an Executable Program File

An executable program file contains only one program entry point, which marks the instruction where execution begins. Each language defines its own program entry point. For example, HP FORTRAN 77/iX uses the main program as the entry point, while HP Pascal/iX uses a program's outer block as the entry point. When creating an executable program file, HP Link Editor/XL separates code and data areas, as shown in Figure 4-4.

The functions of the `LINK` command are:

- Binding
- Merging relocatable object modules
- Searching relocatable libraries (RLs)

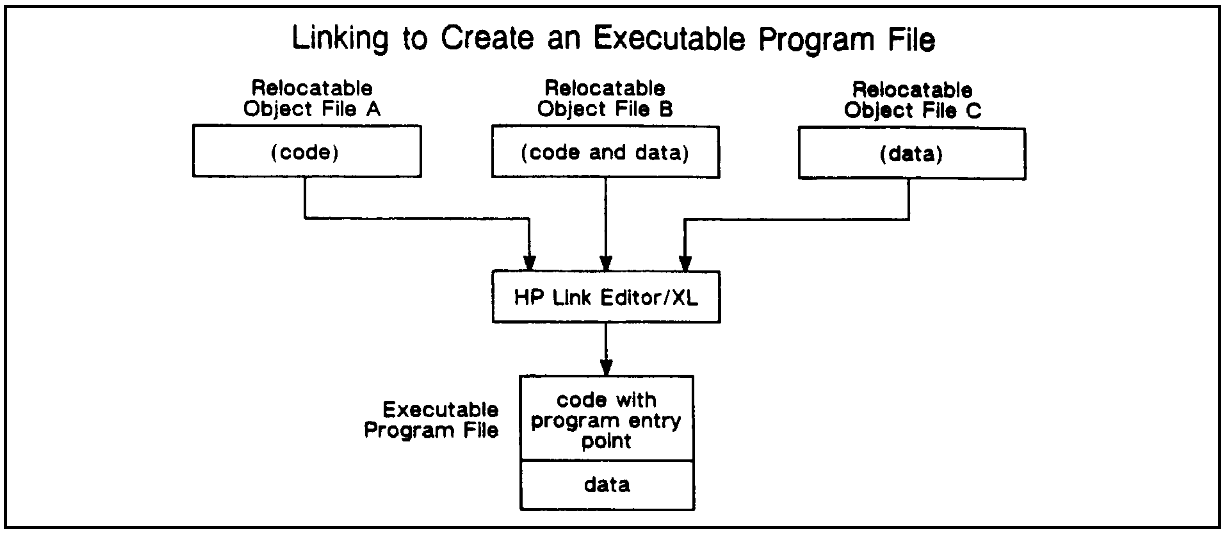

Linking to Create an Executable Program File
Relocatable Object File A
(code)
Relocatable Object File B
(code and data)
Relocatable Object File C
(data)
HP Link Editor/XL
Executable Program File
code with program entry point
data

Figure 4-4. Linking to Create an Executable Program File

Figure 4-5 shows HP Link Editor/XL executable program file commands along with the files that they use.

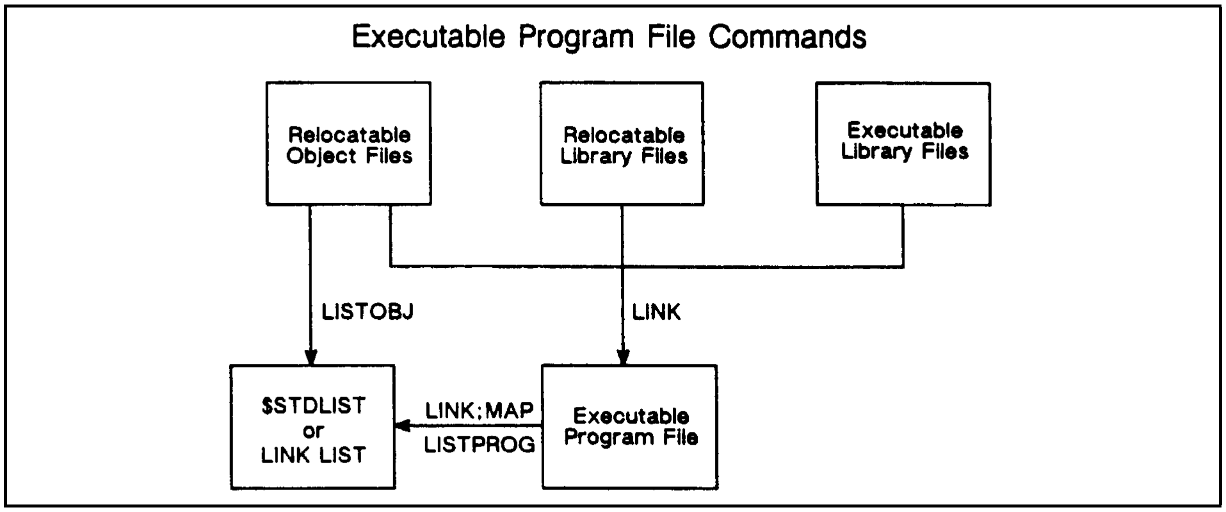


Figure 4-5. Executable Program File Commands

Comparison of Executable and Relocatable Libraries

Relocatable libraries (RLs) and executable libraries (XLs) share important characteristics. Both are:

- Created by programmers using HP Link Editor/XL commands.
- Contain routines necessary for program execution.
- Permit programs to share routines.

Their major differences are:

- An RL stores routines in relocatable form, and an XL stores them in executable form.
- An RL contains relocatable object code, which HP Link Editor/XL explicitly merges into an object module for each program that calls the routine. An XL contains sharable code, and each program refers to the same version of the code. When a program calls an XL routine, the loader reads it from disc into computer memory prior to execution.
- When HP Link Editor/XL merges object modules from an RL into a program file, all the modules become part of one executable object module. An RL routine can share global data with a program. An XL routine can have its own global data area, but executable object modules cannot share global data with the program or other executable modules.
- HP Link Editor/XL merges RL routines into a program file at link time. However, it only reserves space for pointers to XL routines in the External Reference Table at link time, and the loader resolves the references at run time.
- HP Link Editor/XL can search a series of RLs during the linking phase. The loader can search a series of XLs during the execution phase if you provide it with an XL list.

Using a Relocatable Library

MPE/iX relocatable libraries (RLs) give you the ability to use one or more libraries to resolve references at link time.

HP Link Editor/XL can build RLs from independent relocatable object modules and by copying relocatable object modules from one RL to another. It can also extract a copies of relocatable object modules from an RL and place them in a relocatable object file.

On 900 Series HP 3000 systems, an RL automatically expands until it reaches the maximum number of relocatable object modules it can contain. You can improve its structure and increase the size of its internal tables to allow for an additional 25 percent expansion of the library symbol table. If an RL reaches its relocatable object module limit, you can create a new library with a larger limit and copy the contents of the old library to the new one.

When HP Link Editor/XL resolves an external reference, it merges the entire relocatable object module containing that routine into the executable program file, even if only one procedure in that module is required. If the called procedure is one of several procedures in the relocatable object module, the entire module is added to the program file. To avoid adding unnecessary code due to including unreferenced procedures in an executable program file, keep the library routines in separate source files.

You can perform the following operations on RLs:

- Build new RLs.
- Add relocatable object modules to an existing RL.
- Copy selected relocatable object modules from one RL to another or to a relocatable object file.
- Purge selected relocatable object modules from an RL.
- List RL contents.

You can copy a relocatable object module from a relocatable object file into a relocatable library (RL). Creating a searchable RL requires using HP Link Editor/XL to create the RL and placing relocatable object files in it. For example, the commands

```
LINKEDIT
LinkEd> BUILDRL NEWRL
LinkEd> ADDRL FROM=LIB1EX1
LinkEd> EXIT
```

enter HP Link Editor/XL, build a new RL named `NEWRL`, the contents of a relocatable object file named `LIB1EX1` to it, and exit HP Link Editor/XL. For detailed information on manipulating an RL, refer to *Link Editor/XL Reference Manual* (32650-90030).

Using an Executable Library

You can use HP Link Editor/XL to create and maintain executable library files (XLs). XLs contain executable modules with the following characteristics:

- Are in a form that can be directly executed.
- Are shared, so that only one copy of the code need exist on the system. Programs that use an executable module share the same physical copy of the code.
- Have their own global data, separate from the program's global data.
- Have external references between executable modules and calling programs resolved at run time.
- Cannot have outer blocks and are, thus not independently executable.

HP Link Editor/XL creates an XL by linking relocatable object files or modules and placing the resulting modules in a library file. To be functional, an XL must be loaded with an executable program file containing an outer block and an entry point.

MPE/iX executable libraries (XLs) have the following characteristics:

- They can have any valid MPE/iX file name.
- Many can be accessed in one command because you can specify several in the `XL=` parameter of `:LINK` or `:RUN`.

Creating an XL is a process similar to creating a relocatable library (RL); build an empty XL and add modules to it. For example, the commands

```
        :LINKEDIT
LinkEd> BUILDXL MYXL
LinkEd> ADDXL FROM=MYOBJ
LinkEd> EXIT
```

enter HP Link Editor/XL, build a new XL named `MYXL`, add the contents of a relocatable object file named `MYOBJ` to it, and exit HP Link Editor/XL.

For detailed information on manipulating an XL, refer to *Link Editor/XL Reference Manual* (32650-90030).

HP Link Editor/XL automatically pre-links relocatable object modules as you add them to an XL. XLs are not searched at link time, but you can specify a list of them for the loader to search when the program is run. MPE/iX system libraries are XLs. The loader automatically searches system libraries after any libraries you specify at load time.

5

Optimizing a Program

The Optimizer is a program that is an integrated part of an MPE/iX compiler. It is available to use with HP C/iX, HP COBOL II/XL, HP FORTRAN 77/iX, and HP Pascal/iX. When you compile a program, you can select an optimizer option that will use the MPE/iX Optimizer to improve the code.

This chapter provides an overview of:

- Techniques a programmer can use to program for best optimization.
- How Optimizer aids in producing improved code.

For detailed information on how each compiler optimizes and actions necessary when an optimized program fails, refer to the following manuals in the Language Series:

- *HP C/iX Reference Manual Supplement* (31506-90001).
- *HP COBOL II/XL Programmer's Guide* (31500-90002).
- *HP FORTRAN 77/iX Programmer's Guide Supplement* (31501-90002).
- *HP Pascal/iX Programmer's Guide* (31502-90002).

Note Symbolic debugging is not available when the optimizer option is selected.

The MPE/iX Optimizer can help a program take full advantage of the 900 Series HP Precision Architecture (HP-PA) in the following ways:

- The small instruction set and pipeline architecture allows concurrent execution of multiple instructions. Thus, improved program execution speed can be achieved by using careful instruction scheduling.
- Many registers are available in the CPU for fast memory referencing. The Optimizer may promote frequently used variables to reside in registers instead of in memory. On the 900 Series, you can optimize at a level lower than that which is expedient with most commercial optimizers: at compile time, at the machine instruction level, after instruction selection.
- The Optimizer may move loop-invariant instructions out of program loops, eliminate redundant calculations, and eliminate unnecessary memory references.

Efficient instruction scheduling avoids hardware interlocks and eliminates branch delay slots, fully utilizing fast instruction cycle time of the 900 Series. Hardware interlock occurs when two pipelined instructions both require the same resource or when one instruction requires a result of a previous instruction that has not completed. With a graph of program flow to reveal instruction dependencies, instruction scheduling can attempt to avoid hardware interlocks.

On the 900 Series, a branch does not take effect until the second machine cycle after the branch instruction. The instruction immediately after a branch instruction is called the delay slot. Even when you do not select optimization, the compiler tries to schedule a useful instruction in the delay slot. However, the MPE/iX Optimizer does a better job of branch delay scheduling and may be able to completely eliminate some branches.

Optimizer Levels

The levels of code optimization that the MPE/iX Optimizer provides are Level 0, Level 1, and Level 2.

Level 0 optimization is the default optimization level and provides simple optimizations that minimize compile time. Use this level when debugging a program or running a program that will not be run many times (for example, a student job program that is run a few times and discarded). Level 0 functions include:

- Simple branch delay slot scheduling.
- Dead code elimination.
- Faster register allocation (including copy elimination).

Level 1 optimization is local to basic blocks, but does not optimize globally. Use Level 1 to achieve some optimization without spending excessive time compiling. Level 1 functions include:

- Branch optimization.
- Dead code elimination.
- Faster register allocation (including copy elimination).
- Instruction scheduling.
- Peephole optimization.

Level 2 optimization is global optimization. It provides the greatest saving of space and time achievable with the Optimizer and produces the most compact and fastest running program of all the levels. Use Level 2 when running a debugged program a large number of times. This is the correct level to choose for most applications. Level 2 functions are:

- All Level 1 optimizations.
- Coloring register allocation.
- Induction variable elaboration and strength reduction. (An induction variable is a variable dependent on the value of a loop counter.)
- Common subexpression elimination.
- Constant folding.
- Loop invariant code motion.
- Unused definition elimination.
- Promotion of variables to registers.

Use of MPE/iX Optimizer with Languages

In MPE/iX Native Mode, HP C/iX, HP COBOL II/XL, HP FORTRAN 77/iX, and HP Pascal/iX provide the optimizer option. Currently, HP COBOL II/XL provides only Levels 0 and 1. HP C/iX, HP Pascal/iX, and HP FORTRAN 77/iX provide Levels 0, 1, and 2.

Optimizer Assumptions

During compilation, a compiler gathers information about the use of variables and passes it to the Optimizer. The MPE/iX Optimizer uses the information to ensure that each code transformation it performs maintains the correctness of the program (at least to the extent that the original unoptimized program is correct).

The compiler assumes that inside a subroutine or function, only the following variables can be accessed directly, indirectly, or by another function call:

- Common variables declared in this routine.
- Local variables (static and dynamic).
- Parameters to this routine.
- Global variables visible in this routine.

If you have code that violates these assumptions, Optimizer can change the behavior of the program in an undesirable way. Avoid the following coding practices to ensure correct program execution in optimized code:

- Referencing outside the described bounds of an array. This can cause address corruption or cause the program to abort when it is run.
- Using variables that can be accessed by a process other than the program, such as shared common variables. The compiler assumes that the program is the only process accessing the data. (HP FORTRAN 77/iX has some exceptions.) HP C/iX, HP Pascal/iX, and HP FORTRAN 77/iX provide compiler options to change the assumptions about a routine.
- Avoid using variables before they have been initialized. The optimized version of a program may run differently than the unoptimized version.

For detailed information, refer to *HP FORTRAN 77/iX Programmer's Guide Supplement* (31501-90002) and *HP Pascal/iX Programmer's Guide* (31502-90002).

Coding for Performance and Optimization

The MPE/iX Optimizer modifies code to use machine resources efficiently, using less space and running faster. It improves code, but does not alter the algorithm used in code. Coding for good performance requires recognition of the following facts:

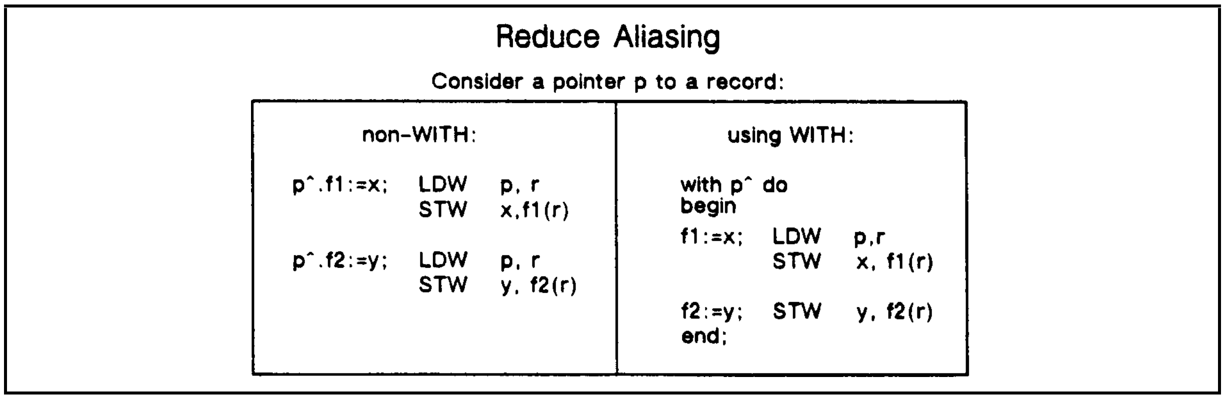
- Coding practices alter performance.
- Mismatches exist between programming languages and most architectures.
- Coding techniques can enhance or inhibit optimization opportunities.

Coding for optimization provides the following methods of optimizing a program at the source code level:

- Reduces or avoids aliasing
- Uses optimal data types
- Identifies common subexpressions
- Reduces procedure calls
- Avoids non-native alignment

Reduce Aliasing

The compiler must generate explicit loads and stores when programs use aliasing with pointers. If you specify that the pointer does not change (for example, by using `WITH` in HP Pascal/iX), you can eliminate some loads and stores. Thus, more optimization (such as using registers) can occur. Figure 5-1 shows an example of reducing aliasing.



Reduce Aliasing

Consider a pointer p to a record:

non-WITH:	using WITH:

```
p^.f1:=x;  LDW   p, r
           STW   x,f1(r)

p^.f2:=y;  LDW   p, r
           STW   y, f2(r)
```
	```
	with p^ do
	begin
	f1:=x;  LDW   p,r
	        STW   x, f1(r)

	f2:=y;  STW   y, f2(r)
	end;
	```

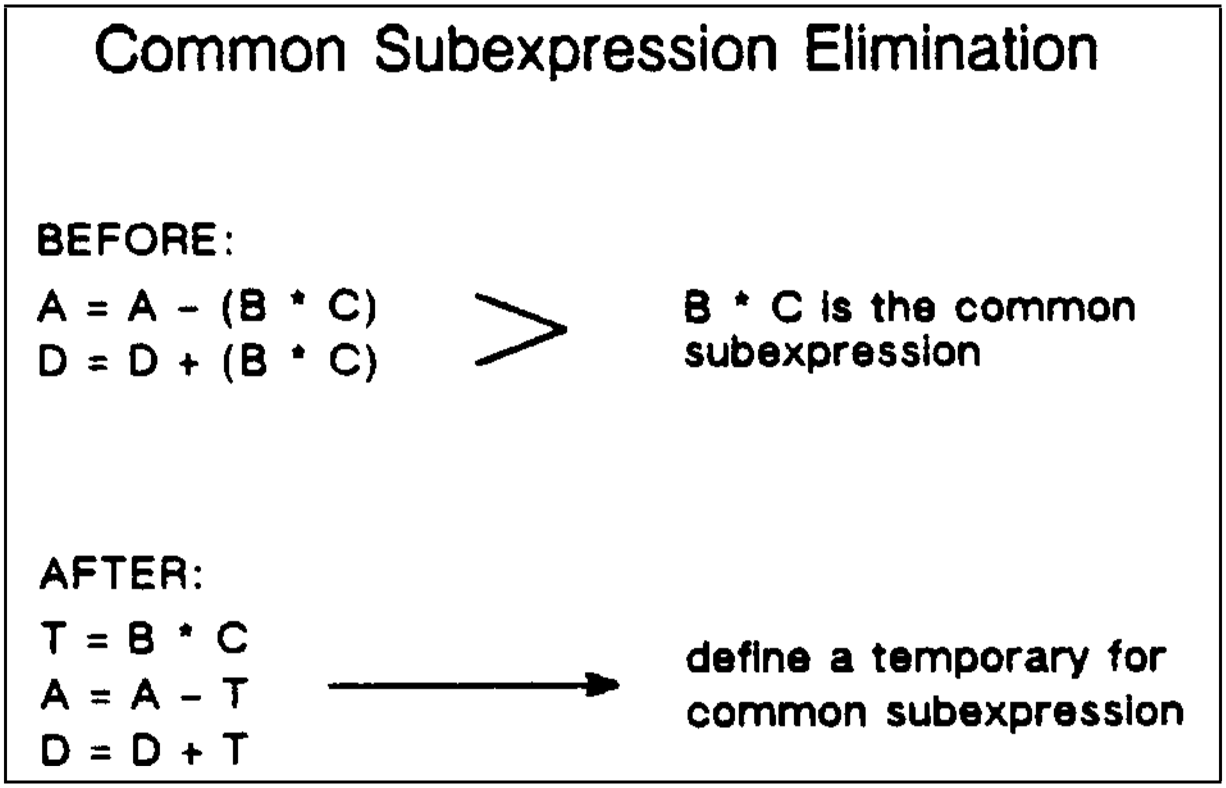
Figure 5-1. Reducing Aliasing

Use Optimal Data Types

Examples of optimal data types are 8-bit character and 32-bit integer. For detailed information refer to the appropriate manual in the Language Series.

Eliminate Common Subexpressions

You can improve the performance of optimized and unoptimized code by using programmer identification of common subexpressions. Figure 5-2 shows an example of common subexpression elimination.



Common Subexpression Elimination

BEFORE:

```
A = A - (B * C)
D = D + (B * C)
```

$>$ B * C is the common subexpression

AFTER:

```
T = B * C
A = A - T
D = D + T
```

$\longrightarrow$ define a temporary for common subexpression

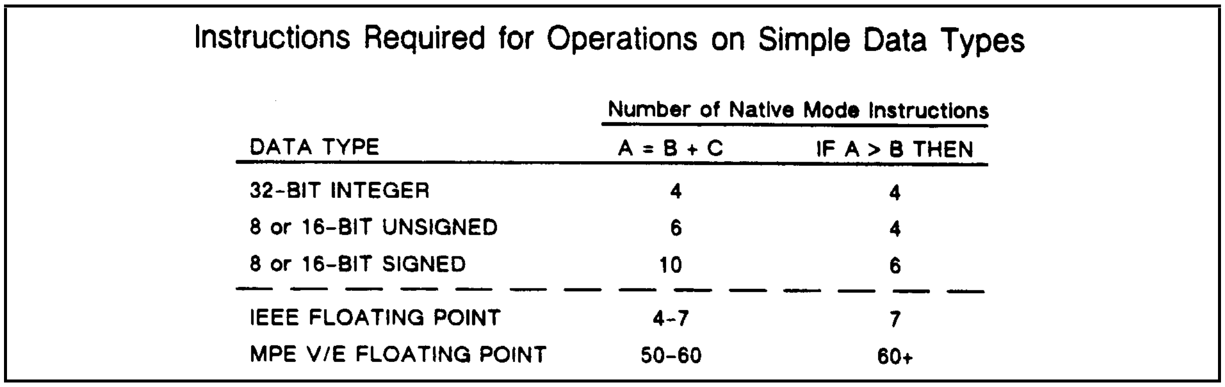
Figure 5-2. Eliminating Common Subexpressions

Instructions Required for Operations on Simple Data Types

Comparison of floating point instructions and integer instructions is not valid because the floating point instructions may execute in a different cycle from the integer instructions and may require synchronization. The instructions that compute floating-point arithmetic are done in a coprocessor and do not execute in a single machine cycle. However, the instructions that compute integer arithmetic are done in a CPU and complete in a single machine cycle. Figure 5-3 shows examples of the instructions for operations on simple data types.

These examples assume the following characteristics are true:

- Unoptimized code
- Well aligned operands
- No range or overflow checking



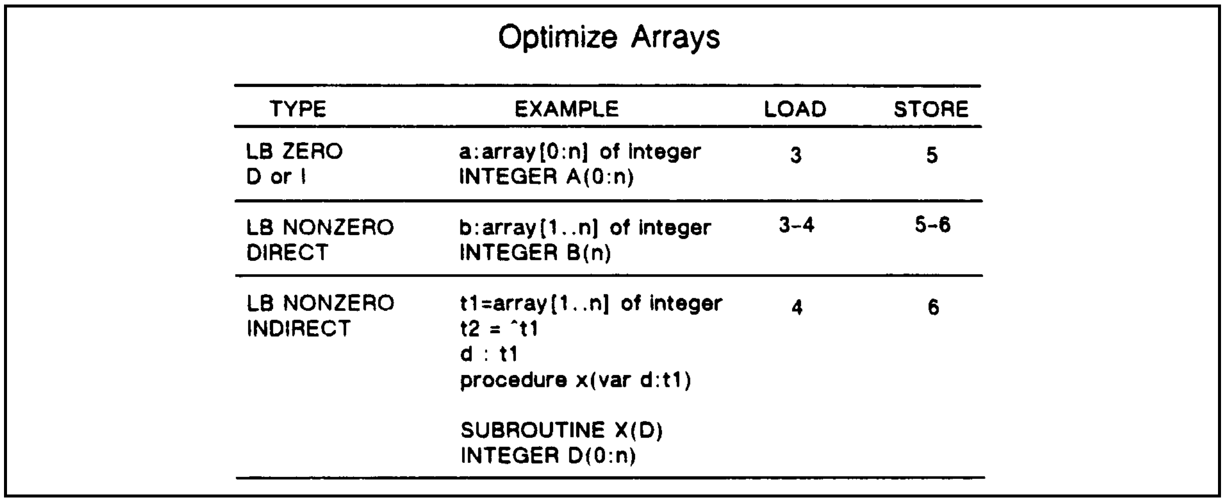
Instructions Required for Operations on Simple Data Types

	Number of Native Mode Instructions	
DATA TYPE	A = B + C	IF A > B THEN
32-BIT INTEGER	4	4
8 or 16-BIT UNSIGNED	6	4
8 or 16-BIT SIGNED	10	6
IEEE FLOATING POINT	4-7	7
MPE V/E FLOATING POINT	50-60	60+

Figure 5-3. Instructions Operations on Simple Data Types

Optimize Arrays

The compiler does not always initialize array elements when an array is created. Ensure that all variables are properly initialized. Uninitialized variables that did not cause problems on MPE V/E-based systems may cause programs to abort on MPE/iX-based systems. Figure 5-4 shows examples of array optimization.



Optimize Arrays

TYPE	EXAMPLE	LOAD	STORE
LB ZERO D or I	a:array[0:n] of integer INTEGER A(0:n)	3	5
LB NONZERO DIRECT	b:array[1..n] of integer INTEGER B(n)	3-4	5-6
LB NONZERO INDIRECT	t1=array[1..n] of integer t2 = ^t1 d : t1 procedure x(var d:t1) SUBROUTINE X(D) INTEGER D(0:n)	4	6

Figure 5-4. Optimizing Arrays

Reduce Procedure Calls

Procedure calls in code limit optimization because some registers cannot be kept live (retain values) across calls. You can remove calls to user procedures from the main body of code and, instead, branch to a common area. Figure 5-5 shows an example of reducing procedure calls.

Reduce Procedure Calls

```
            BEFORE                          AFTER
 IF (error1) THEN               IF (error1) THEN
    CALL REPORT_ERR (1)            IERR = 1
    GO TO 100                      GO TO 100
    ENDIF                          ENDIF

 IF (errorN) THEN               IF (errorN) THEN
    CALL REPORT_ERR (N)            IERR = N
    GO TO 100                      GO TO 100
    ENDIF                          ENDIF

100 END                            GO TO 101

                               100 CALL REPORT_ERR(IERR)
                               101 END
```

Figure 5-5. Reducing Procedure Calls

Expand Small Procedures In-line

Expanding a small procedure in-line increases the scope for the optimizer, reduces procedure call overhead, and allows additional constant folding of constant value parameters. You should expand procedures shorter than five lines. Figure 5-6 shows an example of expanding a procedure in FORTRAN 77/iX. In HP Pascal/iX, the you can use the compiler option `$OPTION INLINE$` to expand the code for a routine in-line at the point of call.

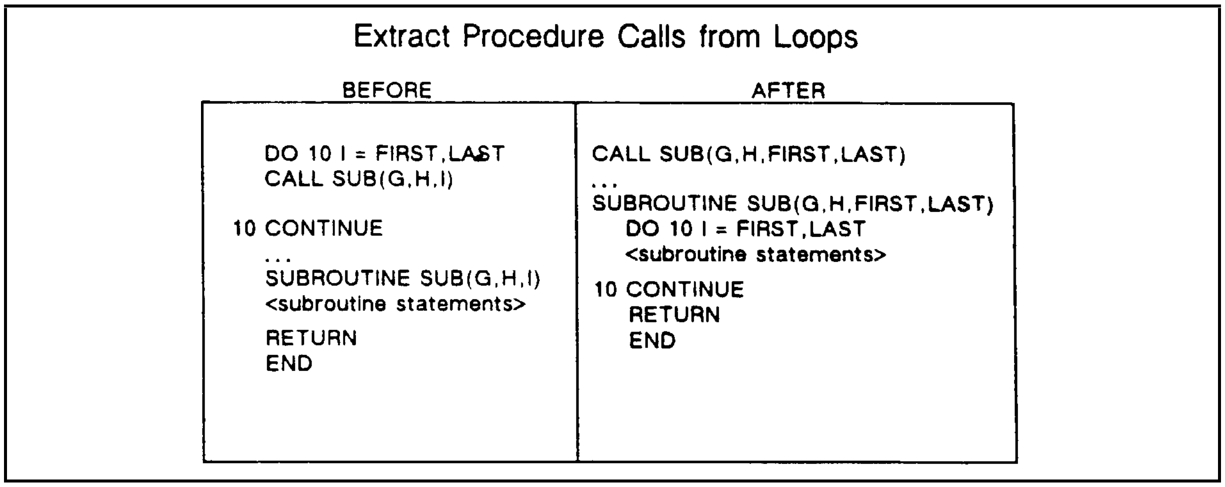
Expand Small Procedures In-Line

```
            BEFORE                          AFTER
    ...                               ...
 10 CALL SUB(A,B,C)                10 A = (B**2) + (C**2)
                                      B = B + 1
    ...                               C = C + 1
 20 CALL SUB(X,Y,Z)
                                      ...
    ...                            20 X = (Y**2) + (Z**2)
    END                               Y = Y + 1
    SUBROUTINE SUB(Q,R,S)             Z = Z + 1
       Q = (R**2) + (S**2)
       R = R + 1
       S = S + 1
       RETURN
    END
```

Figure 5-6. Expanding Small Procedures In-line

Extract Procedure Calls from Loops

It is inefficient to code a loop containing only a procedure call because of the required overhead by each procedure call. It is a recommended programming practice to code the loop inside the procedure. Figure 5-7 shows an example of extracting a procedure call from a loop.



Extract Procedure Calls from Loops

BEFORE

```
      DO 10 I = FIRST,LAST
      CALL SUB(G,H,I)
   10 CONTINUE
      ...
      SUBROUTINE SUB(G,H,I)
      <subroutine statements>
      RETURN
      END
```

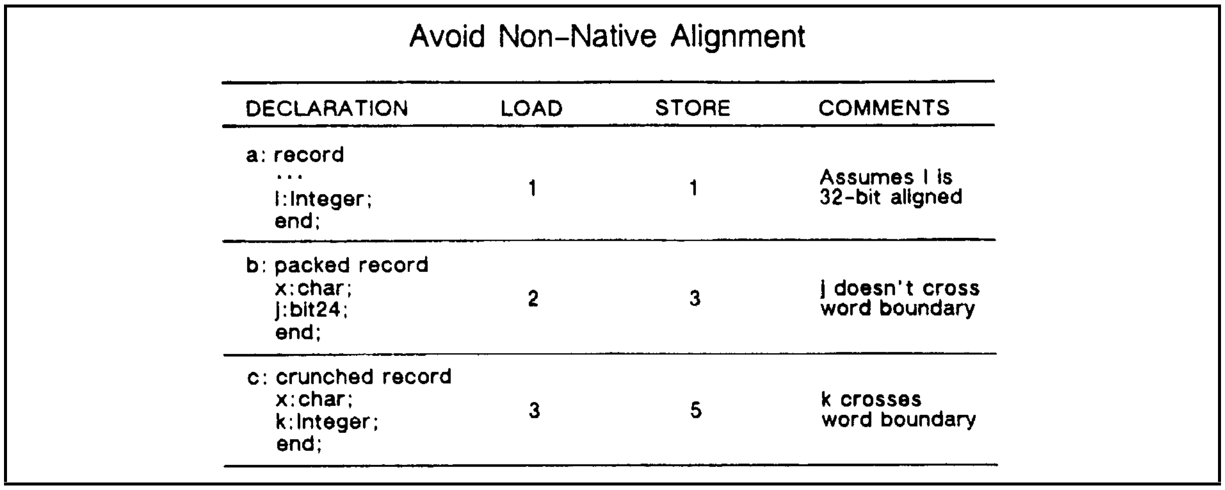
AFTER

```
      CALL SUB(G,H,FIRST,LAST)
      ...
      SUBROUTINE SUB(G,H,FIRST,LAST)
         DO 10 I = FIRST,LAST
         <subroutine statements>
   10 CONTINUE
         RETURN
         END
```

Figure 5-7. Extracting Calls from Loops

Avoid Non-native Alignment

Figure 5-8 shows an example of avoiding non-native alignment.



Avoid Non-Native Alignment

DECLARATION	LOAD	STORE	COMMENTS

a: record
...
i:integer;
end;
	1	1	Assumes i is 32-bit aligned

b: packed record
x:char;
j:bit24;
end;
	2	3	j doesn't cross word boundary

c: crunched record
x:char;
k:integer;
end;
	3	5	k crosses word boundary

Figure 5-8. Avoiding Non-native Alignment

Optimize HP COBOL II/XL Data Types

Optimizing HP COBOL II/XL data types requires recognition of the following considerations:

- 32-bit binary integers are desirable. `PIC S9(9) COMP SYNC` specification is optimal.
- Relying on default specifications is undesirable. You must specify `COMP` to get a binary integer. Otherwise, it defaults to a decimal integer. You must specify `SYNC` to guarantee word alignment.
- Decimal validation adds overhead.
- Signed numeric fields are preferable to unsigned numeric fields.

Optimize HP COBOL II/XL Data Types

Optimizing HP COBOL II/XL data types requires recognition of the following considerations:

- 32-bit binary integers are desirable. `PIC S9(9) COMP SYNC` specification is optimal.
- Relying on default specifications is undesirable. You must specify `COMP` to get a binary integer. Otherwise, it defaults to a decimal integer. You must specify `SYNC` to guarantee word alignment.
- Decimal validation adds overhead.
- Signed numeric fields are preferable to unsigned numeric fields.

6

File System

The File System is the part of the MPE/iX operating system that manages data access on the 900 Series of the HP 3000 family of computers. The MPE/iX I/O System transfers data between the File System and physical devices (for example, printers and tape drives). Figure 6-1 shows the relationships of a program, the MPE/iX File System, the MPE/iX I/O System and the system hardware. The File System is the interface between a program and the rest of the system.

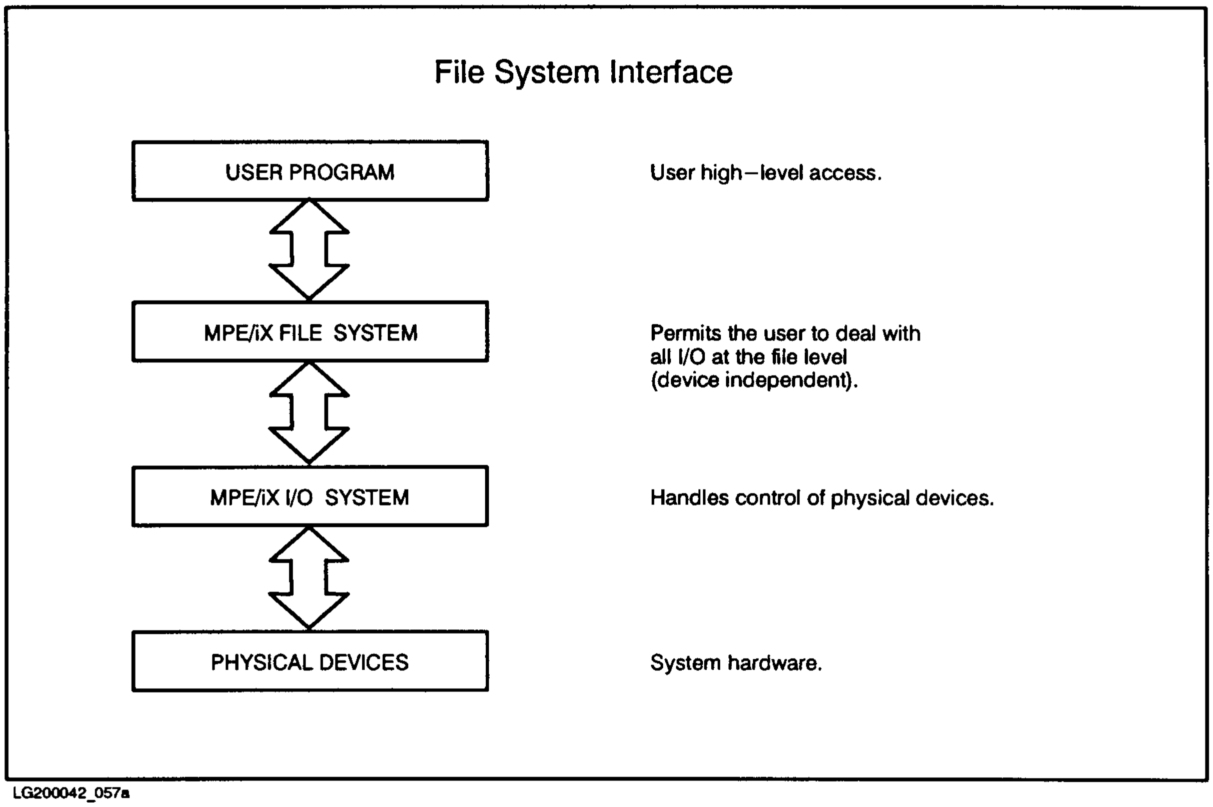


Figure 6-1. File System Interface

Records and Files in the File System

The File System manages data being transferred or stored with peripheral devices. It handles I/O operations, such as passing information to and from user processes, compilers, and information management subsystems. Conceptually, information for data transfers is arranged as elements of data in a record. The record is input, processed, and output as a single unit. Logically related records are grouped into sets called files, which can be kept in any storage medium or sent to any I/O peripheral, as shown in Figure 6-2. In some cases, records may be physically grouped together in blocks when they are in a file residing on a non-disc device.

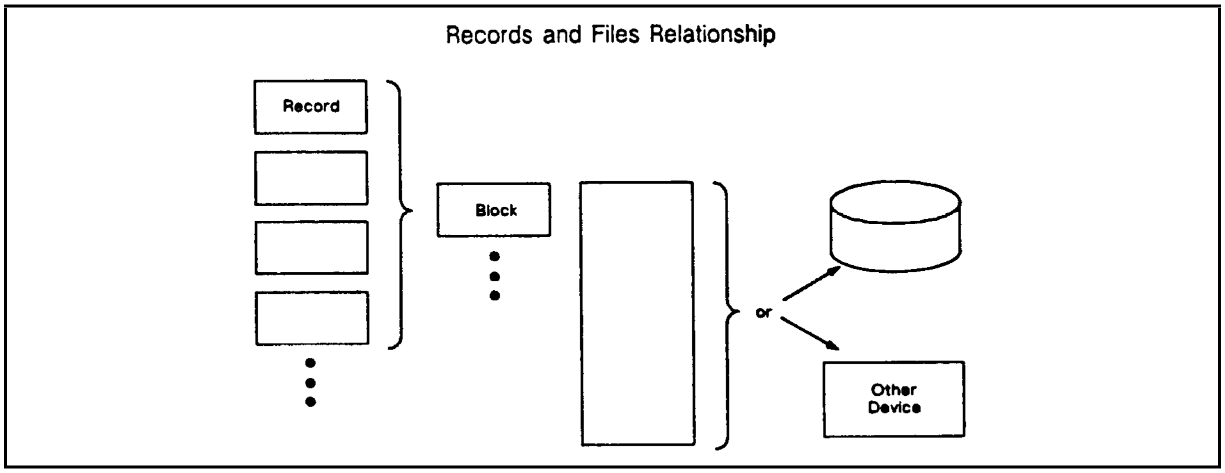


Figure 6-2. Records and Files Relationship

Since all I/O operations are done through the mechanism of files, you can access different devices in a standard, consistent way. The names assigned to a file when it is defined in a program do not restrict that file to residing on the same device every time the program is run; the program is device independent.

The File System recognizes two basic types of files, classified by the media on which they reside when processed: disc files and device files.

For detailed information on the File System interface, disc files, device files, and spooled device files, refer to *Accessing Files Programmer's Guide* (32650-90017).

Device Files

Device files are files currently being input to or output from any peripheral device, except a system or private domain disc volume. A foreign disc or serial disc file is considered to be a device file. A device file is nonsharable; it is accessed exclusively by the job or session that acquires it and is owned by that job or session until the job or session explicitly releases it or terminates.

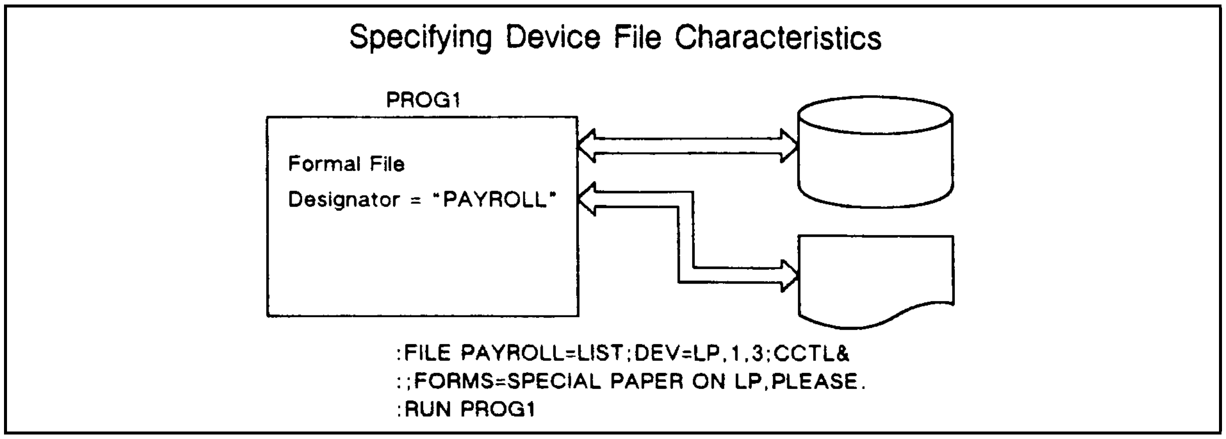
A device file is a set of logically related records transferred to or from a non-random disc device such as a terminal, magnetic tape, or line printer. A device file is built differently from a disc file. You define the physical and operational characteristics each time you access the device. However, the File System is used to open, access, and close a device file, just as it does a disc file. When you have a program that is written to use a disc file and you want it to use a device file at run time, use the `DEV=` parameter of the `:FILE` command to specify a device class (such as `TAPE`) or a specific logical device number that corresponds to a device on your system. If you do not use the `DEV=` parameter, the `:FILE` command defaults to defining a disc file. Magnetic tape has no mandatory settings. You can use the `REC=` parameter of the `:FILE` command to specify physical characteristics.

The File System `HPFOPEN` and `FOPEN` intrinsics automatically set actual device characteristics that override any specifications passed from the program or the `:FILE` command. For example, the actual device characteristics of a line printer are always `NEW`, `ASCII`, `UNDEFINED`, and `WRITE ONLY` access.

If a device file characteristic is not specified through one of the following, then it has a default value:

- Use of device characteristics.
- `:FILE` command.
- The way in which a program opens the file.

Figure 6-3 shows an example of specifying device file characteristics in a `:FILE` command. In this figure, the device specified by `:DEV=` for the file named `PAYROLL` is a line printer; the output priority of the spoolfile is 1; and the number of copies of output to print is 3. The `;FORMS=` message is sent to the console when the file is ready to be output to a device.



```
:FILE PAYROLL=LIST;DEV=LP,1,3;CCTL&
:;FORMS=SPECIAL PAPER ON LP,PLEASE.
:RUN PROG1
```

Figure 6-3. Specifying Device File Characteristics

The most common medium for storage of a device file is magnetic tape. The File System has several capabilities in handling magnetic tape labels, and the Labeled Tape Facility provides for the handling of multireel tape sets. For detailed information on magnetic tape labels, refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364) and *Accessing Files Programmer's Guide* (32650-90017).

Disc Files

A disc file is a set of logically related records that reside on disc. A disc file is immediately accessible by the system and is potentially sharable by several jobs or sessions at a time. The structure of a disc file is determined by the physical constraints of the disc, such as available space, and user-defined elements of the file. You use the File System to build, open, access, and close a disc file.

A disc file resides on one or more volumes. A file does not have to reside as a contiguous piece on one volume, but can be broken into many extents distributed on different volumes.

The File System must store data on disc in an efficient manner, since files are constantly manipulated: added, deleted, and updated. It uses demand paged virtual memory to manage virtual memory and the user interface to files. This occurs transparently from the user's perspective. Page-sized portions of files are swapped into memory only when needed.

The user-defined elements of a disc file are those within your control. Elements you can specify include:

- Which disc (or discs) the file will reside on.
- Record type, size, and access.

For detailed information on record types and elements you can specify, refer to the "Record Structure" subsection and *Accessing Files Programmer's Guide* (32650-90017).

Disc File Storage

The File System physically stores a file on disc by breaking it into extents and maintaining pointers to where each extent resides. It logically stores a file based on where the file reference indicates the file belongs in the directory structure. Both concepts are discussed in Chapter 1, "Overview."

The group specified for a file in the standard file reference format (filename.groupname.acctname) determines the volume set on which the file resides. You can select the volume set, volume name, and class for a file by using the `HPFOPEN` intrinsic.

Most common uses of files do not require specification of volume sets or classes, but it is sometimes useful to restrict a file to a particular volume or group of volumes. If a file resides on only one volume, then damage to other discs will not destroy the file. However, when a volume specification is given, the File System distributes extents throughout a volume set to prevent one physical disc from running out of space.

Disc storage is logically partitioned into:

- Volume, which is a physical disc and a member of a volume set. It may also be a member of one or more volume classes. It is removable while the system is running, is automatically recognized and mounted by the system, and the operator is notified when an unmounted volume is requested.

Free space is tracked by a separate free space bit map on each individual disc volume. To find space to put a new disc file, several tables may be checked. You can display disc space information by running `DISCFREE.PUB.SYS`.

- Volume set, which consists of a maximum of 255 volumes or volume classes. It provides a way of partitioning user data into removable entities and are used only for permanent storage space.

The directory structure is spread across a volume set. Each volume has a label object, which contains the file labels for files. The directory has no arbitrary limit to the number of files per volume. The system disc is usually configured to be `LDEV 1`.

- Volume class, which can contain a maximum of 255 volumes and is optional within a volume set. You cannot mount a disc by a volume class designation; it is used exclusively to allocate disc storage space.

You can specify disc volume restrictions in the following ways:

- Use the *volume name option* or the *volume class option* of `HPFOPEN` or the *device* parameter of `FOPEN` to specify either a volume name or a volume class name. This restricts the placement of your file s extents to either the specified volume or the specified volume class within the volume set, thus facilitating file portability.
- Using `FOPEN`, you can restrict the placement of a file extent to a specified volume or a specified volume class within the volume set assigned to the group in which the file will be created.
- Using `FOPEN`, you can specify a volume name or a volume class in a way that maintains FOPEN compatibility with MPE V/E-based systems. For example, MPE/iX translates a logical device number (`LDEV`) passed to `FOPEN` into the volume name currently mounted on the disc drive and places the volume name in the file label.

For more information on specifying disc volume restrictions, refer to *Accessing Files Programmer's Guide* (32650-90017).

File Directory Structure

File directory structure is based on several file characteristics that determine how the file is classified and how it is handled during file manipulation operations. These characteristics are discussed in the following subsections.

Domains

Files can be classified on the basis of domain. The domain of a file determines if it is permanent, is temporary (lasts only for the duration of the job or session), or exists only for one particular process.

The File System maintains separate directories to record the location of `PERMANENT` and `TEMP` files. `PERMANENT` files are recorded in the System File Directory; and `TEMP` files, in the Job Temporary File Directory. A permanent or temporary file can be opened or closed with a file domain of `OLD` (although you cannot create a file with an `OLD` domain by using the MPE/iX command `:BUILD`).

This designation differs from `PERMANENT` only in the fact that first the Job Temporary File Directory and then the System Permanent File Directory are searched for the filename. `OLD` files are described in the "Closing a File" subsection. There is no File System directory for files that exist only to their creating process (`NEW` files).

The subsections below define domains and describe how to change, list, and search for the domain of a file and how to open and close files with various domains. Table 6-1 summarizes the features of `NEW`, `TEMP`, and `PERMANENT` files.

Table 6-1. New, Temporary, and Permanent File Features

NEW Files	TEMPORARY Files	PERMANENT Files
Exists only to creating process	Exists as job temporary file	Exists as permanent file in File System
Space not yet allocated	Some or all space already allocated	Some or all space already allocated
Physical characteristics not previously defined	Physical characteristics defined	Physical characteristics defined
Known only to creating session or job	Known only to creating session or job	Known system wide
Exists only for duration of program execution	Exists only for duration of creating session or job	Permanent

In some cases, the domain you specify for a file may be restricted by the type of device on which the file resides. The domains permitted are summarized in Table 6-2.

Table 6-2. Valid File Domains

Device Type	Valid Domains (see note below table)
Disc	`NEW, TEMPORARY, PERMANENT, or OLD`
Magnetic tape device	`NEW, PERMANENT, or OLD`
Terminal	`NEW, PERMANENT, or OLD`
Line printer	`NEW`
Plotter	`NEW`

Note When you specify a file domain using `HPFOPEN`, you should open only disc files with the *domain option* set to `NEW`. Device files can be opened with the *domain option* set to `NEW` (to maintain compatibility with MPE V/E), but a warning is returned in the *status* parameter.

NEW Files

When you create a file, you can indicate to the File System that it is a `NEW` file; it did not previously exist. Space for this file has not yet been allocated. As a `NEW` file, only the program that creates it knows about it, and it exists only while the program is executing. When the program concludes, the file vanishes, unless you take action to retain it.

TEMP Files

A `TEMP` file is one that already exists, but only the job or session that created it knows about it. Some or all of the space for the file has been allocated, and its physical characteristics have been defined. A file in this domain is a job temporary file; it has been created for a specific purpose by its job or session. It may not be needed after the job or session ends. When the job or session concludes, the file automatically disappears, unless you take action to change it to a `PERMANENT` file. This is described in the "Changing Domains" subsection, below.

PERMANENT Files

A `PERMANENT` file exists as a permanent file in the File System. Its existence is not limited to the duration of its creating job or session. Any job or session can access the file when security restrictions allow. Some or all of the space for the file has been allocated, and its physical characteristics have been defined. When the job or session concludes, the file remains.

Effect of File Domain on Operations

You can select or change the domain disposition of a file when you close the file using the `FCLOSE` intrinsic. This is called the closing disposition of the file and follows these rules:

- Any file can be deleted when closed. This is the default for a `NEW` file.
- A `NEW` file can be changed to a `TEMP` file when closed.
- A `NEW` or `TEMP` file can be changed to `PERMANENT` when closed.
- A `PERMANENT` file can be changed to a `TEMP` when closed by a PM (privileged mode) user.

The File System action defaults to delete a `NEW` file, temporarily keep a `TEMP` file, and save a `PERMANENT` file. The way in which subsystems use files affects these defaults. You can specify a closing disposition with a `:FILE` command that overrides how the program or subsystem closes the file. Other commands such as `:PURGE`, `:BUILD`, and `:SAVE` provide various mechanisms for changing file domain. For detailed information on these commands, refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364).

The `FCLOSE` intrinsic is called for every file on which an `HPFOPEN` or `FOPEN` has been performed. The File System ensures that an opened file is automatically closed at program termination, without requiring an explicit statement. However, a subsystem must explicitly call `FCLOSE` to ensure that the action taken will be as specified in your program.

Determining file disposition at close time can have consequences on running programs (especially in batch mode). For example, a program can open a `NEW` file by default, write data to the file, and attempt to save the file as OLD. A directory error (for example, a duplicate file name) can be detected only at close time; and data is lost, unless you have used one of the following methods to check for or avoid the situation:

- Create a NEW file by using BUILD before executing the program in which the file is opened as an OLD file.
- Delete the previous copy of the file before the run.
- Rename the existing file by using FRENAME.

The File System closes files when it terminates a program, without requiring the program to explicitly close them. This allows no recourse for accidentally losing data. It is a recommended programming practice, when the content of a file is important, to write the program to properly handle closing disposition.

The CREATE domain of HPFOPEN creates a NEW file and makes it PERMANENT when closed. It eliminates the need to HPFOPEN or FOPEN it as a NEW file, HPFCLOSE or FCLOSE it as a PERMANENT file, and open it with HPFOPEN or FOPEN again as an OLD file.

Changing Domains

A file's domain can be changed. Any disc file can be made permanent or can be deleted after it has served its purpose. You can use the disposition parameter of the FCLOSE intrinsic to specify a different domain for a file as it closes. You can use the :FILE command to change the domain of a file; specifying the DEL, TEMP, or SAVE parameter determines the disposition of a file when it is closed after its next use. The DEL parameter deletes a file after its next use. The TEMP parameter changes the domain of a NEW file to TEMP when it is closed. The SAVE parameter changes the domain of a NEW or TEMP file to PERMANENT when it is closed.

You can change the domain of a file from TEMP to PERMANENT without opening and closing the file by using the :SAVE command. This command prompts you for the lockword if the file has one.

For examples of changing domains, refer to *Accessing Files Programmer's Guide* (32650-90017). For detailed information on the :FILE and :SAVE commands, refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364).

File Specifications

File specifications allow you to identify a file. This is necessary in order to:

- Locate it in the directory structure
- Refer to it from a command or program
- Determine how to handle it during file manipulations such as opening and closing
- Transfer it during I/O

The File System uses several designations and classifications to facilitate file identification. The use of file designators and classes allows the File System to recognize a file so that commands and programs can reference it. MPE/iX allows you to designate a name for a file in several ways. Techniques for file identification are described in the subsections below.

File Designators

A file designator is a name used by the File System to reference a file with great flexibility. It allows you to equate a name other than the file name used in the standard file reference format, described below.

The two types of MPE/iX file designators are:

- Formal file designator
- Actual file designator

A formal file designator is the name by which a program recognizes a file. It is specified programmatically or in a file equation and is not the file name found in the file list of a directory. It is the file name coded into the program, along with the program's specifications for the file. The `:FILE` command refers to a file by its formal file designator. A formal file designator is frequently called a user file name.

An actual file designator is a file name provided by the user for the File System to use in place of the formal file designator to accomplish a task. In many cases, the formal file designator and the actual file designator are the same. The actual file designator is the file name given in the file list of a directory. It is described in the standard file reference format:

`filename`[*/lockword*][*.groupname*][*.accountname*]

To read from an input file, a program requires that a name for the file must be specified in the program. However, it is desirable to allow a user at run time to provide a file of any name to be used for this purpose. This is accomplished by using file designators to associate the file name the user provides with the file name the program expects. Programmers can use an arbitrary name as a formal file designator and equate it to an appropriate actual file designator at run time, a technique facilitating maximum file reference flexibility. When used in this way, the formal file designator contains one to eight alphanumeric characters, beginning with a letter.

Note `HPFOPEN` option (Item #38), the *file privilege option*, when used to set a `NEW` file's privilege level to other than three (least-privileged, of user level), disallows all subsequent access to that file by the `FOPEN` intrinsic. (For compatibility reasons, `FOPEN` can access only a file whose file privilege level is three.)

The formal file designator is the name passed to the `HPFOPEN` or `FOPEN` intrinsic and used when `HPFOPEN` or `FOPEN` looks for a file equation for that file. When users invoke a `:FILE` command, they can specify a file name to equate to the formal file designator. This file name is the actual file designator. At run time, the program will find the actual file designator associated with the formal file designator and open the user-specified file. Figure 6-4 shows an example of using file designators. In this example, the file directory on disc shows an actual file designator `PAYROLL5` and the `REPORT` program contains a formal file designator named `PAYROLLX`. At run time, a user can use the `:FILE` command to create a file equation associating the name `PAYROLLX` with the name `PAYROLL5`. When the user runs the `REPORT` program, it uses the content of the `PAYROLL5` file wherever it performs an operation on `PAYROLLX`.

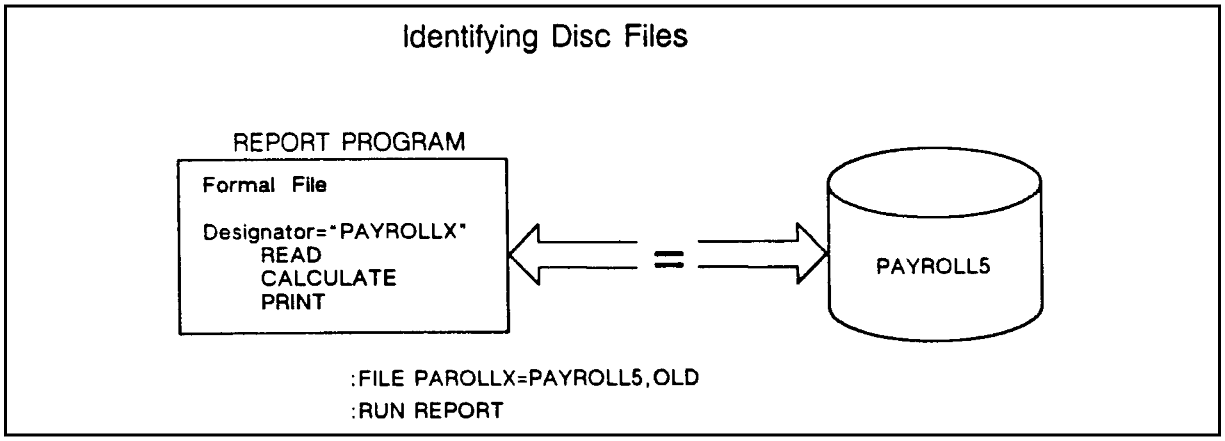


Figure 6-4. Identifying a Disc File using File Designators

Backreferencing a File

You can invoke a :FILE command that refers to a previously invoked :FILE command using a technique called "backreferencing." Once you establish a set of specifications in a :FILE command, you can apply them to other file references in your job or session by using the formal file designator, preceded by an asterisk (*). For example, if you have defined a file named MYFILE in a :FILE command, you can later refer to all the same specifications by stating *MYFILE. This is useful when you want to be sure the file specifications for a file used by one program match those already set up for a file in another program. It also saves supplying the same parameters over again.

Whenever you reference a predefined file in a File System command, you must enter the asterisk before the formal file designator if you want the predefinition to apply.

Figure 6-5 shows an example of back referencing. In this example, a :FILE command equates the formal file designator EMPLOYEE in the PERSONEL program with the actual file designator MASTER. At run time, the program opens a file named MASTER where it specifies EMPLOYEE. If you wanted to run a different program, named FINANCE, with the file MASTER, back referencing allows access to it without requiring that you repeat all the parameters that were specified in the first :FILE command. The :FILE command for the FINANCE program simply back references the actual file designator given in the previous :FILE command by specifying the string *EMPLOYEE. This allows access to the MASTER file with all the same attributes specified in the first :FILE command.

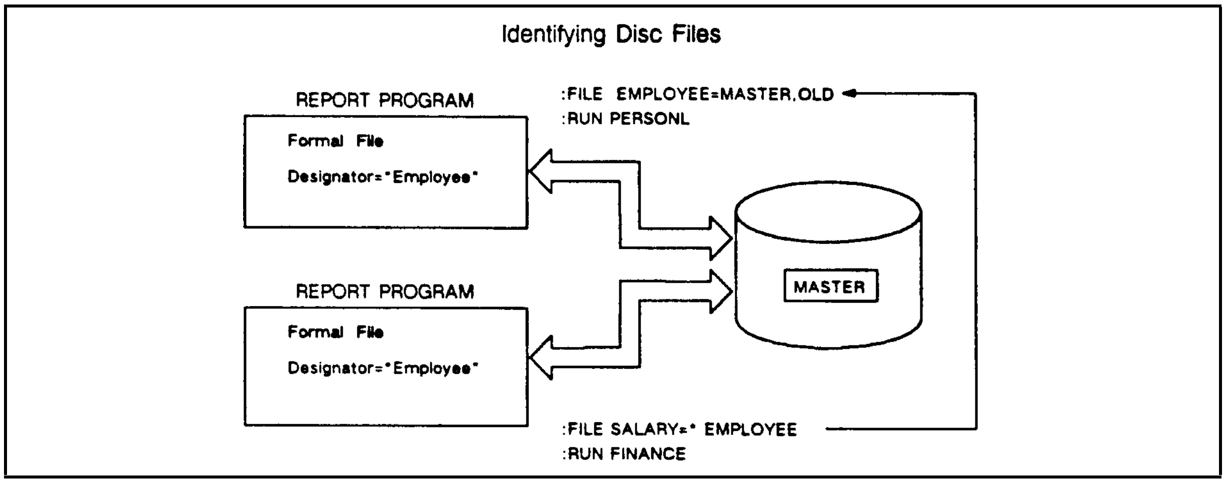


Figure 6-5. Backreferencing a Previously Identified File

For detailed information, refer to *Accessing Files Programmer's Guide* (32650-90017).

File Classes

The File System recognizes two general classes of files:

- User-defined files, which users define, create, and make available for their own purposes.
- System-defined files, which the File System defines and makes available to all users. For example, they are used to indicate standard input/output devices.

The file classes are distinguished by the file names and other descriptors (such as group or account name) that refer to them. You can use a file name and descriptors, in combination, as either:

- A formal file designator in a program or file equation
- An actual file designator that identifies the file to the File System

User-defined Files

You can reference any user-defined file by writing its name and descriptors in the standard file reference format, as follows:

`filename`[*/lockword*][*.groupname*][*.accountname*]

The maximum size of a file designator written in this format is 35 characters, including delimiters.

When you reference a file that belongs to your logon account and group, you can use the simplest form of the standard file reference format; the user or system file name. Specify both the actual and formal file designators in the standard file reference format.

The file name is from 1 to 8 alphanumeric characters, beginning with a letter (unless the file has a lockword, in which case you must specify the lockword and a delimiter). In the following examples, both formal and actual file designators appear in this format:

```
:FILE ALPHA=BETA
:FILE REPORT=OUTPUT
```

```
:FILE X=AL126797
:FILE PAYROLL=SELFL
```

A file name must be unique in its group. A reference to a file is always qualified by adding to the file name the name of the group and account in which the file resides. For example, if you create a file named `FILX` under `GROUPA` and `ACCOUNT1`, the system will recognize your file as `FILX.GROUPA.ACCOUNT1`. A file with the same file name created under a

`GROUPB` is recognized as `FILX.GROUPB.ACCOUNT1`. Thus, you need only to ensure that a file name is unique in its group.

Groups serve as the basis for your local file references. When you log on, if the default File System file security provisions are in effect, you have unlimited access to all files assigned to your logon group and your home group. You are permitted to read and execute programs residing in the public group of your log on account. This group, always named `PUB`, is automatically created under every account to serve as a common file base for all users of the account. You can also read and execute programs residing in the `PUB` group of the System Account. This special account, always named `SYS`, is available to all users on every system.

You can refer to files that belong to different logon accounts and groups by specifying qualifying information in the optional parameters of the standard file reference format. For detailed information, refer to *Accessing Files Programmer's Guide* (32650-90017).

If you do not supply explicit group and account name information in a standard file reference format, MPE/iX supplies the defaults of the group and account in which you are currently logged on.

When you create a disc file, you can assign to it a lockword. It must later be supplied in the standard file reference format to access the file in any way. The lockword is independent of, and serves in addition to, the other File System security provisions governing the file.

You assign a lockword to a new file by specifying it in the *filereference* parameter of the `:BUILD` command or in the *formalreference* parameter of the `HPFOPEN` or `FOPEN` intrinsic used to create the file.

For detailed information on using lockwords and supplying lockwords in jobs and sessions, refer to *Accessing Files Programmer's Guide* (32650-90017).

System Files

System-defined files are:

```
$STDIN
$STDINX
$STDLIST
$NEWPASS
$OLDPASS
$NULL
```

`$NEWPASS` and `$OLDPASS` are disc files that provide temporary files for passing information between job steps. `$STDIN`, `$STDINX`, and `$STDLIST` are device files that provide default input and output files. The system-defined file `$NULL` is in neither category because it is not associated with a disc or device. Figure 6-6 shows system-defined files in use.

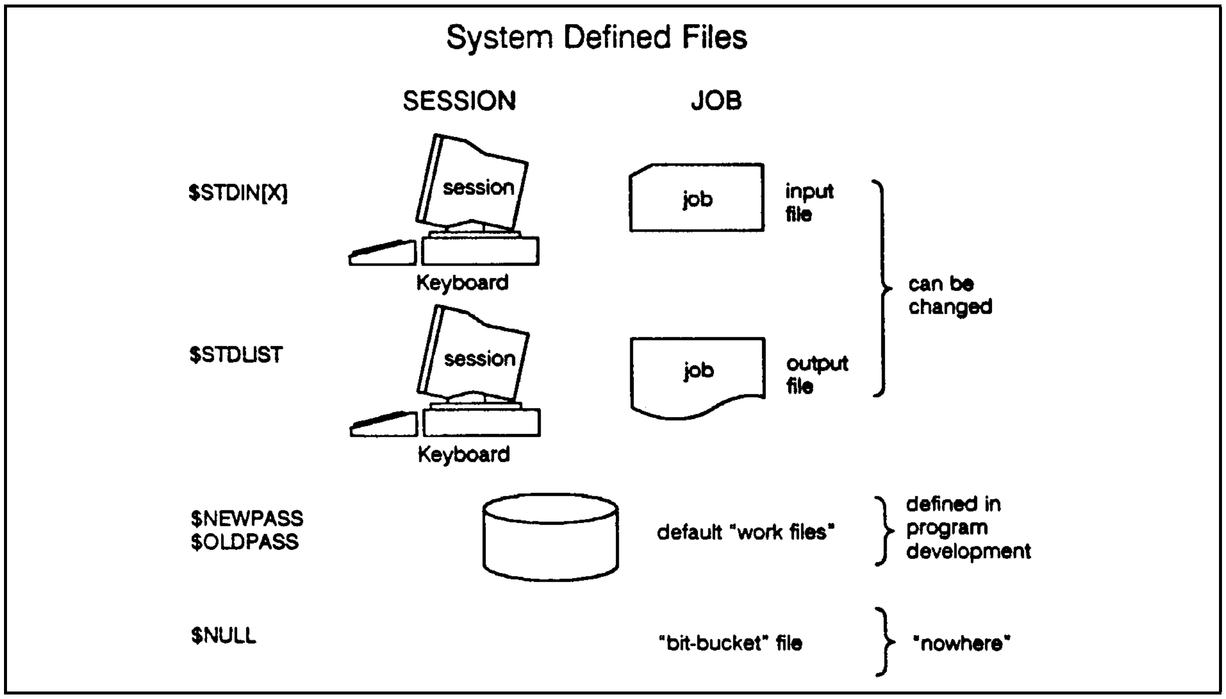


Figure 6-6. System Files in Use

System-defined file designators indicate files that the File System uniquely identifies as standard input/output devices for jobs and sessions.

These designators are:

- `$STDIN`; the standard job or session input device from which your job or session is initiated.
- `$STDINX`; the same as `$STDIN`, except handling of MPE/iX command images is different.
- `$STDLIST`; standard job or session listing device.
- `$NULL`; name of a file that is always treated as an empty file.

These designators are useful in redirecting program input and output to standard devices. For detailed information on system-defined files, refer to *Accessing Files Programmer's Guide* (32650-90017).

You can use a formal file designator in a program to specify a system-defined file. For example, a program can specify the system-defined file name `$STDLIST` to make output appear on a terminal; or it can assign output to `$NULL` to test a program to see if it compiles and runs to completion, where the actual output is not needed. Figure 6-7 shows examples of using a system defined file.

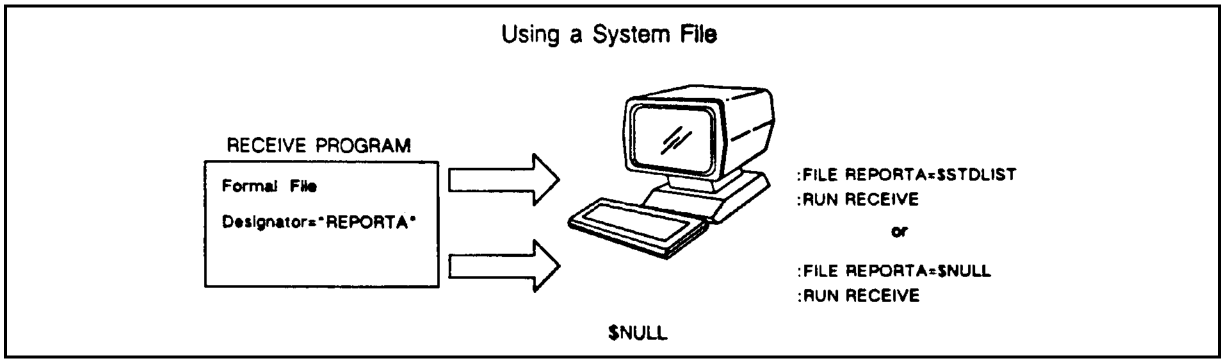


Figure 6-7. Using a System File

Input/Output Sets

All file designators can be classified as an input set, use for input files of an output set, used for output files. For your convenience, these sets are summarized in Table 6-3 and Table 6-4. For information on interactive and duplicative input files and output files, refer to *Accessing Files Programmer's Guide* (32650-90017).

Table 6-3. Input Set

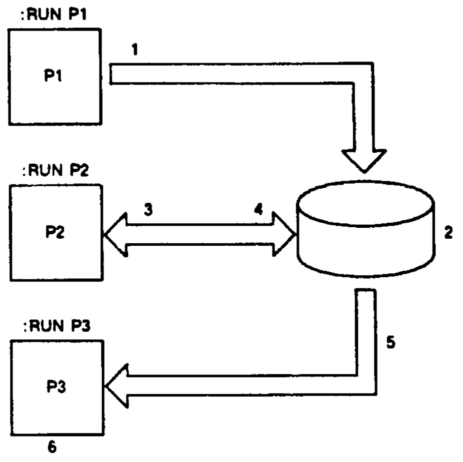
File Designator	Function or Meaning
`$STDIN`	Job/session input device.
`$STDINX`	Job/session input device allowing commands
`$OLDPASS`	Last `$NEWPASS` file closed.
`$NULL`	Constantly empty file that returns end-of-file indication when read.
**formal- designator*	Back reference to a previously defined file.
filereference	File name (including any account and group names and lockword, if necessary). Indicates an old file. May be a job/session temporary file created in a program (including the current one) run in the current job/session. May be a permanent file saved by any program or a `:BUILD` or `:SAVE` command in any job/session.

Table 6-4. Output Set

File Designator	Function or Meaning
`$STDLIST`	Job/session list device.
`$OLDPASS`	Last file passed.
`$NEWPASS`	New temporary file to be passed.
`$NULL`	Constantly empty file that returns end-of-file indication when read.
**formal- designator*	Back reference to a previously defined file.
filereference	File name (including any account and group names and lockword, if necessary). Unless you specify otherwise, it is a temporary file residing on disc that is destroyed on termination of the creating program. If closed as a job/session temporary file, it is purged at the end of the job/session. If closed as a permanent file, it is saved until you purge it.

Passed Files

Programmers, particularly those writing compilers or other subsystems, sometimes create a temporary disc file that can be automatically passed to succeeding MPE/iX commands within a job or session. This file is always created under the special name `$NEWPASS`. When the program closes the file, MPE/iX automatically changes its name to `$OLDPASS` and deletes any other file named `$OLDPASS` from the job or session temporary file domain. (Domains are described in a subsection below). After the file is closed, your commands and programs reference the file as `$OLDPASS`. Only one file named `$NEWPASS` and one file named `$OLDPASS` can exist in the job or session domain at any one time. Figure 6-8 shows the automatic passing of files between program runs. For an example of file passing and detailed information, refer to *Accessing Files Programmer's Guide* (32650-90017).



1. User Program P1 writes $NEWPASS.
2. $NEWPASS is closed; name changed to $OLDPASS.
3. Program P2 reads from $OLDPASS and writes to $OLDPASS.
4. $OLDPASS closed; reamins $OLDPASS.
5. Program P3 reads from $OLDPASS.
6. $OLDPASS remains until replaced, deleted, or saved (renamed).

Figure 6-8. Passing Files between Program Runs

`$NEWPASS` and `$OLDPASS` are specialized disc files with many similarities to other disc files, but with some differences. For example, the default close disposition of `$NEWPASS` includes renaming it. For detailed comparisons of `$NEWPASS` to new files and `$OLDPASS` to old files refer to *Accessing Files Programmer's Guide* (32650-90017).

Searching File Directories

There are two directories with addresses of files: Job Temporary File Directory for the addresses of `TEMP` files, and System File Directory for the addresses of `PERMANENT` files.

There is no directory for `NEW` files. When a file is opened, the File System may search both directories, depending on the file domain options specified, starting with the Job Temporary File Directory, until it finds the file address associated with the file.

Finding Files

You can use the the following commands to see file and file equation lists:

- `:LISTFILE` command for `PERMANENT` files
- `:LISTFTEMP` command for `TEMP` files
- `:LISTEQ` command for current file equations

For a detailed description of the commands, refer to *Getting System Information Programmer's Guide* (32650-90018).

Types of Operations Allowed on Files

The types of operations most frequently used on a file involve reading, writing, saving, appending, information to the end of a file, updating a file, and various combinations of operations. The type of operation allowed is determined when a file is opened. You can override the operations allowed by system defaults with specifications in your program or the `:FILE` command. The type of operation is not a permanent characteristic of a disc file, so it is not in the file label. Thus, no overriding can take place at this level, although the physical characteristics of a device file may dictate the types of operations allowed. For example, a line printer is always a write-only file, and a real operation cannot be specified for it.

Physical characteristics of a file cannot be changed after it has been created. Thus it is logical for the file label to take precedence over all commands. Other determinants are effective only when a new file is being created.

Note `:FILE` commands, `HPFOPEN` calls, and `FOPEN` calls cannot alter physical characteristics of an existing file.

Opening a File

Before a process can read, write, or manipulate a file, it must initiate access to the file by opening it with the `HPFOPEN` or `FOPEN` intrinsic call. This call applies to all disc and device files. When `HPFOPEN` or `FOPEN` is executed, it returns a file number used to identify the file in subsequent intrinsic calls issued by the process.

If the file is successfully opened, the file number returned is a positive integer. If the file cannot be opened, the file number returned is zero (0).

If the process issues more than one HPFOPEN or FOPEN call for a file before it is closed, it results in multiple, logically separate accesses of that file. In this case, MPE/iX returns a unique file number for each access and maintains a separate logical record pointer. (indicating the next sequential record to be accessed) for each access where you did not request or permit the *multi-access option* at HPFOPEN or FOPEN time.

When you open a file, HPFOPEN or FOPEN establishes a communication link between the file and your program. The link is established by determining the device on which the file resides, allocating it to your process, verifying access right, and performing other required tasks. These tasks include constructing control blocks required by MPE/iX for access to this file. For detailed information on establishing the communication link and determining the File System hierarchy, refer to *Accessing Files Programmer's Guide* (32650-90017).

For an example of opening new and old disc files, refer to *Accessing Files Programmer's Guide* (32650-90017).

When a process opens a disc file, the HPFOPEN or FOPEN call specifies that the file is OLD (located in either the System Permanent File Directory or the Job Temporary File Directory) or NEW. An OLD file is an existing file, and a NEW file is a file to be created during the process. When a process accesses a file residing on a non-sharable device, the device's attributes may override the OLD or NEW specification. Specifically, a device used only for input automatically imposes a PERMANENT domain on a file. A device used only for output, such as a line printer, automatically imposes a NEW domain on a file. Serial input/output devices, such as terminals and magnetic tape units, follow the *domain option* specification in the HPFOPEN or FOPEN call.

Note The HPFOPEN intrinsic assumes that all files on non-sharable devices (device files) are PERMANENT files. To maintain compatibility with MPE V/E, device files can be opened with the *domain option* set to NEW, but a warning is returned in the *status* parameter.

When a job attempts to open a PERMANENT file on a non-sharable device, MPE/iX searches for the file in its internal tables. If it does not find the file, it requests the operator to locate the file. When a job opens a new file on a non-sharable device (other than magnetic tape), it uses the first available device. When a job opens a new file on a magnetic tape unit, the operator is usually required to make the tape available. The specification of a device name or device class when HPFOPEN or FOPEN is issued, implies a request for the initial allocation of an unopened device.

For detailed information on, and examples of, opening a file on a non-sharable device, refer to *Accessing Files Programmer's Guide* (32650-90017).

Closing a File

You terminate access to a file from a program with the FCLOSE intrinsic. It applies to files on all devices (disc files and device files) and deallocates the device on which the file resides. If your program has several concurrent HPFOPEN or FOPEN calls issued to the same file, the device is not deallocated until the last FCLOSE intrinsic is executed.

You can use the FCLOSE intrinsic to change the disposition of a disc or magnetic tape file. For example, a file you open as a NEW file can be closed and saved as an OLD file with permanent or temporary disposition. A disc file you open as a TEMP file can be closed as TEMP or saved as a PERMANENT file.

When a program opens a NEW disc file with an HPFOPEN or FOPEN call, the File System does not search to determine if a file of the same name exists. This occurs when you attempt to save a file with the FCLOSE intrinsic.

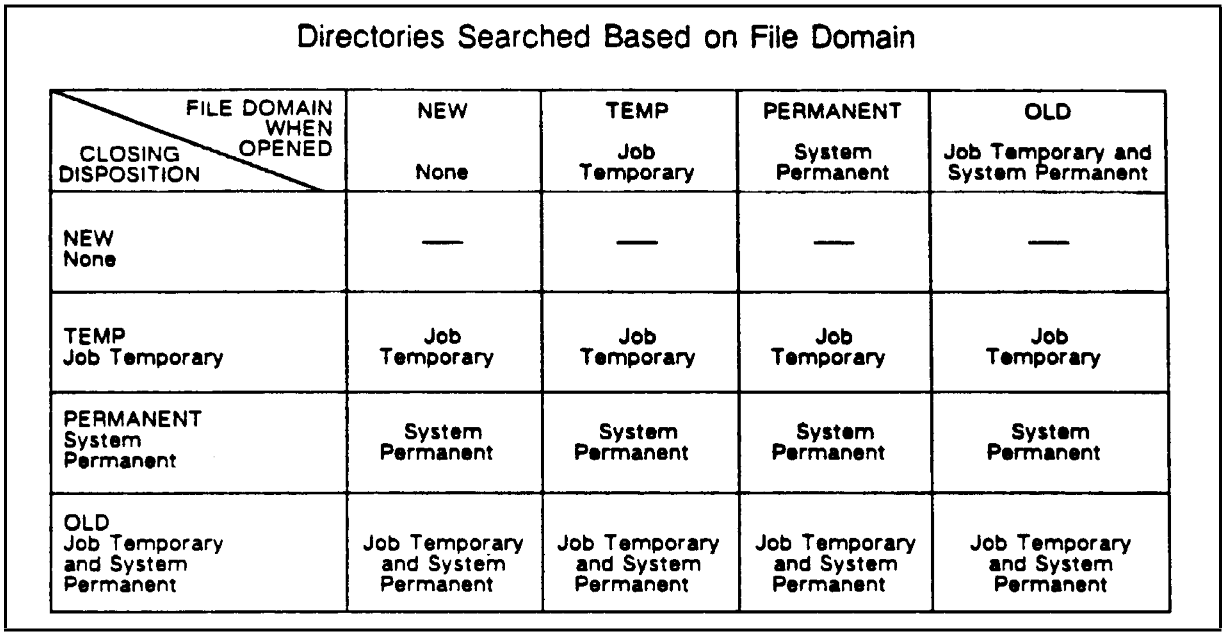
When a program opens a disc file specified as a NEW file in the option of an HPFOPEN or FOPEN call that determines the domain, and saves it with the FCLOSE intrinsic, the MPE/iX File System conducts a search. If the file is to be saved as a TEMP file, it searches the Job Temporary File Directory. If the file is to be saved as a PERMANENT file, it searches the System Permanent File Directory. If the File System finds a file of the same name in a directory it searches, it returns an error code to your program. Thus, you can open a NEW file with the same name as an existing file, but an error occurs if an FCLOSE intrinsic attempts to save it in the same domain with a file of the same name.

Similarly, when a program opens a disc file specified as a TEMP file in the *domain option* of an HPFOPEN or FOPEN call and saves it with the FCLOSE intrinsic, only the Job Temporary File Directory (not the System File Directory) is searched. Thus it is possible to have three files with the same name, a PERMANENT file, a NEW file, and a TEMP file. If a file opened as TEMP is closed and saved as a PERMANENT file with the FCLOSE intrinsic, MPE/iX searches the System Permanent File Directory. If it finds a file of the same name, it returns an error code to the program.

If a program opens a disc file specified as OLD in the *domain option* of an HPFOPEN or FOPEN call and saves it with the FCLOSE intrinsic, the File System searches the Job Temporary File Directory and the System Permanent File Directory to determine if a file of the same name already exists. If it finds a file of the same name in either directory, it returns an error code to your program.

If your program does not issue an FCLOSE intrinsic call on files that have been opened, MPE/iX closes all files automatically when the program's process terminates. In this case, MPE/iX closes all opened files with the same disposition they had before being opened. NEW files are deleted, OLD files are saved and assigned to the domain in which they previously belonged, either PERMANENT or TEMP. This can be altered with a :FILE command.

For examples of closing a new file as either a TEMP or a PERMANENT file, refer to *Accessing Files Programmer's Guide* (32650-90017). Figure 6-9 shows the directories searched for file names when a file is opened and closed, depending on the file domain.



Directories Searched Based on File Domain

CLOSING DISPOSITION \ FILE DOMAIN WHEN OPENED	NEW None	TEMP Job Temporary	PERMANENT System Permanent	OLD Job Temporary and System Permanent
NEW None	—	—	—	—
TEMP Job Temporary	Job Temporary	Job Temporary	Job Temporary	Job Temporary
PERMANENT System Permanent	System Permanent	System Permanent	System Permanent	System Permanent
OLD Job Temporary and System Permanent	Job Temporary and System Permanent	Job Temporary and System Permanent	Job Temporary and System Permanent	Job Temporary and System Permanent

Figure 6-9. Directories Searched Based on File Domain

The operation of the FCLOSE intrinsic used with unlabeled magnetic tape is dependent on conditions within the process using the device. It is possible for a single process to HPFOPEN (or FOPEN) a magnetic tape device using a device class and later HPFOPEN (or FOPEN) on the same device again by using its device name/logical device number. You can do this in a way that makes more than one tape file open concurrently.

When no concurrent tape files are open, a tape is closed with the temporary no-rewind disposition, rewound, and unloaded. When file open and file close calls are nested, tape files can be closed without deallocating the physical device, as shown in Figure 6-10.

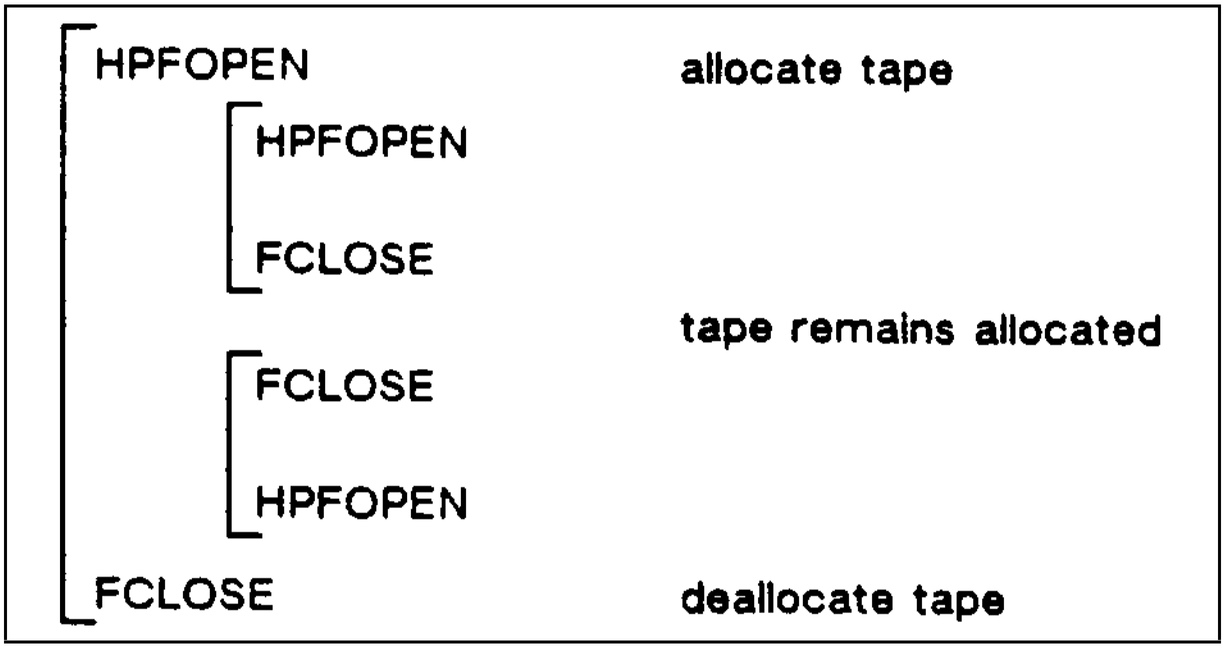


Figure 6-10. Nested HPFOPEN/FOPEN and FCLOSE Pairs

Nesting of `HPFOPEN` or `FOPEN` and `FCLOSE` pairs keeps a tape that has been `FCLOSE`d from rewinding. A tape closed with the `TEMP`, no-rewind disposition is rewound and unloaded unless the process closing it has another file currently open on the device.

For detailed information on file disposition, end-of-file marks, and use of `FCLOSE` with magnetic tape, refer to *Accessing Files Programmer's Guide* (32650-90017).

Using Files at Run Time

At run time, you can perform the following operations with files:

- Identify disc files to use
- Use a system-defined file
- Use a device file
- Specify closing disposition of a file
- Create a file, as needed
- Determine operations allowed for a file
- Define file accessibility

Hierarchy of File Overrides

The :FILE command cannot override at run time any specifications in the disc file label for an existing disc file. (The file label specifies physical characteristics of the file; record structure, file structure, file code, and actual file name.) Anything in the disc file label has precedence. Similarly, device characteristics always prevail over a :FILE command invoked for a device file. :FILE can override the following:

- Specifications that are not in a file label for a disc file.
- Specifications that are not already given for a device file.
- Specifications given when an intrinsic in the program opens the file.

If the program creates a disc file at run time, then :FILE can override the opening characteristics that the program was compiled to use and create a file with different characteristics.

It is possible to use an intrinsic such as HPFOPEN or FOPEN to override file equations. When an intrinsic is set up to do this, the file equation produced by invoking :FILE can be ignored unless the file name given in the intrinsic opening the file is dereferenced (preceded by an asterisk (*)). For detailed information on the hierarchy of overrides, and for information on how this applies to labeled tapes, refer to *Accessing Files Programmer's Guide* (32650-90017).

When a :FILE command is entered that contradicts some of the HPFOPEN or FOPEN parameters for a file, The File System maintains a hierarchy of overrides, as shown in Figure 6-11.

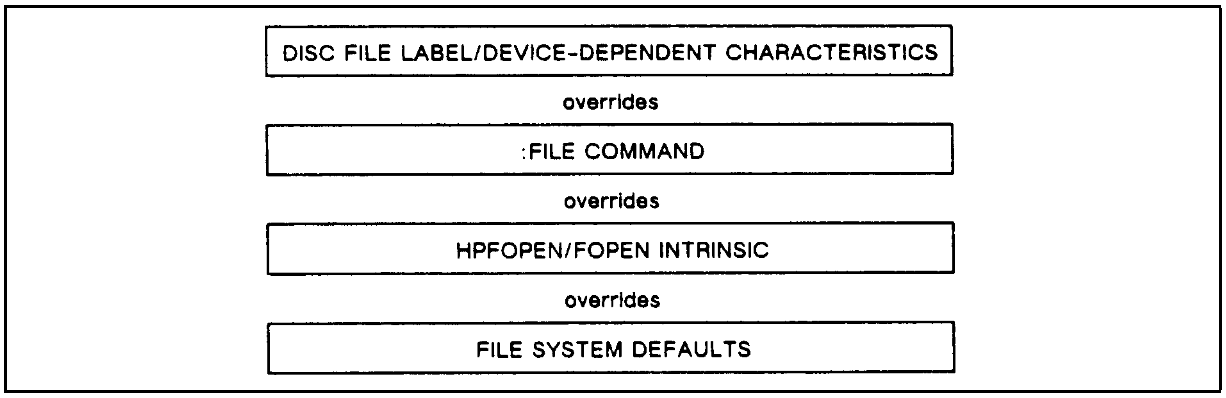


Figure 6-11. File System Hierarchy of Overrides

Record Structure

Record structures provide a definition of storage format, record type, and size.

Storage Format

Devices on the 900 Series HP 3000 can transmit information in ASCII (American Standard Code for Information Interchange) and/or binary code, depending on the device. For example, a line printer handles ASCII formatted data, while a disc can transmit and store data in either format. You can use optional parameters in the `HPFOPEN` or `FOPEN` intrinsic to specify the code (ASCII or binary) in which a new file is to be recorded when it is written to a device that supports both codes.

Examples of ASCII files on the HP 3000 include program source files, general text and document files, and MPE/iX stream files containing MPE/iX commands. Examples of binary files include program files containing linked object code and application data files. For detailed information, refer to *Accessing Files Programmer's Guide* (32650-90017).

Record Types

A file can contain records written in one of three formats, called record types:

- Fixed-length
- Variable-length
- Undefined-length

Figure 6-12 shows how the content of each record type appears.

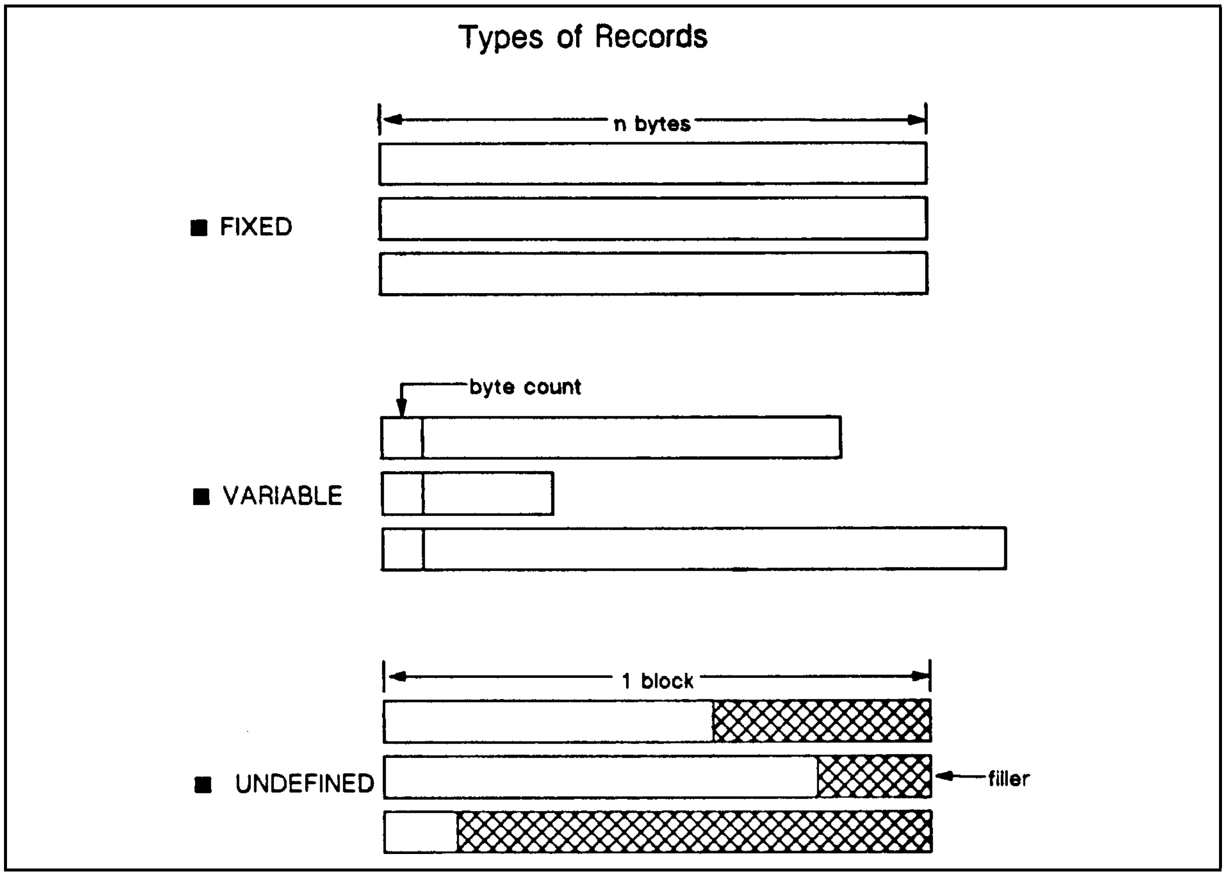


Figure 6-12. Record Types

You can specify the format for records by using one of the following:

- `HPFOPEN` or `FOPEN` intrinsic
- MPE/iX command `:BUILD`
- MPE/iX command `:FILE`

Files residing on disc or magnetic tape can contain records in any of the three formats. For files on other devices, the File System overrides any specifications you supply for records and handles them as having undefined length.

A comparison of logical record formats is given in Table 6-5. For a detailed discussion of fixed-length, variable-length, and undefined-length records, refer to *Accessing Files Programmer's Guide* (32650-90017).

Table 6-5. Comparison of Logical Record Formats

Fixed-length	Variable-length	Undefined-length
File System knows data length	File System knows data length	File System does not know data length
All records same length	Record length varies	Record length varies
Record space contains only data	Record space contains data plus byte count	Record space contains data plus filler
Request actual record size	Request maximum record size	Request maximum record size

Specifying a Record Size

You can specify the size of records in your file by using one of the following:

- MPE/iX command `:BUILD` (for disc files)
- MPE/iX command `:FILE`
- `HPFOPEN` or `FOPEN` intrinsic

You can specify your own record size or accept the default size for the device you are using. MPE/iX default record sizes for devices are shown in Table 6-6. Subsystem defaults may be different from MPE/iX defaults. For example, the Editor default can be 72 or 80 bytes, depending on the text format, while the MPE/iX standard default is the record size configured for the device. For detailed information on specifying a record size, refer to *Accessing Files Programmer's Guide* (32650-90017).

Table 6-6. Standard Default Record Sizes

Device	Record Size (Bytes)
Disc	256
Magnetic tape unit	256
Terminals (most)	80*
Line printer	132*
Plotter	510
	* Controlled by configured record length.

The File System sets up all records (ASCII and binary) to begin on 2-byte boundaries for device files. Even if you specify an odd-byte record, the next record cannot begin in the next byte. In this case, the File System adjusts the record size by adding one byte to make the next record begin on a word boundary. If the file is ASCII, the extra byte is made inaccessible for data. If the file is binary, the byte is accessible. Figure 6-13 below, shows how the File System handles odd-byte record sizes in Compatibility Mode for device

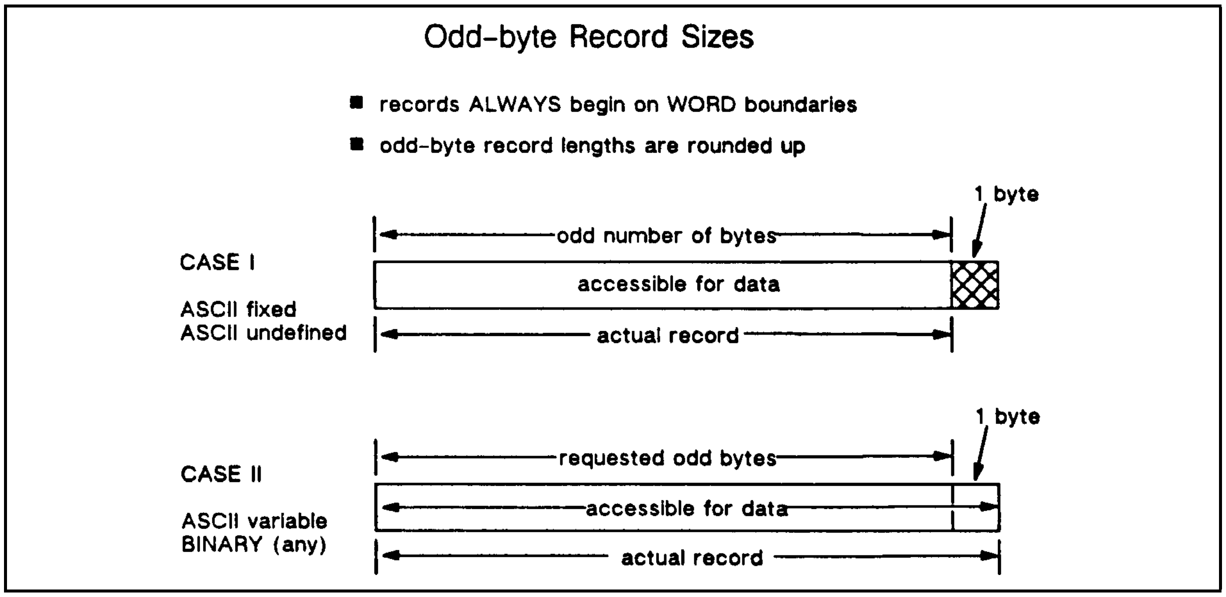


Figure 6-13. Odd-byte Record Sizes

The adjustment the File System makes for odd-byte record sizes is important

when reading tapes created on other systems. If the tape has actual odd-byte length records that are blocked, you cannot specify the exact byte length and blocking factor and obtain accurate results, because you will get a cumulative effect of each successive record being "off" by more bytes. It is a recommended practice to handle the block as one large logical record in one block and then programmatically pull apart the odd byte lengths. You can use the `DEBLOCK` option of `FCOPY` for this purpose.

File Structure

When you create a file, the File System imposes a structure and access method on the content of the file. Access to records in a file are dictated by the file type specified at file creation. Important factors in creating a file are discussed below.

File Types

Depending on your intended use of the file, you can specify the following file types:

- Standard file, which is the most common type of file. It is structured as a group of records, beginning with record 0 and ending with record *n-1*, where *n* is the maximum specified in the *filesize* option. The Standard file type is the default created when you first open a file. Examples of Standard files are Editor files and program files.
- KSAM file. Keyed Sequential Access Method (KSAM) is a method of organizing records based on the content of key fields within the records. Each record in a KSAM file contains a primary key field; the contents of the primary key field determine the primary logical

sequence of records in the file. Other key fields can be defined so that you can sequence the file in other orders. The order in which records are physically written to the file is the chronological order. The chronological order may be the same as the primary key sequence or unrelated. For detailed information on the creation and use of KSAM files, refer to Chapter 7, "Data Management" and *KSAM/3000 Reference Manual* (30000-90079).

- RIO file. Relative I/O (RIO) is a random access method that permits individual file records to be deactivated. These inactive records retain their relative position in the file. RIO files are intended for use primarily by COBOL programs; however, you can access these files by programs written in any language.

You can access RIO files in two ways: RIO access and non-RIO access. RIO access ignores the inactive records when the file is read sequentially using the `FREAD` intrinsic. These records are transparent; however, they can be read by random access using `FREADDIR`. They can be overwritten serially and randomly using `FWRITE`, `FWRITEDIR`, or `FUPDATE`. With RIO access, the internal structure of RIO blocks is transparent.

- Circular file, which is a wrap-around structure that behaves like a standard sequential file until it is full. As records are written to a circular file, they are appended to the tail of the file; when the file is filled, the next record added causes the block at the head of the file to be deleted and all other blocks to be logically shifted toward the head of the file. Circular files are particularly useful as history files and debugging files. For detailed information on circular files, refer to *Accessing Files Programmer's Guide* (32650-90017).
- Message file, which is used by Interprocess communication (IPC), a facility of the File System that permits multiple user processes to communicate with one another easily and efficiently. Message files act as first-in-first-out queues of records, with an entry made by `FWRITE` and a deletion made by `FREAD`. One process can submit records to the file with the `FWRITE` intrinsic while another process takes records from the file using the `FREAD` intrinsic.

For detailed information on the creation and use of Message files, refer to *Interprocess Communication Programmer's Guide* (32650-90019).

File Codes

MPE/iX subsystems often create special-purpose files whose functions are identified by four-digit integers called file codes, written in the system file labels. `HPFOPEN`, `FOPEN`, `:BUILD`, and `:FILE` have parameters for specifying a file code for a file when you create it.

File codes are useful when you want to run a program that produces an output file several times and need to be able to uniquely identify the output files from separate runs (or sets of runs). You can use a `:FILE` command to supply a unique file code for each run (or set of runs).

For user files, you can use any number from 0 through 1023 for a file code. If you do not specify a file code when you create a file, MPE/iX automatically applies the default file code 0. Numbers above 1023 are predefined by Hewlett-Packard for special system files. You should not redefine them. For detailed information, refer to *Accessing Files Programmer's Guide* (32650-90017).

For detailed information on creating and using a file code and on reserved file codes, refer to *Accessing Files Programmer's Guide* (32650-90017). For information on determining an existing file code, refer to *Getting System Information Programmer's Guide* (32650-90018).

Creating a File

When creating a file, you choose physical characteristics for it based on how the file will be used. These physical characteristics are determined by parameters you choose when you:

- Create the file with the `HPFOPEN` intrinsic, `FOPEN` intrinsic, or the `:BUILD` command.
- Specify the file with the `:FILE` command.

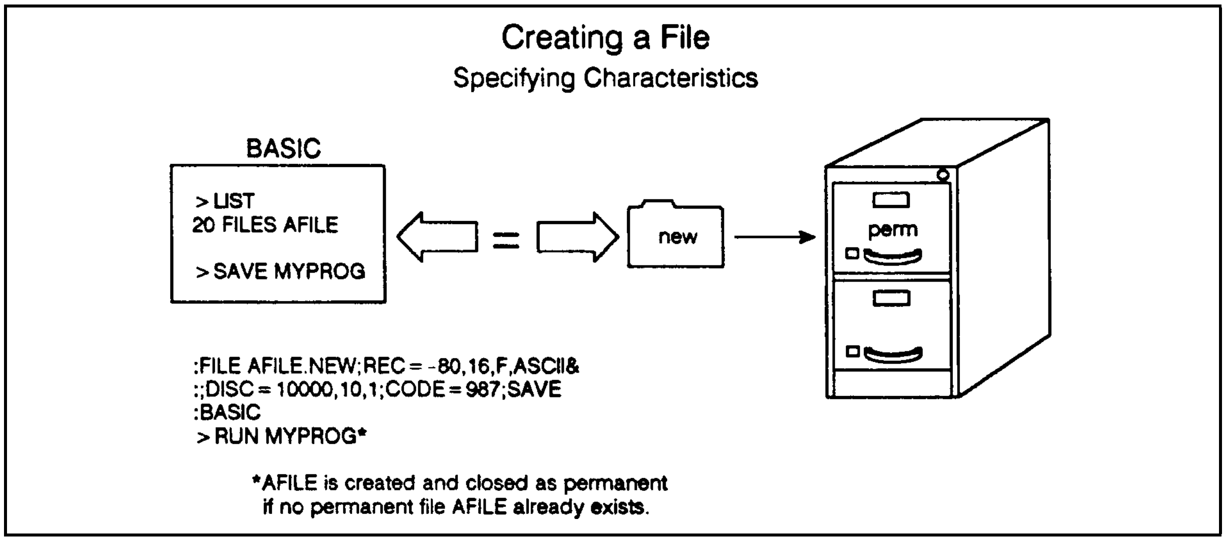
Once a file has been created, you cannot change its physical characteristics. You can change other characteristics by redefining them each time you open the file. The `:FILE` command, `HPFOPEN` intrinsic, and `FOPEN` intrinsic can alter the way a file is to be used.

The physical characteristics include record format, record size, volume class, and many other attributes. The intrinsics and commands for specifying the physical characteristics of a file are described below. (For detailed information on creating a file, refer to *Accessing Files Programmer's Guide* (32650-90017).

HPFOPEN and FOPEN Intrinsics

The `HPFOPEN` intrinsic is a programmatic tool that establishes access to a disc or device file and enables you to create a file on a sharable device. Its optional parameters are a superset of the options in the FOPEN intrinsic and provide more efficient file access. You can use HPFOPEN parameters to specify record format, record size, volume class, and many other physical characteristics. HPFOPEN and FOPEN allow file names to include command interpreter (CI) variables and expressions.

When a previously non-existent file is created at run-time, the File System must set up the physical characteristics of a `NEW` file. You can use a `:FILE` command to specify the physical characteristics of a `NEW` file. These override any system defaults and any specifications given in the program or subsystem when it opens the file because no file label has been defined for this `NEW` file. If you want to keep the `NEW` file permanently, you should close it as a `TEMP` or `PERMANENT` file. Otherwise, it is deleted when the program or subsystem terminates. Figure 6-14 shows an example of creating a file.



```
>LIST
20 FILES AFILE

>SAVE MYPROG
```

```
:FILE AFILE.NEW;REC=-80,16,F,ASCII&
:;DISC=10000,10,1;CODE=987;SAVE
:BASIC
>RUN MYPROG*
```

*AFILE is created and closed as permanent if no permanent file AFILE already exists.

Figure 6-14. Creating a File

The FOPEN intrinsic is another programmatic tool for supplying the File System with information about a file. You can use optional parameters at file creation to specify record structure, file identification, file domain, and file usage characteristics.

For detailed information on the HPFOPEN and FOPEN intrinsics, refer to *MPE/iX Intrinsics Reference Manual* (32650-90028) and *Accessing Files Programmer's Guide* (32650-90017).

The :BUILD Command

The :BUILD command creates a file in much the same way as the HPFOPEN and FOPEN intrinsics, except that they are used within a program and :BUILD is entered as an MPE/iX command. The :BUILD parameters have meanings and applications that are similar to the corresponding parameters for HPFOPEN and FOPEN. For detailed information on how to use the :BUILD command, refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364).

You can use the MPE/iX command :BUILD to create a file and specify the physical characteristics of the file to the File System. When you invoke :BUILD, the File System sets up the physical characteristics in the file label and allocates space for a permanent file on disc. For detailed information, refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364) and *Accessing Files Programmer's Guide* (32650-90017).

Deleting Files

You can use the :PURGE command to delete files from disc. This command deletes the specified file by removing the directory entry pointing to the file. It can be used to delete PERMANENT or TEMP files. The :PURGE command does not accept wildcard characters.

Renaming Files

You can use the :RENAME command on a TEMP or PERMANENT file if you are the creator of the file. :RENAME can be used to change the file name and add or change a lockword. You can use :RENAME to move a file to a different group in which you have SAVE access. Renaming the file does not change the file domain, and you cannot rename a file while it is in use.

Saving Temporary Files

You can use the :SAVE command on a temporary file to make it permanent. This operation requires you to provide the file's lockword (if it has one). For example, you could save $OLDPASS to a new name using the following syntax, where *newfilereference* is the name to which you want to save $OLDPASS:

```
:SAVE $OLDPASS, newfilereference
```

Listing File Characteristics

You can use the `:LISTFILE` command to see what the characteristics of a file are. `:LISTFILE` provides the following information:

- File names and structure of permanent disc files
- Various levels of detail about each file, depending on the user's capability on the system
- Number of records in the file
- Number of extents
- Maximum number of extents
- Records per block

The :FILE Command

The `:FILE` command determines how a file will be accessed. You can use `:FILE` to describe any of the characteristics available with `HPFOPEN`, `FOPEN`, or `:BUILD`, but you cannot actually create a file with the `:FILE` command. While `HPFOPEN`, `FOPEN`, and `:BUILD` physically allocate space for a file and define its characteristics, the `:FILE` command can define only how a file will be accessed at run time.

`FILE` sets up an environment for the file at run time and specifies the file's attributes when it is opened. When you invoke a `:FILE` command, the Command Interpreter checks it for syntactic correctness and saves a file equation in a job or session table for use when the referenced file is opened. Another process, such as an `HPFOPEN` or `FOPEN`, performed on the referenced file activates the information specified in the `:FILE` command. To be effective, the `:FILE` command must be invoked before the file is accessed because the parameters specified for `:FILE` take effect when the file is accessed. They remain in effect until one of the following conditions occurs:

- The job or session ends.
- A `:RESET` command is invoked.
- The parameters are overridden by another `:FILE` command invoked for the same formal file designator.

File equations are kept track of by a table created for the job or session. You can use the MPE/iX commands `:LISTEQ` to see a list of the current file equations and and `:LISTFTEMP` to see a list of the current temporary file names.

For an example of using the `:FILE` command and a comparison of the parameters for `HPFOPEN`, `FOPEN`, and `:FILE`, refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364) and *Accessing Files Programmer's Guide* (32650-90017).

Record Selection and Data Transfer

Data is transferred in the File System by a mechanism including records, record pointers, and the intrinsics for transferring them.

Record Pointers

The File System uses record pointers to find specific records for use. Record pointers block and deblock records and indicate specific records in a file. A file opened with the *inhibit buffering option* parameter set to `BUF` (the default) is accessed by using a record pointer. (A file opened with the *inhibit buffering option* parameter set to `NOBUF` is accessed via a block pointer.)

Note Buffering does not apply to ordinary disk files.

Pointer Initialization

When you open a file, the `HPFOPEN` or `FOPEN` intrinsic sets the record pointer to record 0 (the first record in the file) for all operations. If you have opened the file with Append access (using the *access type option* parameter), MPE/iX moves the record pointer to the end of the file prior to a write operation. This ensures that any data you write to the file is added to the end of the file instead of written over existing data. (Append and other access types are described later in this manual.) Following initialization, the record pointer can remain in position at the head of your file, or it can be moved by the intrinsics used in record selection (for example, `FREAD`, `FWRITE`, and `FPOINT`).

Record Selection

Various File System intrinsics are designed to transfer records to and from files. The record pointer indicates the specific location where a file can be accessed. Records can be transferred to or from this location, or the pointer can be moved to another place in the file you wish to access.

There are five methods of record selection for accessing a file:

- Sequential access, in which you transfer data to and from the place the record pointer currently indicates.
- Random access, in which you use `FPOINT` to move the record pointer before transferring data, or you use `FREADDIR` to choose it at access time.
- Update access, in which you choose a record and write a new record over it.
- RIO access, in which you access only records that are activated.
- Mapped access is a special type of access available only through the `HPFOPEN` intrinsic, in which you bypass File System data transfer mechanisms by referencing the file by using a pointer declared in your program.

For detailed information on record selection methods and the intrinsics used for data transfer, refer to *Accessing Files Programmer's Guide* (32650-90017).

Mapped Access to Disc Files

MPE/iX employs a "mapped files" technique for performing disc file access. File access efficiency is improved when code and data portions of files required for processing reside in memory. Accessing memory is faster than performing physical disc I/O operations. The mapped files technique eliminates file system buffering and optimizes global system memory management.

File mapping is based on MPE/iX demand paged virtual memory, which uses to advantage the large amount of virtual memory on the system. When a file is opened, it is logically mapped into virtual memory. An open file and its contents are referenced by virtual addresses. Each byte of each opened file has a unique virtual address.

File mapping improves I/O performance without imposing additional CPU overhead or sacrificing data integrity and protection. Traditional disc caching schemes for increasing I/O performance impose a CPU overhead penalty. The 900 Series hardware and system architecture allow MPE/iX to perform file mapping without incurring this penalty. System hardware performs the virtual to physical address translations for locating portions of the mapped files, thus eliminating CPU overhead for this function.

If the required pages are not in memory, the MPE/iX Memory Manager fetches them directly from disc and places them in memory. This eliminates File System buffering. Pages are "prefetched" to reduce the amount of physical disc I/O. Prefetching means that the page specified for fetching and the group of pages surrounding it are a fetched all at once. This improves efficiency because the processor is likely to require pages that are located near each other. Two benefits of this are:

- Eliminating unnecessary data movement in memory improves system performance.
- Memory space usage is optimized.

MPE/iX File System access intrinsics are built on the mapped file technique. Programs using file access methods supported by MPE file types and intrinsics obtain the benefits of file mapping without requiring changes.

You can directly access mapped files when programming in Native Mode languages with pointers by using the `HPFOPEN` intrinsic. For example, you can obtain the advantage of File System naming and data protection for accessing array type structures and developing specialized access methods.

You can write programs that address files through virtual memory, instead of calling File System intrinsics for disc reading and writing. The file interface provides opening and closing of user mapped files with normal naming and security, but with improved `LOAD` and `STORE` speed on file references.

Multiple Record Transfers

In most applications, programs conduct input and output in normal recording mode, where each read or write request transfers one logical record to or from the stack. In specialized applications, however, you may want a program to read or write, in a single operation, data that exceeds the logical record length defined for the input or output file. For example, multirecord transfer may be preferable, as a good programming practice, for applications that run only on MPE/iX.

You can bypass the normal record-by-record input and output, instead receiving large data transfers by specifying multirecord mode (MR) by using the *multirecord option* parameter in the `HPFOPEN` or `FOPEN` call, or by using the `:FILE` command. For detailed information and examples on multiple record transfers, refer to *Accessing Files Programmer's Guide* (32650-90017).

Data Transfer Control Operations

To move a record pointer to a particular place without necessarily transferring data, the following intrinsics perform three types of record selection:

- `FSPACE`, provides spacing. It moves the record pointer backward or forward.
- `FPOINT`, provides pointing. It resets the record pointer.
- `FCONTROL`, provides rewinding. It resets the record pointer to record 0.

For detailed information and examples on spacing, pointing, and rewinding, refer to *Accessing Files Programmer's Guide* (32650-90017). For detail information on `FSPACE`, `FPOINT`, and `FCONTROL`, refer to *MPE/iX Intrinsics Reference Manual* (32650-90028).

Reading from a File

There are several ways to move data to a program from a disc file or device file. The File System intrinsics used for reading data from a file are:

- `FREAD` reads a logical record or a portion of a record from a file to a program.
- `FREADDIR` reads a specific logical record or portion of a record from a random access opened disc file to a program.
- `READ` reads a character string from the job or session input device (`$STDIN`) to a program.
- `READX` reads a character string from the job or session input device, (`$STDINX`) to a program.
- `FREADLABEL` reads a user file label from a disc file or labeled magnetic tape file to a program.
- By referencing a pointer to an open mapped file.

If the standard input device (`$STDIN`) and the standard list device (`$STDLIST`) are opened with an `HPFOPEN` or `FOPEN` intrinsic call, the `FREAD` and `FWRITE` intrinsics can be used with them. For example, you can use the `FREAD` intrinsic to transfer information entered from a terminal to a buffer in the stack, and you can use the `FWRITE` intrinsic to directly transfer information from the stack to the standard list device.

You can use the `FREADDIR` intrinsic to read a record from a file opened by the `HPFOPEN` or `FOPEN` intrinsic and transfer the record to an array in the stack. When `FREADDIR` reaches the end of a file, the end-of-file condition code `CCG` is returned. If `FREADDIR` does not successfully read information, a `CCL` condition code is returned.

When a labeled tape file has been opened, you can use the `FREAD` intrinsic to read data from the opened file. The system uses the block size, record size, and file format on the tape label to determine the amount of data to read. You can call `FGETINFO` or `FFILEINFO` to obtain these values.

You can use the `FREADLABEL` intrinsic to read a user-defined label on a labeled magnetic tape file. To read a user-defined header, a program must call `FREADLABEL` before issuing the first FREAD for the file. Execution of the first FREAD causes MPE/iX to skip past any unread user-defined header labels.

Other specialized read intrinsics are READX and FREADLABEL.

For detailed information on reading a file, refer to *Accessing Files Programmer's Guide* (32650-90017).

Writing to a File

You can move data from a program to a disc file or to a device file in several ways. The File System intrinsics used for writing the data are as follows:

- FWRITE writes a logical or physical record or portion of a record from your program to a file on any device.
- FWRITEDIR writes a specific logical record from a program to a disc file.
- PRINT prints a character string from a program to the job or session listing device, $STDLIST.
- FWRITELABEL writes a user file label from your program onto a disc file or labeled magnetic tape file.
- By assigning a value to a location specified by a pointer to an open mapped file.

If the standard input device ($STDIN) and the standard list device ($STDLIST) are opened with an HPFOPEN or FOPEN intrinsic call, the FREAD and FWRITE intrinsics can be used with these devices. For example, the FWRITE intrinsic can be used to transfer information from a buffer in your process stack directly to the standard list device.

A program can use the FWRITEDIR intrinsic to read records from one file and write them, in reverse order, into a second file. You can use the FGETINFO intrinsic to locate EOF in the file to be read. This information is returned to the program in a variable.

You can write to either a labeled or an unlabeled magnetic tape file. Writing to a labeled tape file differs from writing to an unlabeled tape file when the user program attempts to write over or beyond the physical end-of-tape (EOT) marker.

For detailed information on writing to a file, refer to *Accessing Files Programmer's Guide* (32650-90017).

Updating a File

The FUPDATE intrinsic updates a logical record of a disc file. It affects the last logical record (or block for NOBUF files) accessed by any intrinsic call for the file named and writes information from a buffer in the stack into this record. Following the update operation, the record pointer is set to indicate the next record position.

As a physical data storage device, magnetic tape is not designed to enable the replacement of a single record in an existing file. An attempt to perform this type of operation causes problems in maintaining the integrity of records on the tape. Magnetic tape files, therefore, should not be maintained (updated) on an individual record basis, but should be updated during copy operations from one file to another. For detailed information on updating files, refer to *Accessing Files Programmer's Guide* (32650-90017).

Using Mapped Access to a File

The MPE/iX File System employs access to mapped files, accessible through the use of pointers in the `HPFOPEN` intrinsic. A mapped file is a file that can be accessed directly through machine loads and stores, bypassing File System intrinsics normally used for disc files.

Mapped access is not available through the standard Input/Output statements of most programming languages. Thus, the programmatic use of `HPFOPEN` adds capability to programs. You can achieve mapped access to a file by declaring a pointer in a program and passing it to the `HPFOPEN` intrinsic. The pointer is initialized to point to the beginning of the data area of the opened file.

After `HPFOPEN` returns the address of the file, simply reference the pointer as an array or any type you want. The machine architecture translates the address to a file page and ensures integrity and protection of the file.

Standard disc files with fixed-length or undefined-length record formats are allowed any type of access (Read, Write, Read/Write, and so on) when opened for mapped access. The following file types are allowed Read Only access when opened for mapped access:

- Standard disc files with variable-length record formats
- KSAM files (with `COPY` mode enabled)

The following file types cannot be opened for mapped access:

- Relative I/O (RIO) files
- Message (MSG) files
- Circular (CIR) files
- KSAM files (with `COPY` mode disabled)
- Device files

The File System provides protection by keeping a list of file rights. When a page fault or protection fault occurs, it checks the list. If you pass the security check, your protection ID is placed in a range of pages allowing access to the file.

All File System and data transfer intrinsics applicable to the file can be used with a mapped file. When mixing `FREAD` and `FWRITE` calls with mapped access, you must consider the file's data type (ASCII or binary), record format, and record size to ensure that data written to the file using mapped access makes sense when read by `FREAD`.

When you open a file using mapped access and write data to it, you must use the `FPOINT` and `FCONTROL` intrinsics to reset end-of-file (`EOF`) before closing the file. Otherwise, all data written to the file is lost when you close it. In the case of a newly created file, the `EOF` initially points to record zero. Mapped accessing of a file bypasses File System services that, otherwise, automatically set various File System pointers, including `EOF` and the logical record pointer. You are responsible for resetting `EOF` prior to closing the file.

For detailed information on the advantages of mapped access to files, refer to *Accessing Files Programmer's Guide* (32650-90017).

Device File Buffers

A buffer is an area in memory used by the File System to hold one block (one physical record) of a file. When a program reads a record from a buffered file, a block is brought into the buffer. For detailed information, refer to *Accessing Files Programmer's Guide* (32650-90017).

Sharing a File

Accessing and controlling a file that is open only to you is a relatively simple matter. However, when several users are simultaneously accessing a file, each user must be aware of special considerations for sharing the file:

- How will others be allowed concurrent access to the file?
- Will concurrent access require special management?

When an `HPFOPEN` or `FOPEN` intrinsic call is issued for a file, the File System regards the request as an individual accessor of the file and establishes a unique file number and other file control information for the file. Even when a program issues several different `HPFOPEN` or `FOPEN` intrinsic calls for the same file, each call is treated as a separate accessor. Under the normal (default) security provisions of MPE/iX, when an accessor opens a file not currently in use, the access restrictions that apply to this file for other accessors depend on the access mode this first accessor requested:

- If the first accessor opens a file for read-only access, then any other accessor can open it for any other type of access (for example, write-only or append). However, the other accessors are prohibited exclusive access.
- If the first accessor opens a file for any other access mode (for example, write-only, append, or update), this accessor maintains exclusive access to the file until it closes it; no other accessor can access the file in any mode.

A program can override the defaults by specifying other options in `HPFOPEN` and `FOPEN` intrinsic calls. A user running this program can, in turn, override both the defaults and programmatic options by using the `:FILE` command. Table 6-7 describes these options and the actions MPE/iX takes when the options are in effect and simultaneous access is attempted by other `HPFOPEN` or `FOPEN` intrinsic calls. The action depends on the current use of the file and the access requested.

Table 6-7. File Sharing Restriction Options

Access Restrictions	:FILE Parameters	Description
Exclusive Access	`EXC`	After you open a file, prohibits concurrent access in any mode through another `HPFOPEN/FOPEN` request by any program (including this one), until this program issues `FCLOSE` or terminates.
Exclusive Write Access	`SEMI`	After you open a file, prohibits concurrent write access through another `HPFOPEN/FOPEN` request by any program (including this one), until this program issues `FCLOSE` or terminates.
Sharable Access	`SHR`	After you open a file, allows concurrent access in any mode through another `HPFOPEN/FOPEN` request by any program (including this one), in any session or job (including this one).

The Exclusive Access option is useful to update a file and prevent other users or programs from reading, writing, or altering information that is about to be changed.

The Exclusive-write Access option allows other accessors to read the file but prevents them from altering it. For example, it is useful for updating a parts list, where you want to append new part numbers, without prohibiting other users from reading current part numbers.

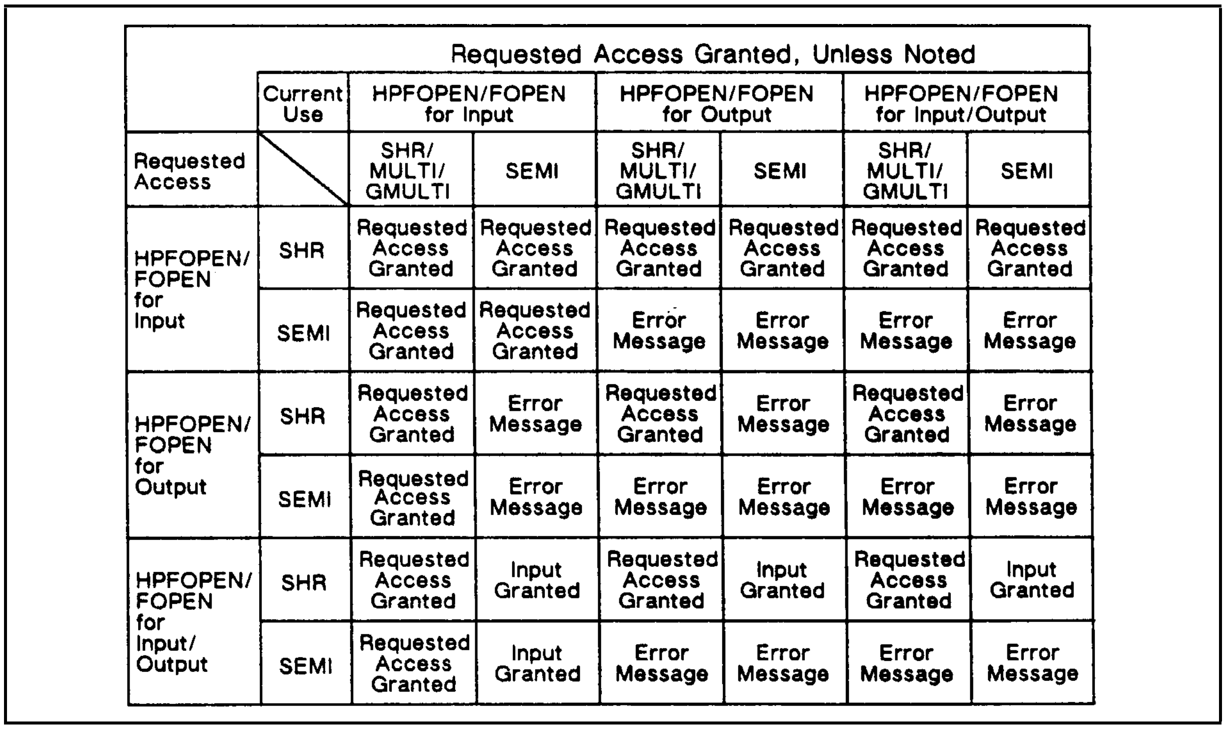
The Sharable Access option allows a file to be shared in all access modes by requests from multiple programs. It is useful for allowing several users to read different parts of the same file. Effectively, each accessor accesses its own copy of the portion of the file in the accessor's buffer.

For detailed information on Exclusive, Exclusive-write, and Sharable Access, refer to *Accessing Files Programmer's Guide* (32650-90017).

The multi-access option extends the features of Sharable Access to allow a deeper level of multiple access. It makes the file simultaneously available to other accessors in the same job or session and permits them to use the record pointer and other file-control information. When several concurrently running programs (processes) are writing to the file, the effect on the file is the same as if one program were performing all output. Multi-access provides truly sequential access by several concurrently running programs.

Global multi-access permits simultaneous access to a file by processes in different jobs or sessions. Figure 6-15 shows the action resulting from multi-access of files.

For detailed information on the multi-access and global multi-access options, refer to *Accessing Files Programmer's Guide* (32650-90017).



		Requested Access Granted, Unless Noted					
	Current Use	HPFOPEN/FOPEN for Input		HPFOPEN/FOPEN for Output		HPFOPEN/FOPEN for Input/Output	
Requested Access		SHR/ MULTI/ GMULTI	SEMI	SHR/ MULTI/ GMULTI	SEMI	SHR/ MULTI/ GMULTI	SEMI
HPFOPEN/ FOPEN for Input	SHR	Requested Access Granted	Requested Access Granted	Requested Access Granted	Requested Access Granted	Requested Access Granted	Requested Access Granted
	SEMI	Requested Access Granted	Requested Access Granted	Error Message	Error Message	Error Message	Error Message
HPFOPEN/ FOPEN for Output	SHR	Requested Access Granted	Error Message	Requested Access Granted	Error Message	Requested Access Granted	Error Message
	SEMI	Requested Access Granted	Error Message	Error Message	Error Message	Error Message	Error Message
HPFOPEN/ FOPEN for Input/ Output	SHR	Requested Access Granted	Input Granted	Requested Access Granted	Input Granted	Requested Access Granted	Input Granted
	SEMI	Requested Access Granted	Input Granted	Error Message	Error Message	Error Message	Error Message

Figure 6-15. Actions Resulting from Multiaccess of Files

When a file is shared among two or more processes and one or more of the processes is writing to it, the processes must be properly interlocked to prevent undesirable results. The FLOCK and FUNLOCK intrinsics provide the necessary interlocking by using a Resource Identification Number (RIN) as a flag to interlock multiple accessors. For detailed information on this topic, refer to *Accessing Files Programmer's Guide* (32650-90017) and *MPE/iX Intrinsics Reference Manual* (32650-90028).

Maintaining File Security

Three security features are available on the MPE/iX system to restrict access to a single file or all disc files in a particular account or group: lockwords, the MPE/iX File Access System matrix, and access control definitions (ACDs).

Lockwords

Lockwords are file "passwords" which provide a primary level of file security by restricting access to a file to users not knowing the file's lockword. These lockwords may be assigned to a file when the file is created with the BUILD command or when the file is renamed. To delimit the lockword, enter a forward slash (/) following the file name:

```
BUILD MYFILE/KEY

RENAME MYFILE/LOCK
```

For additional information, refer to the *MPE/iX Commands Reference Manual Volumes 1 and 2* (32650-90003 and 32650-90364).

MPE/iX File Access System

The HP 3000 system includes a set of security provisions with each account, group, and individual file. This set specifies restrictions on access to a single disc file or to all disc files in a particular account or group.

The restrictions are based on three factors:

- Modes of access (for example, reading, writing, or saving).
- Types of user (for example, a user with Account Librarian capability, Group Librarian capability, or a user to whom the access modes specified are permitted.)

The security provisions for any file describe the permitted access modes for various users of the file.

Specifying and Restricting File Access by Access Mode

When a program opens or creates a file, it can define the way the file can be accessed by specifying a particular access mode for the file (for example, read-only, write-only, update, and so on). These specifications apply to files on any device, and only the creator of file can change or override them.

When specifying the access mode for a file, it is important to know the location of the current end-of-file before and after the file is opened and the location of the logical record pointer that indicates where the next operation will begin. For example, the choice of the correct access mode can write a record to a file and:

- Append it following the last record or overwrite an existing record.
- Correct information in a file rather than delete it.
- Redirect output from one device to another.

For detailed information on this topic, refer to *Accessing Files Programmer's Guide* (32650-90017).

Table 6-8 describes the file access mode types.

Table 6-8. File Access Mode Types

Access Mode	:FILE Parameter	Description
Read only	`IN`	Allows you to read a file, but not to write on it.
Write only	`OUT`	Allows you to write on a file, but not to read it. Any data already in the file is deleted when the file is opened.
Write (save) only	`OUTKEEP`	Allows you to write on a file, but not to read it. You can add new records both before and after current end-of-file indicator.
Append only	`APPEND`	Allows you to append information to a file, but not to overwrite the current information or read the file. You can add new records only after the current end-of-file indicator. Used when present contents of a file must be preserved.
Input/output	`INOUT`	Allows unrestricted input and output access of file. Information already on the file is saved when the file is opened. (In general, it combines the features of `IN` and `OUTKEEP`.)
Update	`UPDATE`	Allows use of `FUPDATE` intrinsic to alter records in a file. Record is read into your data stack, altered, and rewritten to the file. All data already in the file is saved when the file is opened.

Specifying File Access by Type or User

Restrictions on accessing a file are established when the file is created according to the default established for the group and account where the file resides.

The capabilities of a user who accesses a file can determine the security restrictions applied. Table 6-9 describes the types of users recognized by the MPE/iX Security System, their mnemonic codes, and definitions.

Table 6-9. User Type Definitions

User Type	Mnemonic Code	Description
Any user	`ANY`	Any user defined in the system, including all categories defined below.
Account librarian user	`AL`	User with account librarian capability, who can manage certain files within the account that may or may not all belong to one group.
Group librarian user	`GL`	User with group librarian capability, who can manage certain files within his home group
Creating user	`CR`	User who created this file
Group user	`GU`	Any user allowed to access this group as his logon or home group, including all `GL` users applicable to this group.
Account member	`AC`	Any user with authorized access under this account, including access through `AL`, `GU`, `GL`, and `AC`.

For detailed information on this topic, refer to *Accessing Files Programmer's Guide* (32650-90017).

The user access modes to a file are determined by four separate levels of security settings. All four levels are checked and must be passed to grant a user access to a file. The only two levels over which a standard user has control are the file and lockword levels. However, it is important for a standard user to understand all levels of the security system, since the combination of all the settings determines the user's access to the file. Figure 6-16 shows the security levels checked when a user attempts to gain access to a file.

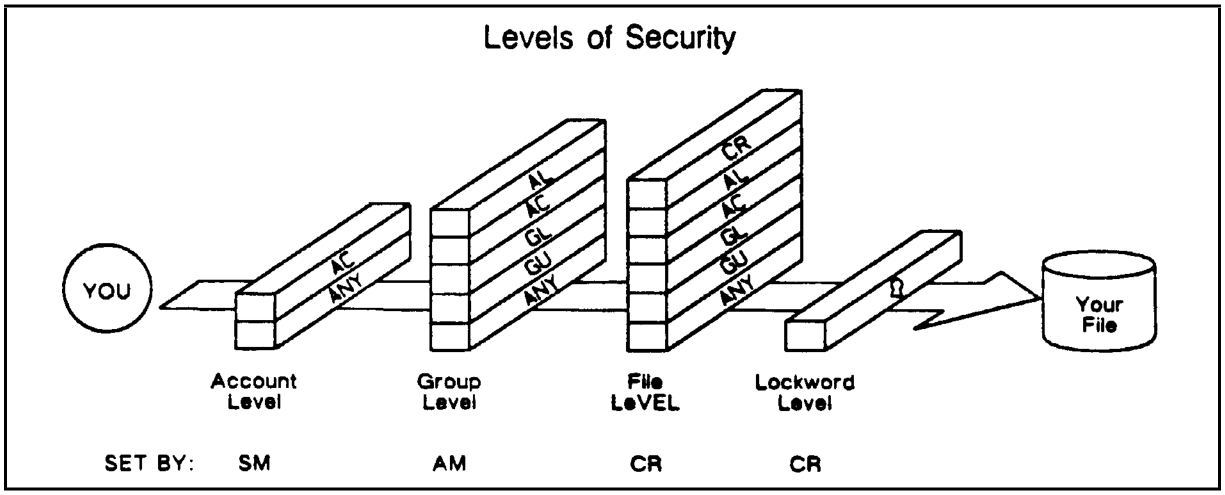


Figure 6-16. Security Level for File Access

The default security at account, group, and file levels are as follows:

- Account level

```
SYS (R,X:ANY;W,A,L:AC)
other accounts (R,X,W,A,L:AC)
```

- Group level

```
PUB (R,X:ANY;A,W,L,S:AL,GU)
other groups (R,W,A,L,X,S:GU)
```

- File level

```
(R,X,W,A,L:ANY)
```

At the group level, the MPE/iX Security System recognizes six access modes:

Reading (R)
Appending (A)
Writing (W)
Locking (L)
Executing (X)
Saving (S)

It recognizes seven user types:

Any User (ANY)
Account Librarian User (AL)
Group Librarian User (GL)
Group User (GU)
Account Member (AC)
System Manager (SM)
Account Manager (AM)

For detailed information on this topic, refer to *Accessing Files Programmer's Guide* (32650-90017).

When a file is created, the security provisions that apply to it are the default provisions assigned by MPE/iX at the file level, coupled with the user-specified or default provisions assigned to the account and group to which the file belongs. At any time, however, the creator of the file can change the file level security provisions. Thus, the total security provisions for a file depend on specifications made at three levels: account, group, and file. A user must pass tests at the account, group, and file levels (in order) to successfully access a file in the requested mode.

If no security provisions are explicitly specified by the user, the following provisions are assigned at the file level by default:

Reading (R)
Appending (A)
Writing (W)
Locking (L)
Executing (X)
Any User (ANY)

Because the total security for a file always depends on security at all three levels, a file not explicitly protected from a certain access mode at the file level may benefit from the default protection at the group level. For detailed information on this topic, refer to *Accessing Files Programmer's Guide* (32650-90017).

Access Control Definitions (ACDs)

Access Control Definitions (ACDs) allow the owner of a file to specify permissions for access to a file or to a device. An ACD takes precedence over the previous access controls, namely the MPE file access matrix and lockwords. The owner of a file is any of the following: the creator of the file, the account manager (AM capability) where the file resides, or the system manager (SM capability).

A device or file can be paired with an ACD. When a user tries to acquire the device or to open the file, the system first checks the authorization list of the associated ACD. When a file is paired with an ACD, the ACD is put into its file label extension. The ACD contains a list of *access modes* paired with users.

Specifying and Restricting File Access

Similar to the MPE File Access System, ACDs use *modes* to specify the types of access to grant or refuse to users as follows:

R	read
W	write
A	append
L	lock
X	execute
NONE	none (no access)
RACD	read and copy the ACD permission file

ACDs are defined in a similar manner as file access modes. For example:

```
ACD = (R,W:MGR.ACCTING, DENNIS.LEE; R:@.PAYROLL; A:@.@
```

MGR.ACCTING and DENNIS.LEE can read and write to the file associated with this example ACD. Any user in PAYROLL can read it, and any user on the system can append to it. If a user is not given a permission, the user does not have it.

Managing ACDs with Commands and Intrinsics

Use commands to manage ACDs interactively through the command interpreter. Use intrinsics to manage ACDs in a program.

1. Creating ACDs

   ACDs for files may be created or owned by the system manager (SM capability), the account manager (AM capability) where the file resides, or by the creator of the file. ACDs for devices can only be created or owned by the system manager (SM capability).

   When creating a new file with the `COPY`, `FCOPY`, or `RESTORE` commands, use parameters to create its ACD. Use the `ALTSEC` command or the `HPACDPUT` intrinsic to create an ACD for an existing device or file.

2. Preserving ACDs Device ACDs are not permanent objects and must be redefined every time the system is rebooted. The easiest ways to do this are to put `ALTSEC` commands into the SYSTART file, either directly or in a command file.

   File ACDs are permanent objects and survive a reboot. When storing files to tape, `FCOPY` and `STORE` store the files' ACDs unless you specify otherwise.

3. Listing ACDs

   To get information about ACDs interactively, use the `SHOWDEV` command for devices, and the `LISTFILE` command for files. To get information about ACDs programmatically, use the `HPACDINFO` or `HPACDPUT` intrinsics.

   Any user can find out whether an ACD exists for a particular device or file, but only an owner, or user granted RACD (read ACD) access, can get a listing of the ACD contents.

4. Copying ACDs

   To copy an ACD from one file to another, use the `COPYACD` parameter of the `ALTSEC` command. Only the owner or user granted RACD (read ACD) access can copy an ACD.

5. Modifying ACDs

   To change an ACD, use the `ALTSEC` command or the `HPACDPUT` intrinsic. Only an owner can alter an ACD.

6. Deleting ACDs

   To delete an ACD, use the `ALTSEC` command or the `HPACDPUT` intrinsic. Only an owner can delete an ACD.

7. Migrating ACDs

   Both MPE V/E and MPE/iX support the use of ACDs. MPE/iX, however, allows more *user-mode* pairs than MPE V/E.

Note Device ACDs shold not be migrated because they are tied to their system's configuration.

Move file ACDs between MPE V and MPE/iX by using the `STORE` and `RESTORE` commands, where `COPYACD` is the default.

Note You must have authorization to use the `COPYACD` parameter of the `STORE` and `RESTORE` commands. If you are not an owner of the file and do not have RACD permission, you will get an error.

Changing Disc File Security Provisions

The security provisions for the account and group levels are managed only by users with System Manager or Account Manager capabilities, respectively. However, you can change the security provisions for any disc file you have created by using the `:ALTSEC` command. This does not affect any account-level or group-level provisions that may cover the file, and it does not affect the security provided by a lockword. For detailed information on this topic, refer to *Accessing Files Programmer's Guide* (32650-90017).

Suspending and Restoring Security Provisions

You can temporarily suspend the security restrictions on any disc file you create by using the `:RELEASE` command. This allows any user to access the file in any mode, thus providing unlimited access to the file. It does not remove lockword protection, and it does not modify the file security settings recorded in the system. It merely bypasses them temporarily. `:RELEASE` remains in effect for a file until you enter the `:SECURE` command in this or a later job or session or until the file is modified.

For detailed information on this topic, refer to *Accessing Files Programmer's Guide* (32650-90017).

7

Data Management

Data management involves collecting data and reporting information. The following categories describe the concept of data management on a 900 Series HP 3000 system:

- Collecting data: the data entry process, including validating.
- Organizing data: analysis of data and its relationships; development of an appropriate data structure after analyzing the requirements of the application; for example, producing a logical record layout or schema.
- Storing data: the process of installing data in a meaningful form. This is the task of taking raw data and reformatting it for application programs to use.
- Accessing data: the retrieval of data for further processing. This includes additional reorganization or reduction of data for reports or other uses of the data.

Data Management Subsystems

Data Management subsystems on the 900 Series HP 3000 are:

- MPE/iX File System
- KSAM/V (Keyed Sequential Access Method)
- ALLBASE/SQL
  - TurboIMAGE/XL
  - HP SQL
- QUERY/V
- TurboIMAGE DBchange/V

KSAM/V

KSAM is an acronym for Keyed Sequential Access Method. KSAM/V provides a disc file structure for organizing and accessing records in a file according to the contents of key fields. It is Hewlett-Packard's implementation of indexed sequential processing. KSAM/V allows you to access records in two ways:

- Sequentially, according to a sequence defined by key values.
- Randomly, by finding the record that has a field matching a specific key value.

A KSAM file consists of two associated MPE/iX files:

- A data file containing all of the file's data records
- A key file, containing one or more balanced trees that maintain primary and alternate logical sequences for the data records.

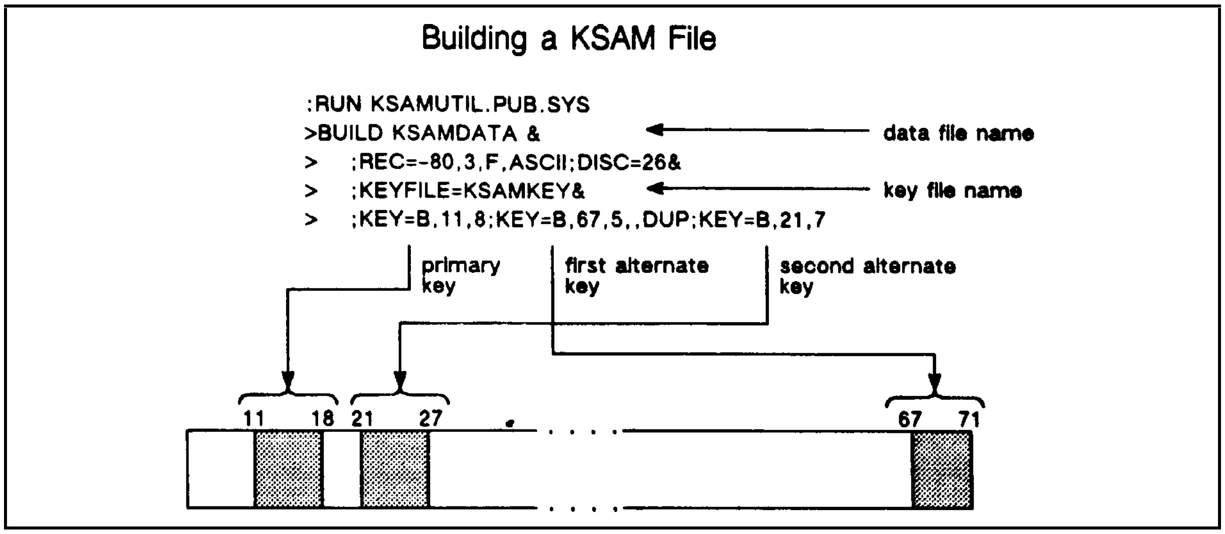
The data file contains the actual data. It can consist of fixed-length or variable-length records. Each record in the data file contains one primary key field and may have a maximum of 15 alternate key fields. Data records are stored in a KSAM file in chronological order (this is the order in which they were written to the file), without regard to key sequence. In KSAM mode, you can specify standard access if you wish. In addition to KSAM mode, you can specify standard (NOKSAM) mode.

The associated key file must be used to access the data records.

KSAM procedures add, delete, read, and update KSAM records and are callable from user applications.

KSAMUTIL allows you to manipulate, verify, and analyze KSAM files. Because KSAM file structure is different from MPE/iX file structure, you cannot create a KSAM file with the MPE/iX command `:BUILD`. You can rename it with the MPE/iX command `:RENAME`, but it is inadvisable because it destroys the cross reference between the data file and the associated key file. Therefore, it is extremely useful to have `KSAMUTIL` commands that are designed to operate specifically on KSAM files and preserve the proper cross referencing.

Figure 7-1 shows an example of building a KSAM file.



```
:RUN KSAMUTIL.PUB.SYS
>BUILD KSAMDATA &
>   ;REC=-80,3,F,ASCII;DISC=26&
>   ;KEYFILE=KSAMKEY&
>   ;KEY=B,11,8;KEY=B,67,5,,DUP;KEY=B,21,7
```

Figure 7-1. Building a KSAM File

The KSAMUTIL command BUILD requires the following parameters:

- Data file name.
- Key file name.
- One or more key types, with corresponding key location, and key size.

The first key specified becomes the primary key. Each subsequent key specified becomes an alternate key. Notice in the example that the first alternate key allows records to have duplicate values in that field. The fourth subparameter of the `KEY` field is the key blocking factor. Default blocking is one record per block, which usually results in wasted disc space. You can find the blocking factor by dividing the sector size (256) by the record length and then rounding down.

FCOPY (KSAM Options)

You can use FCOPY with KSAM options that make it perform correctly for a KSAM file instead of an MPE/iX file. FCOPY provides many file manipulation services. For example, you can use it to:

- Load a KSAM file by copying data from an MPE/iX or a KSAM file into a KSAM file.
- Reorganize a KSAM file by building another KSAM file with keys specified in a different order and copying the old KSAM file into the newly built one.
- Convert an ISAM (Indexed Sequential Access Method) file to a KSAM file by building a KSAM file and copying the magnetic tape containing the ISAM file data to the newly built KSAM file.
- Copy only active KSAM records (data records that are not logically deleted by having a flag in the first word). This feature allows you to physically delete KSAM records from a file and compact the file size. FCOPY defaults to copying only active records unless you specify otherwise.
- Recover logically deleted records by using the `:NOKSAM` option.
- Write logically deleted records to a separate file by using the `:NOKSAM` option and specifying `;SUBSET=#%377,%377#,1`.
- Write any KSAM file to `$STDLIST` or a line printer. Thus, during a session, you can either display the file on the terminal or list it out on a line printer. During a job, you can list it on a line printer.
- Copy data in a KSAM file to an MPE/iX file in chronological, primary, or alternate key sequence.
- Recover corrupt key files or files with missing key files.
- Recover anomalies resulting from system interrupts by specifying `;KEY=0`.

The FCOPY utility has three KSAM options:

- `;KEY=nn`

  - if omitted, copies in primary key sequence, retaining old key trees.

  - if *nn*=0, copies in chronological sequence and rebuilds key trees.

  - if *nn*>0, copies in sequence by key number *nn*, where the key numbers are assigned in order of occurrence.

- ;NOKSAM
  - copies all records in chronological sequence, including deleted records.
  - if omitted, only active records are copied.
- FCOPY default, which copies active records in the primary key sequence

When using the KSAM options of FCOPY, consider the following information:

- KEY and NOKSAM are mutually exclusive options; use them only when the FROM file is a KSAM file.
- NOKSAM processes the FROM file as an MPE/iX file. Use it to copy all data in a KSAM data file, including the logically deleted records.
- Do not use the NOKSAM option to copy a KSAM file with variable-length records.
- If you use the NOKSAM option to reload a file due to keyfile corruption, you must also specify NOUSERLABELS. Using the SUBSET option ensures that only valid (non-deleted) records are copied.
- When copying an existing KSAM file, you can let FCOPY create a TO file for you by using the syntax:

  ```
  >FROM=KSAMDATA;TO=(NDATA,NKEY)
  ```

  where NDATA and NKEY are previously non-existent KSAM data and key fields.

Figure 7-2 shows an example of loading a KSAM file. The diagram shows how the key file is built on KSAM file information specified in the KSAMUTIL command BUILD.

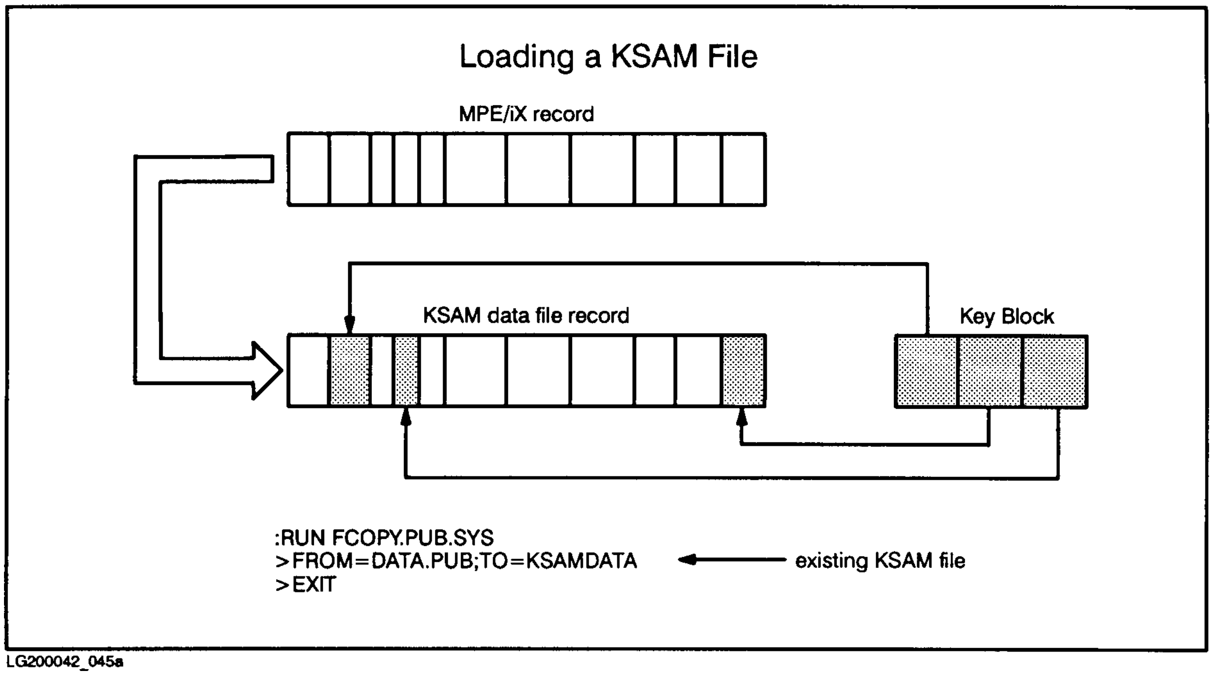


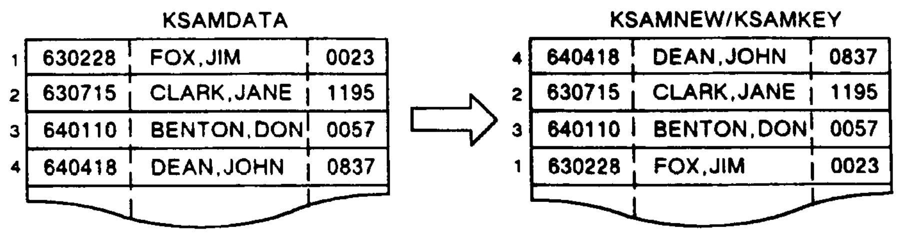
Figure 7-2. Loading a KSAM File

The steps required to reorganize a KSAM file are:

1. Use KSAMUTIL to build a new KSAM file with new key fields.
2. Use FCOPY to copy the old KSAM file to the new one, specifying whichever key in the old file is the primary key in the new file.
3. Use KSAMUTIL to purge the old file and rename the new copy to the old file name.

Figure 7-3 shows an example of reorganizing a KSAM file.

Reorganizing a KSAM File



KSAMDATA

1	630228	FOX,JIM	0023
2	630715	CLARK,JANE	1195
3	640110	BENTON,DON	0057
4	640418	DEAN,JOHN	0837

KSAMNEW/KSAMKEY

4	640418	DEAN,JOHN	0837
2	630715	CLARK,JANE	1195
3	640110	BENTON,DON	0057
1	630228	FOX,JIM	0023

```
:RUN KSAMUTIL.PUB.SYS
>BUILD KSAMNEW;REC=-80,3,F,ASCII;DISC=26&
>  ;KEYFILE=KEYNEW&
>  ;KEY=B,21,7;KEY=B,67,5,,DUP
>EXIT

:RUN FCOPY.PUB.SYS
>FROM=KSAMDATA;TO=KSAMNEW;KEY=21
>EXIT

:RUN KSAMUTIL.PUB.SYS
>PURGE KSAMDATA
KSAMDATA.KSAMNEW, KSAMKEY PURGED
>RENAME KSAMNEW, KSAMDATA
>RENAME KEYNEW, KSAMKEY
>EXIT
```

Figure 7-3. Reorganizing a KSAM File

KSAM procedures

KSAM procedures allow you to programmatically add, delete, read, and update KSAM files. The procedures vary depending on the language in which the program is written. You can access KSAM files through KSAM procedures in the native languages that run on the 900 Series HP 3000:

- HP C/iX
- COBOL II/XL
- HP FORTRAN 77/iX, through calls to intrinsics (for example, HPFOPEN and FOPEN).
- HP Pascal/iX, through calls to intrinsics (for example, HPFOPEN and FOPEN).

For detailed information on using KSAM procedures in HP C/iX, HP Pascal/iX, HP COBOL II/XL, and HP FORTRAN 77/iX, refer to *KSAM/V Reference Manual* (30000-90079).

ALLBASE/SQL

ALLBASE/SQL is designed for HP 3000 900 Series systems. ALLBASE/SQL contains both a network model database management system, called TurboIMAGE, and a relational model database management system, called HP SQL. It has been optimized for MPE/iX systems. HP SQL, the relational model database management system used when MPE/iX is running in Native Mode, is discussed in the subsection below. TurboIMAGE/XL is discussed later in this chapter.

HP SQL

HP SQL is the relational model database management system (DBMS) module of ALLBASE/SQL. (Table 7-1 shows ALLBASE/SQL specifications useful to programmers. For detailed information on ALLBASE/SQL components, refer to the Data Management Series.)

HP SQL allows views to be created. A view is a table derived by defining a filter over one or more tables to let users or programs view only certain data in the tables. Views improve security by allowing users to access only the data they have a need to know. Since the view is not actually a physical table, use of views does not result in redundant data. When data in a table is updated, all views that use the data are automatically updated.

An HP SQL query requires that the programmer specify only the data needed, minimizing the amount of preplanning and coding necessary. A non-procedural interface allows the query of HP SQL databases without specifying data access path information. You can operate on entire sets of data at one time, rather than on one record at a time. The major features of HP SQL are:

- Relational data model allows you to specify required data, without specifying the retrieval method.
- Language preprocessors allow the same statements that are used for an interactive query to be embedded in HP Pascal/iX and HP COBOL II/XL application programs.
- Concurrent access allows multiple users to simultaneously access data.
- Specification of levels of access privileges maintain security for users and groups of users.
- Data independence allows you to make changes to the database structure without requiring modification of applications.
- You can define *views* that allow a user or group of users to see parts of one or more tables as a single, virtual, table. These act as a filter to customize a table.
- Query optimizer reduces requirement for query planning details.
- B-tree indexes support fast data access.
- User controlled transactions ensure that data is always in a consistent state.
- Automatic locking ensures data integrity in a multiuser environment by preventing access to data while it is being updated.
- Automatic rollback recovery preserves logical data integrity due to a soft crash.
- You can invoke rollback capability in a program to allow erroneous data, usually generated in an on-line situation, to be removed before the transaction is completed.

- Rollforward recovery preserves logical and physical data integrity due to a hard crash.
- Dynamic restructuring allows you to change data structure, table capacities, and security without unloading and loading the database.
- Null data values allow use of fields that are relevant to some, but not all, records in a table.

HP SQL does not require you to define explicit relationships between data sets. When you perform a query, it determines relationships by matching values between fields common to two or more data sets. If you consider a data set to be analogous to a table, a given field in the data set would occur as a column in the table. Data from any number of tables that share a common column can be related, as needed, and you can define them to be an HP SQL database. The query optimizer determines the best data access strategy based on factors such as the presence of indexes and the relative sizes of accessed tables.

Interactive SQL (ISQL) is the interactive interface to HP SQL. It provides the functionality of a data definition language (DDL) and a data manipulation language (SQL). DDL allows the database administrator to control all aspects of database creation and modification. SQL allows the programmer or frequent user to interactively query a database.

Security

Security is maintained by allowing specification of appropriate levels of access privileges to individual users or groups of users.

HP SQL allows read (SELECT) access and write access privileges to be assigned at the table level. Read or write access restriction at a finer granularity than the table level, such as at the column level, may be obtained by defining a view of the table that omits the sensitive information. Modification (UPDATE) authority may be granted at the column level without requiring that a view be specified. Write access may be assigned to allow any combination of the following capabilities:

- Row modification (UPDATE)
- Row insertion (INSERT)
- Row deletion (DELETE)

The database administrator (DBA) assigns access privileges by grouping users with common access needs into authorization groups. The particular read and write authorities are then assigned by the DBA to each of these groups. Users with unique access requirements may also be granted privileges directly.

Logical Transaction

A logical transaction is a series of database modifications of which either all or none must be performed to leave the database in a consistent state. The particular grouping of modifications defined by the user to be a transaction varies depending on the particular application. An example of a transaction is an accounting entry to pay a bill: the credit to the cash account and debit to the accounts payable account must both be performed to avoid leaving the accounts out of balance. This concept of a logical transaction is essential to ensuring that data integrity is maintained when multiple users are concurrently accessing the database or in the event of system failure.

Concurrency

HP SQL preserves data integrity, when multiple users are accessing a database, through a comprehensive locking scheme based on the transaction concept. When a user begins a transaction, a lock is automatically granted for each page read or modified by the transaction. (A page is a unit of data storage that contains 4,096 characters.) This ensures that no one else may update the data on those pages while the user is reading or updating them. If data is only being read, then other users are not prevented from reading it simultaneously; they just can't update it. If data is being updated, however, data integrity is ensured by preventing the other users from reading or updating the data. When the user's transaction is completed, all acquired locks are automatically released.

Tables (data sets) may also be blocked explicitly. This feature is provided to allow the programmer greater flexibility in applications where it is advantageous to lock large portions of the database. Since explicit locks reduce concurrency, they are not recommended for general use.

Recovery

HP SQL ensures that the logical and physical integrity of the database is protected in the event of a program abort, system failure, or destruction of the media on which the database resides.

Rollback recovery is an automatically activated recovery feature that ensures that the database is always in a logically consistent state. HP SQL logs write transactions to a log file on disc. In the event of a system failure or program abort, HP SQL uses this log file to automatically back out any partially completed transactions.

The rollback capability may also be invoked in an HP SQL program. This is a particularly valuable feature in an on-line application, as it allows a user who has entered incorrect information to nullify the transaction before its completion.

Rollforward recovery protects the physical and logical integrity of the database against media failure. In the event of a hardware or software failure, the transactions from the log file are reapplied to a backup copy of the data to bring it up to the current state.

A simultaneously updated copy of the log files used for rollback or rollforward recovery may be kept on another disc to provide additional protection in the case of disc failure.

Database Creation

As part of the database design process, the database administrator (DBA) must decide how many databases should be included in each HP SQL DBEnvironment. A DBEnvironment may contain one or more databases. Since the DBEnvironment is the maximum scope for recovery, multiple databases that will be accessed by way of a single logical transaction should be placed in the same DBEnvironment. The DBEnvironment is also the level at which the data is backed up. Therefore, unrelated databases should be placed in separate DBEnvironments.

After the DBA has designed the database structure on paper, he may easily create the database. A DBEnvironment must be configured for the database unless it will be included with other databases in an existing DBEnvironment. The remaining step in creating an HP SQL database is to create the tables (CREATE TABLE command), indexes (CREATE INDEX command), and views (CREATE VIEW command) that make up the database.

Database Restructuring

HP SQL provides a full set of database restructuring capabilities. HP SQL supports dynamic restructuring for commonly required structural changes. Dynamic restructuring allows users to continue to access data, except for the affected areas, during restructuring.

HP SQL provides dynamic restructuring for the following cases:

- Expanding table capacities
- Altering security designations
- Adding columns
- Adding or deleting indexes, views, and tables.

Table 7-1. ALLBASE/SQL Specifications

Specifications	HP SQL
Data Types:	Packed decimal (IEEE standard) double-precision floating point (8 byte)
	Integer (2 & 4 byte)
	Fixed length (<3996 bytes) Character
	Variable length (<3996 bytes) Character
Languages:	COBOL Pascal
Passwords/Security Groups	Unlimited Groups
Maximum Database Parameters	
Tables (sets) per database	Unlimited
Records per table (set)	Unlimited
Record length	3996 bytes
Columns (items) per table (set)	255
Field (item) length	3996 bytes
Sub-items per item	n/a
Children per parent	n/a
Indexes per table	Unlimited
Search items per set	n/a
Columns per index	15
Sort items per path	n/a

TurboIMAGE/XL Data Base

A database is a collection of logically related data in one or more files. The files contain both the data itself and structural information on how one piece of data relates to another. Pointers in the database allow access to the actual data and indexing across the multiple files in the database. A network database ties together fields containing information that may be relevant to a single transaction even though they physically reside in different records.

The terminology used to describe a database differs from that used to describe MPE/iX files. Table 7-2 shows the corresponding terms.

Table 7-2. Data Base Terminology

MPE/iX	Network Data Base	Relational Data Base
field	data item	column
record	data entry	row (or tuple)
file	data set	table
file definition	schema	- - -
collection of files	database	database
application	database applications	database environment

A network database consists of data sets. The structure of the database is a schema, which the user writes. The schema is processed and kept in a special file called a root file, which is also part of the database. The term database application refers to a complete data processing application using a database management system and a database.

The database described in this chapter is a TurboIMAGE/XL data base. A TurboIMAGE/XL database has two types of data sets:

- Detail set, which is a collection of related data entries, each of which contains one or more a forward pointers, one or more backward pointers, and the data itself. A standalone data set contains no pointers. TurboIMAGE/XL uses the pointer information to tie together all data entries whose search items have the same value.
- Master set, which maintains indexing into detail data sets or into itself, if it is a manual master set. Indexing consists of a key data item.

A given network database can have many detail sets and many master sets.

A key data item is frequently called simply a key item. It uniquely identifies the related data entries in a master data set. A search item value is not necessarily unique. In a master set, a key item is the data item used to perform the search.

Master Set

Each master set entry contains a key item pointing to a subset of detail set entries, all with related values in a particular data item. Thus, you can quickly reference any subset of entries by finding its master key item (index pointer). Each master set can contain values for one detail data item to search. (In other words, it can have only one field as a key item.) However, each master set can provide indexing for a maximum of 16 detail sets by using a key item in the master set as an index for a maximum of 16 detail sets. Each detail set can be linked to 16 master sets and can contain a maximum of 16 searchable data items. A master set can be automatic or manual.

Chain Head

In a master set data entry, the key item and the data items accompanying it are called a chain head. It consists of:

- A count number indicative of the number of detail entries with matching key item values.
- The pointer to the last data entry in the detail set whose key item value matches the value of the master set key item.
- The pointer to the first data entry in the detail set whose key item value matches the value of the master set key item.
- The data itself.

In the detail set, the first and last data entry described by the pointers in the chain head are the beginning and ending of the chain of data entries in the detail set that have the same value for the associated data item as the master set key item. A chain can have a maximum of 2**31 data entries. If more than one master set references a particular detail set, additional pointers are made available in the detail set.

A chain can be searched in either a forward or backward direction. This is convenient, for example, in an application that adds new invoice numbers to an existing data set. Otherwise, you can follow the pointer directly to the last entry in the chain and add a new entry using a minimum of disc overhead. New entries are automatically added at the end of the chain, unless the chain is sorted. Although pointers occupy space, usually a database takes less space than the MPE/iX files it replaces. This is because the reduction in data redundancy can outweigh the internal overhead of maintaining a TurboIMAGE/XL database. Figure 7-4 shows an example of how chain heads in a master set are used in TurboIMAGE/XL data set organization.

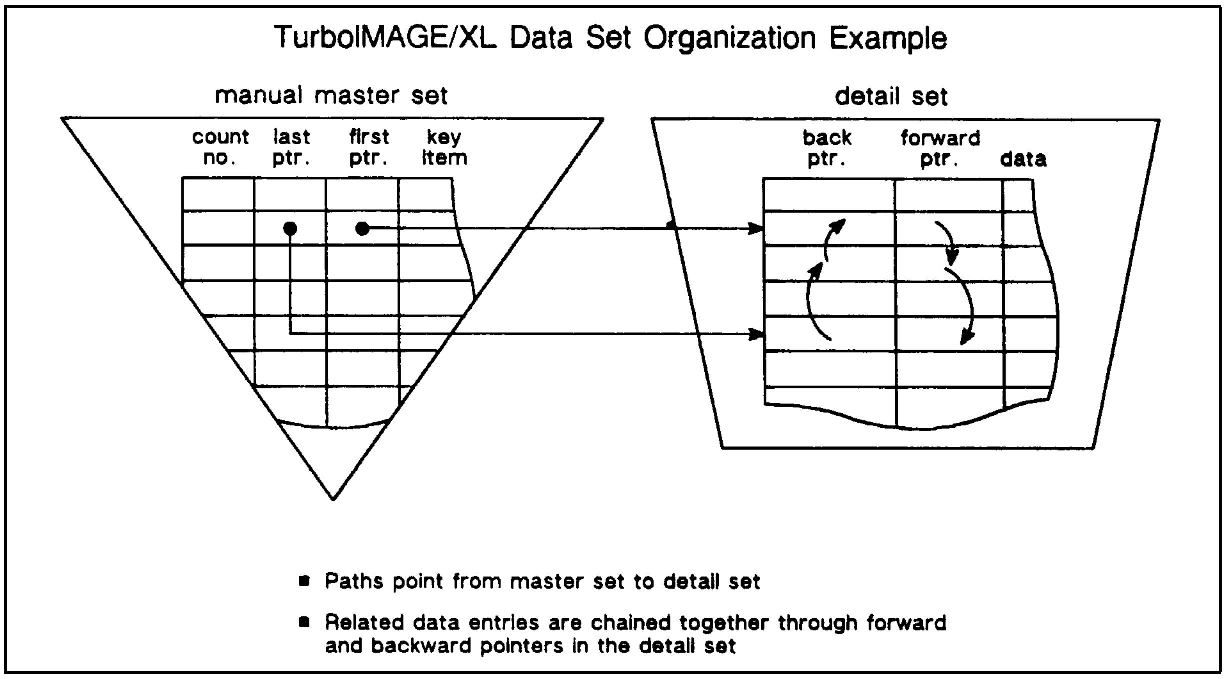

TurboIMAGE/XL Data Set Organization Example
manual master set
detail set
count no.
last ptr.
first ptr.
key item
back ptr.
forward ptr.
data
■ Paths point from master set to detail set
■ Related data entries are chained together through forward and backward pointers in the detail set

Figure 7-4. TurboIMAGE/XL Data Set Organization Example

Automatic Master Set

The characteristics of an automatic master set are:

- Each data entry contains a chain head.
- A data entry (record, in MPE/iX terms) can contain only the key item used as an index pointer.
- TurboIMAGE/XL automatically updates the master set when modifications are made to the related detail sets. Changing a search item requires deleting the old one and adding the new one. The master set is automatically done.
- Use of an automatic master set saves time when key item values are unpredictable or so numerous that it is not expedient to make manual additions and deletions of master data entries. However, it may introduce data redundancy.

Manual Master Set

A manual master set data entry can contain not only a key item, but additional data items that provide information in addition to the key item. The characteristics of a manual master set are:

- Each data entry contains a chain head.
- You must explicitly add or delete all data entries.
- You must add a key item in a master set data entry before adding a related detail set data entry.
- You must delete all related detail sets before adding a master set entry.
- Allows control over data entries that an automatic master set does not.

TurboIMAGE/XL and QUERY/V

Together, TurboIMAGE/XL and QUERY/V provide a database management system. TurboIMAGE/XL is a set of programs and procedures you can use to define, create, access, and maintain a database. QUERY/V is a database inquiry facility that provides a simple method of accessing a TurboIMAGE/XL database without requiring programming. You can use QUERY/V interactively.

TurboIMAGE/XL components are:

- DBSCHEMA, a subprogram that creates a root file.
- DBUTIL, a subprogram that you can use to create, purge, or report on a database, erase data from data sets, and enable and disable logging and recovery options.
- TurboIMAGE/XL procedures, a set of library routines that allow loading, accessing, and manipulating of data in a database.

Creating a Schema

You can use DBSCHEMA to create a root file containing the structural definition of a database by using a schema. A schema analyzes data and formalizes data relationships. The schema is a summary of all the data relationships in a data base and stored as a separate text file. The format used in a schema is TurboIMAGE/XL's database description language. You can enter schema information in an MPE/iX text file, following the database description language format. Schema information includes:

- Password and user class
- Designation of read/write capability for data sets
- Data items
- Designation of read/write capabilities for data items
- Manual master set
- Automatic master set
- Detail set
- Paths from master sets to detail sets
- Key items, identified by path counts
- Capacity for each set (specification of a prime number is recommended for a master set)

Creating a Root File

Once you have created a schema in an MPE/iX text file, you can use the schema processor, DBSCHEMA, to create a root file and store in it an internal description of the database, based on the schema in the text file. The root file is part of a database and contains all the database structural information required by TurboIMAGE/XL and QUERY/V.

The syntax for the command to run the schema processor is:

```
    :RUN DBSCHEMA.PUB.SYS;PARM=n
```

where:

if `PARM=` is omitted, the formal designator for the input file, `DBSTEXT`, is equated to `$STDLIST`. You would use this option, for example, if you wanted to enter the schema directly from your terminal during a session and list the output to your terminal (in addition to creating a root file).

`PARM=1` if `DBSTEXT` has been equated to an actual file designator

`PARM=2` if `DBSLIST` has been equated to an actual file designator

`PARM=3` if both `DBSTEXT` and `DBSLIST` have been equated to actual file designators

Figure 7-5 shows an example of creating a root file. In this case, both `DBSTEXT` and `DBSLIST` are equated to actual file designators: `DBSTEXT` is an existing MPE/iX text file and `DBSLIST` is the line printer.

Creating the ROOT File

```
:FILE DBSTEXT=EDTEXT
:FILE DBSLIST;DEV=LP
:RUN DBSCHEMA.PUB.SYS;PARM=3
```

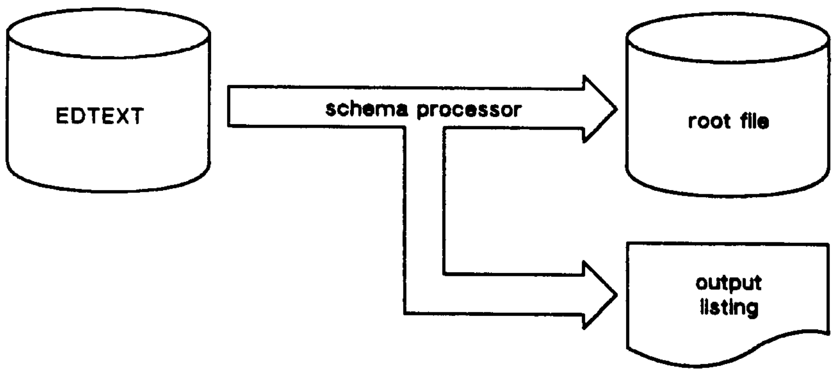


Figure 7-5. Example of Creating a Root File

DBUTIL

DBUTIL is a TurboIMAGE/XL subprogram that has many capabilities. You can use it to:

- Create a database
- Purge a database
- Erase a data set
- Display information about a database
- Enable and disable logging and recovery options
- Activate, deactivate, and report on a database access file

You must be the creator of the database or know its maintenance word to use the DBUTIL commands `PURGE` and `ERASE`. The maintenance word is an optional ASCII string that you can specify with the DBUTIL commands `CREATE` or `SET`. It defines a password to use for executing DBUTIL commands, such as `PURGE` and `ERASE`, and to operate other database procedures. If no maintenance word is defined, then only the database creator can operate them.

The DBUTIL command `PURGE` purges the root file and all of the data sets in the database. (You cannot use the MPE/iX command `:PURGE` to purge a root file, detail set, or master set.

TurboIMAGE/XL has a transaction logging and recovery system with the capability of recovering a database from a transaction-oriented log file in the event of a system failure. DBUTIL allows you to enable or disable these options.

Creating a Data Base

Once you have created a root file, you can use DBUTIL to create the database described in the root file. To create the database, you must be logged onto the system with the same log-on used to create the root file. DBUTIL creates data sets according to the specifications in the internal description of the schema. When created, data sets are initialized to zero and contain no data.

Figure 7-5 shows an example of creating a database. In this case, the manual master set, `CUSTOMER`, has a primary key item named `ACCOUNT` pointing to the detail set `SALES`. The automatic master set, `DATE-MASTER`, has a primary key item named `DATE` that also points to the detail set `SALES`.

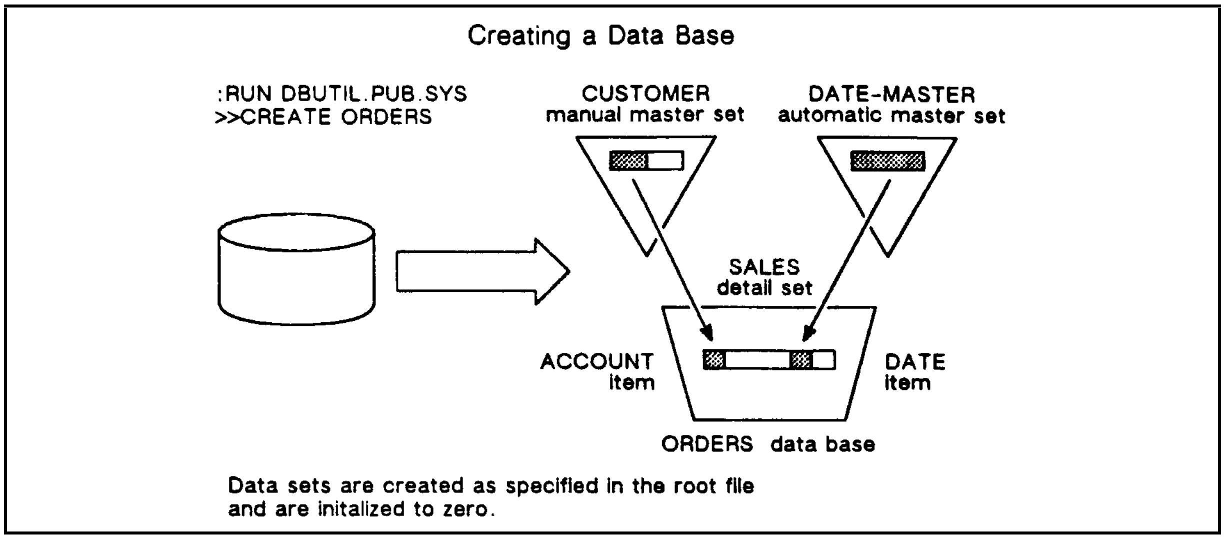


Figure 7-6. Creating a Data Base

TurboIMAGE/XL Procedures

TurboIMAGE/XL procedures are a set of library routines that allow you to load, access, and manipulate data in the database. You can call them from HP C/iX, COBOL II/XL, HP FORTRAN 77/iX, and HP Pascal/iX application programs.

You must be the creator of the database or know its database password word to use TurboIMAGE/XL procedures. (The database password is an optional ASCII string that you can specify with the DBUTIL commands `CREATE` or `SET`. It defines a password to use for executing TurboIMAGE/XL procedures and DBUTIL commands, such as as `PURGE` and `ERASE`. If no database password is defined, then only the database creator can operate them.) The database creator can use a semicolon (;) as a database password to bypass all internal database security. You need not be logged onto the same group and account that contains the database root file and the data sets. You can perform these procedures on a remote database. TurboIMAGE/XL procedures are in the following categories:

- Database access

  `DBOPEN`

```
DBCLOSE
DBLOCK
DBUNLOCK
DBCONTROL
```

- Access data

```
DBINFO
DBGET
DBPUT
DBUPDATE
DBDELETE
```

- Information and status

```
DBINFO
DBEXPLAIN
DBERROR
```

- Logging

```
DBBEGIN
DBMEMO
DBEND
```

TurboIMAGE/XL subprograms are used for storing and loading. You must be the creator or know the maintenance password to use them and you must be logged onto the account and group where the database resides. The subprograms are:

```
DBSTORE
DBRESTOR
DBLOAD
DBUNLOAD
```

Backing Up or Restructuring a Data Base

`DBSTORE` and `DBRESTOR` copy a database to and from magnetic tape or serial disc. They copy the entire database, including the root file. You can use `DBUNLOAD` and `DBLOAD` to assist in restructuring a database, but they copy only data, not the root file or the data set structures.

Changing an Existing Data Base Design

You can change the design of an existing database without writing special programs to transfer data from the old one to the new one. To restructure a database, follow these steps:

1. Run `DBUNLOAD`, specifying the old database.
2. Use the DBUTIL command `PURGE` to purge the old database.
3. Redefine the database using the same database name and use DBSCHEMA to create a new root file.
4. Use the DBUTIL command `CREATE` to create and initialize the new data sets.
5. Run `DBLOAD` on the new database to put the old data into it.

For detailed information on the types of design changes you can make using this method, refer to *TurboIMAGE/XL Reference Manual* (30391-90050).

Recovering a Data Base

TurboIMAGE/XL is designed to maintain the integrity of its data bases. However, it is possible that data or structural information can be lost during a hardware failure or an operating system crash. It is highly recommended that you regularly backup a database.

You can maintain the database by copying it to magnetic tape using `DBSTORE` at regular intervals. If necessary, you can restore the database using `DBRESTOR`. This restores the database to its state at the time it was last stored. You can use this method more often than system backups are done to maintain recent copies of the database and minimize the number of transactions that may be lost.

To recover changes made after the last backup, you must do one of the following:

- Rerun all jobs that modified the database since the last time it was stored.
- If transaction logging was enabled, run the DBRECOV utility.

You can execute the transaction logging and recovery system to return a database to a state near that at the time of system failure. The logging system provides a mechanism to log database transactions to a logfile on magnetic tape or disc. If you must restore the database, first restore the backup data base copy and then run the recovery program, DBRECOV. This re-enters transactions from the log file against the data base. It also allows you to create individual user recovery files, providing information to users that enables them to figure out where to resume transactions.

The database administrator is responsible for enabling and disabling the logging and recovery processes and generating backup database copies. This makes logging a global function controlled at the database level, rather than at the individual user level.

QUERY/V

QUERY/V is a database management subsystem used for retrieval and reporting of data. You can use it to retrieve, report on, update, add, and delete data in a database and display the database structure. It provides you with the ability to:

- Inquire into a database without writing special programs
- Make low-volume data modification and load new data
- Generate reports comparable to those that many languages produce
- Debug new application programs

The report generation capability of QUERY/V allows you to select data on a basis of compound logical comparison. Once selected, you can control which items are reported and how they are formatted on the report.

You can run QUERY/V by entering:

```
:RUN QUERY.PUB.SYS
```

Frequently used QUERY/V commands are:

- `HELP`, which explains QUERY/V commands
- `DEFINE`, which prompts for needed environment information

- `FORM`, which displays a database structure
- `LIST`, which displays data in a data set
- `FIND`, which searches a data set for data items
- `REPORT`, which displays data in a data set located by the FIND command in the format you specify
- `XEQ`, which executes QUERY/V commands stored in a text file
- `EXIT`, which terminates QUERY/V execution

Table 7-3 shows a comparison of data management considerations to help you determine whether to set up a data base with TurboIMAGE/XL or KSAM.

The preferred choice of KSAM over TurboIMAGE/XL for generic key retrieval is based on the fact that KSAM automatically handles partial key searches. A TurboIMAGE/XL database requires defining an additional master set and detail set to handle the partial key search, as well as requiring some additional programming.

Table 7-3. Data Management Considerations

	TurboIMAGE/XL	**KSAM**
Heavy sequential processing	no	yes
Unanticipated inquiries	yes	use `FCOPY`
Program-data independence	yes	no
Easy conversion from ISAM	no	yes
Privacy and security	yes	file level only
Privileged files (protected)	yes	no
Variable-length records	no	yes
Field access by name	yes	no
Generic (partial) key retrieval	no	yes
Sorted retrieval by key	yes	yes

TurboIMAGE DBchange/V

DBchange/V is the interactive utility that allows dynamic restructuring and capacity expansion of a TurboIMAGE database. Database changes are input using a menu format and executed either on-line or in batch mode.

DBchange/V allows easy screen access and input of database restructuring requests for TurboIMAGE databases. Because multiple restructuring changes can be requested during one DBchange/V session, there is no need to run separate programs for each restructure feature. DBchange/V allows you to review multiple change requests before actual updates are processed.

DBchange/V provides the database administrator with an important tool to aid in the use and support of TurboIMAGE databases. Additional benefits include ease of use and batch capabilities. DBchange/V also includes the DICTDBA, DICTDBU, and DICTDBL database utilities, which facilitate the control, maintenance, and restructuring of TurboIMAGE databases.

Key DBchange/V Features

Key DBchange features include:

- Interactive forms access allows flexible database manipulation.
- Two-stage update process allows input and review of changes before actual update.
- Multiple changes can be made to the same database in one DBchange/V session.
- Database restructure requests are input and stored in a change file. DBchange/V uses the change file to process database modifications.
- Modifications can be processed immediately or scheduled for later batch processing.
- TurboIMAGE database modifications can be made in real-time, without requiring a DBUNLOAD and DBLOAD.
- On-line help facility provides information about commands and parameter requirements.
- The Root File is checked for path information inconsistencies, allowing DBchange/V the option to make corrections.

Interactive DBchange/V commands allow the user to:

- Rename a database
- Copy a database
- Change database security
- Restructure a database
- Print schema
- Change set capacity and blocking factors

All TurboIMAGE and MPE file security and capacities are respected. DBchange/V verifies that the user has proper access before it allows changes.

DICTDBA

DICTDBA checks the integrity of synonym chains and identifies broken chains. It generates reports on synonyms and chains for master sets and chain statistics for detail sets.

DICTDBU

DICTDBU unloads the contents of an existing TurboIMAGE database. It can unload the database chained to either a disc or tape file, thus organizing the records for efficient chained access. DICTDBU also provides the capability to unload only selected data sets.

DICTDBL

DICTDBL loads the stored contents of a TurboIMAGE database. It simplifies and speeds up the database load function when only a few sets must be loaded.

Index