



hp e3000
solution transition
advisor

**ksam and the
mpe file system**



white paper

table of contents



executive summary	1
overview	1
intrinsic	2
intrinsic and the mpe file system	3
file system functionality	4
file sizes	5
file names	5
mpe file types	8
standard files	8
rio files (relative i/o)	8
circular files	9
message files	9
memory mapped files	10
spooled devices	11
file domains	12
file equations	13
ksam	15
migrating programs that use intrinsic	17
appendix a: more information about intrinsic	19
appendix b: device independence	29
device files	30

executive summary

Because of HP's strategic decision to discontinue support of its HP e3000 platform, HP recognizes its customers are faced with making significant decisions on transitioning their HP e3000 solutions to other platforms. At this time, many questions arise on how to manage this process:

- What is the impact on affected business functions?
- What are the capital and human costs associated with transition efforts, both now and in the future?
- How can risk be mitigated?
- What is technically required for transitioning all or part of the HP e3000 solution?
- What are possible transition strategies?
- What are the appropriate Independent Software Vendors (ISV) migration tools or mix of tools to use in specific components of the HP e3000 software solution?



There are a total of six white papers in the "HP e3000 Solution Transition Advisor" series. Each one addresses the subject of e3000 transitions from a different angle. The subjects of the six white papers are:

- Business Planning Guide
- Technical Planning Guide
- Compilers and User Interfaces
- TurboIMAGE and Databases
- KSAM and the MPE File System (this paper)
- MPE Commands and Networks

overview

This is the fifth in a series of white papers from Hewlett-Packard, providing HP e3000 customers with a transition process roadmap to assess, analyze, select and manage the appropriate transition strategy for their HP e3000 solutions. It is not intended to handle all HP e3000 transition issues, but a guide to:

- Identifying the common uses and attributes of MPE intrinsics.
- Identify and examine the unique characteristics of the MPE/iX file system.
- Identifying the primary MPE file types, and the issues involved in migrating them.
- Identifying MPE file domains, and the issues involved in migrating software that uses them.
- Examining MPE file equations, and the issues surrounding the migration of software that uses them.
- The issues surrounding the migration of applications that make use of KSAM files on the e3000 to other platforms.
- Identifying and examining the alternatives to KSAM that are available for the various target platforms
- Addressing four different ways to address migrating programs that make extensive use of MPE intrinsics.



intrinsic

On the HP e3000, application programs invoke operating system functions using *intrinsic*. Intrinsic are a special kind of *procedure* that allows user-written application programs to perform tasks that are reserved for the operating system to perform, such as the creation and deletion of files. The MPE intrinsic are unique to MPE/iX, which means that applications that make use of intrinsic cannot be ported to other platforms without special considerations.

Intrinsic are, first and foremost, procedures. All HP e3000 programs are made up of procedures. Each procedure is a callable executable that handles some specific recurring task or tasks. The procedures that make up user-written application programs are compiled and linked together using MPE's Link Editor. User-written procedures may be stored in program files, or in executable libraries.

Intrinsic are special procedures which are distinguished from ordinary user-written procedures in two ways:

1. Intrinsic are part of MPE/iX. Unlike user written procedures, they are not stored in program files, but in system libraries such as XL.PUB.SYS.
2. Intrinsic perform specialized tasks that are reserved for the operating system to perform, such as the creation and deletion of files. The analogous components of HP-UX, Linux[®] and Windows[®] are often referred to as "*system calls*".

Intrinsic are the only parts of MPE/iX that can be called by user programs without the use of unsupported programming techniques such as privileged mode. There are a set of tables in Appendix A that list many of the MPE/iX intrinsic, along with pointers to the manuals containing the details.

Application programs may call intrinsic in either (or both) of two ways.

- An intrinsic may be called *explicitly*. In an explicit call, the application's source code contains a specific reference to the intrinsic. For example, a COBOL program might use a COBOL CALL verb to explicitly call the HPFOPEN or FOPEN intrinsic.
- An intrinsic may also be called *implicitly*. In an implicit call, the application's source code contains no explicit reference to the intrinsic. Rather, the compiler generates the intrinsic call as part of the compilation of a standard language statement. For example, the COBOL OPEN verb will cause the compiler to generate a call to the FOPEN or HPFOPEN intrinsic.

When porting applications from MPE/iX to other platforms, only explicit intrinsic calls need to be considered. When a compiler calls an intrinsic implicitly, it can be assumed that the compiler on the target platform will make the system calls that are appropriate on that platform.

However, when an application program makes an explicit call to an intrinsic, the person doing the port needs to carefully analyze what the programmer was trying to accomplish with the intrinsic call, in order to preserve the functionality in the ported program.

intrinsic and the mpe file system



MPE's file system contains a good deal of useful functionality that is unique. This proprietary technology may be valuable from the point of view of applications that will continue to run on the HP e3000. Unfortunately, it makes applications that must be migrated off of the HP e3000 that much more difficult to port. At the time most HP e3000 applications were written, it was considered good programming practice to take advantage of all the functionality that was available on a platform, regardless of its "openness" or availability on other platforms. Porting HP e3000 applications to one of the recommended target platforms (Linux, HP-UX or Windows) requires that you find substitutes or workarounds for any proprietary technologies that the application might be using on the e3000 which are not part of your selected target platform.

A first step toward doing this is identifying explicit references to intrinsic. In Appendix A, there are tables containing lists of the intrinsic that are used to perform various functions, including accessing files, accessing features of the command interpreter, managing processes, and so forth. If an application program contains references to these intrinsic, it's very likely that the program logic also assumes certain proprietary characteristics of the MPE operating system.

This white paper will focus specifically on identifying these proprietary characteristics of the MPE file system, and suggesting substitutes or workarounds were appropriate. The file system intrinsic (listed in Table A1 of Appendix A) are among the most widely used intrinsic on the HP e3000. Similarly, the command interpreter intrinsic (Table A2 of Appendix A), are also widely used, and these will be discussed in the sixth paper in this series, "**HP e3000 Solution Transition Advisor – MPE Commands and Networks**". If your applications makes use of the intrinsic in the other tables listed in Appendix A, it is using functionality that is beyond the scope of these white papers.

There are a number of features of the MPE file system which are significantly different from the file systems of the recommended target platforms. These include:

- MPE's unique file *naming convention*
- MPE's three level *directory structure*
- MPE's support for specialized file *types*
- MPE's support for specialized file *domains*

When porting applications that access proprietary file system technology from MPE/iX to other platforms, it's important to analyze the business problems that the programmer who originally wrote the program was trying to solve.

- Sometimes, the target platform may offer better tools for solving the business problem. In that case, the best course of action is often to replace the intrinsic references with code that uses these tools.
- If not, the best course of action may be to use an MPE emulation package. With an emulation package, the intrinsic calls can be left in place in the ported program. MPE emulation packages provide functionality on the target platform that's similar to that provided by the intrinsic on the e3000, using identical API's.

file system functionality



file system functionality

The file system is the part of the MPE/iX operating system that manages the transfer of information between application programs running on the HP e3000 and peripheral devices such as disks. It handles input/output operations such as passing of data to and from user processes, compilers, and data management subsystems.

MPE files are record based. This means that the data in each file has a structure, and the operating system is aware of this structure. Application programs are responsible for maintaining this structure, as follows.

1. Application programs writing data arrange it into data elements called **fields**. Each field is a string of bytes. The fields are grouped together into **records**¹. Data is written to the output file one record at a time.
2. An application program reading the data reads it one record at a time. As each record is read, its fields are processed by the application program.
3. A **file** is a group of logically related records which may be kept on any storage medium or sent to any input/output peripheral device. MPE applications are **device independent**, a feature which cannot be taken for granted on other platforms. For more information about device independence, see Appendix B.

Each MPE file is divided into records. The length of each record is user-definable; it may be fixed or variable. That is, each record may have the same length, or the records in a file may vary in length. The file system is aware of this file structure and ensures that data is read and written only in record-sized blocks. If your program tries to write a single byte to an 80 byte record, the file system will write the entire 80 bytes, padding the single data byte with blanks or binary zeroes².

The files on HP-UX, Linux and Windows are byte streams. There is no record structure known to the operating system. Each file is handled by the operating system as a continuous stream of bytes. By default, when a UNIX[®] application writes a string of bytes to a file, the string is delineated by a “new line” character. Since there is no concept of record length, when a program writes a single byte to the file, the byte is followed by the “new line” character. No padding is done, as would be the case on MPE/iX. This may result in a more efficient use of storage space, but it can be a headache for programmers trying to port code from MPE/iX to one of the recommended target platforms.

If your programs use standard COBOL statements to do disk I/O, most of these issues will be handled by the compiler. The compilers on the recommended target systems understand the use of byte stream files and handle the data appropriately. This makes it comparatively easy to port software that uses standard COBOL I/O statements such as READ and WRITE. However, when porting the *data* from the HP e3000 to your target platform, it may be necessary to convert the files to the byte-stream format. There are utilities available to do this. Similarly, if your program contains explicit references to MPE intrinsics, and you choose to use an MPE emulator, the record structures used on MPE will be retained in byte stream files.

¹ On MPE, files have a “column-and-row” structure that’s similar to the structure of a relational database. Each record corresponds to a row. Each field corresponds to a column. This differentiates MPE files from the byte-stream files found on UNIX and Windows. Byte stream files have no record structure beyond that imposed by an application program. If a relational database is going to be used in your transition project, you may want to consider using it to store data that was stored in files on the HP e3000.

² When an MPE file is created, it must be declared to be either an ASCII file or a BINARY file. If the file is declared to be ASCII, short records will be padded with ASCII blanks (hex “20”s). If it’s declared to be BINARY, short records will be padded with binary zeroes.

file sizes

file sizes

MPE artificially restricts the size of files; this is different from the behavior of the file systems of the recommended target platform. For example, if you create a file on a typical UNIX system, there is, (by default) no limit to the amount of data that can be written to that file.

On MPE, when you create a file, you must specify a *filesize*. If you fail to specify one, a default value will be assumed. This *filesize* represents the maximum number of records that can be written to that file. If you create a file on an MPE system, you cannot write more records to the file than the limit specified in the *filesize*.

On the recommended target platforms, limits can be imposed on the amount of data that can be written to a file (or files), but the limits are not imposed at the file level. For example, HP-UX systems permit the system administrator to impose quotas at the directory level, limiting the number of bytes that may be contained by *all* the files in a given directory.

file names

One of the most primitive features of the MPE file system is the file naming convention and directory structure. These features date back to the original design of MPE in the 1970s. The 16-bit “classic” models of the HP 3000 imposed the following restrictions on files and the file system.

- All MPE file names were restricted to no more than 8 characters.
- File names were case insensitive. On MPE the file names “myfile” and “MYFILE” are equivalent.
- All files were contained in a two level directory structure. UNIX and Windows have free-form directory structures in which users may contain directories (or folders), which in turn may contain other directories, and so on. By contrast, on MPE the entire file system was divided into structures called ACCOUNTS. The accounts, in turn were divided into structures called GROUPS. All files had to reside in these groups. Groups could not contain other groups; the entire structure were restricted to two levels.
- Like file names, group names and account names were restricted to 8 case insensitive characters.
- A fully qualified MPE file name was represented by the file name followed by the group name followed by the account name all separated by periods. For example FILE.GROUP.ACCOUNT represents a file named FILE residing in a group called GROUP, which in turn resides in an account named ACCOUNT. Fully qualified file names are analogous to UNIX path names, but they must conform to this three-level syntax.

In the mid 1990s, HP enhanced the MPE operating system by adding POSIX extensions. Support for UNIX style directory structures became part of MPE/iX. MPE/iX³ calls this feature the Hierarchal File System or HFS.

- Files and directories were allowed to have names longer than 8 characters.
- As on UNIX, file and directory names became case sensitive — so that the names “myfile” and “MYFILE” could be used to represent two different files in the same directory.

³With the addition of the POSIX extensions, the operating system was renamed to MPE/iX.

- Path names could be used to refer to files in much the same way as they were used on UNIX, with file names and directory names separated by slashes. For example, a file with the fully qualified MPE filename FILE.GROUP.ACCOUNT could now also be referred to by its POSIX pathname, which would be /ACCOUNT/GROUP/FILE.
- MPE's directory structure could now be more than 2 levels deep.
- In the past, only MPE Accounts could be attached to the top of the directory tree (i.e. the "root"). Now files and directories could also be attached to the root.
- In the past, MPE Accounts could only contain MPE groups. Now files and directories could be attached directly to the accounts.
- In the past, MPE Groups could only contain files. Now Groups could also contain directories and files.
- As on the target systems, directories could contain files as well as other directories, which could in turn contain more directories. This allows for a directory structure that can go far deeper than the 2 levels permitted by the "classic" MPE file system.

These enhancements were added to MPE in a way that maintained backward compatibility. In this way, existing MPE applications that were written around pre-POSIX file naming conventions could continue to run without the need for any modifications. The vast majority of MPE applications were originally designed prior to the POSIX enhancements, and were not modified to take advantage of them. For this reason, when porting these applications, MPE's file naming conventions must be taken into account.

For example, if a program running under MPE creates a file called MYFILE, it can refer to it in a number of different ways, including:

- MYFILE
- "myfile",
- MYFILE.MYGROUP,
- MYFILE. GROUP. ACCOUNT, (assuming that the user running the program is logged onto GROUP and ACCOUNT).

However, if you port this program to UNIX, problems will ensue. The creating program will still be able to refer to the file it created as MYFILE. But an attempt to reference it in any of the other ways shown above will result in an error.

- UNIX file names are case sensitive, so an attempt to reference the file as "myfile" will fail.
- UNIX will assume the MYFILE.MYGROUP and MYFILE.GROUP.ACCOUNT are the names of files in the current working directory. It won't resolve the group and account names, because groups and accounts don't exist on UNIX.

If you need to port an application that uses these older MPE file naming conventions, (as most of them do) you have two choices:

- You can either convert the application to use the HFS-like naming conventions of the target platform, or
- You can use one of the emulator products that allows for the use of the old names.

Most of the emulator packages that are available for the recommended target platforms translate MPE style file names to HFS style names. For example, suppose you run a program on a UNIX system that creates a file called MYFILE.

- The program's current working directory will be /ACCOUNT/GROUP. Therefore the file will be created with the path /ACCOUNT/GROUP/MYFILE
- The emulator will ignore the case of referenced filenames. Therefore, references to MYFILE or "myfile" will both be resolved to the same file.
- Although accounts and groups don't exist on UNIX, the emulator will resolve references to MYFILE.MYGROUP or MYFILE.MYGROUP.ACCOUNT to the correct file.

If your target system is being used for purposes other than emulating an HP e3000, you may find it undesirable to attach the MPE accounting structure to the root. Another option is to place the two level MPE structure somewhere lower down in the directory tree of your target platform. Emulator packages allow you to define the location of the MPE root using an environment variable. For example, you might define MPEROOT="/directory1/directory2/". In this case, the MPE fully qualified file name: FILE.GROUP.ACCT will reference the UNIX path /directory1/directory2/ACCT/GROUP/FILE.

Emulation can greatly reduce the amount of work needed to migrate MPE/iX programs, but at a cost. Emulation imposes a certain amount of overhead upon the system that the migrated program is running upon, which may have an impact on the performance of your migrated application. There is also the monetary cost of the emulation package itself to be considered.

Another aspect to consider with emulation is access to files from non-emulated applications. The emulation routines maintain information about the files for you automatically, such as record size and ASCII/binary. Non-emulated applications are unaware of the information and if not careful can corrupt the files.

If you are lucky enough to be asked to port an application that was written around the HFS rules, you'll find that the HFS file naming convention is very similar to that used on UNIX platform, so the port will be relatively straightforward. Similarly, there are POSIX-compliant environments available for Windows platforms. Either way, MPE's HFS file naming conventions are the same as those used on the target platforms.

mpe file types

mpe file types

Another feature of the MPE/iX file system is its support for specialized file types. Special file types provide features that may not be accessible using standard COBOL statements, and therefore may require the use of MPE intrinsics. When a file is created, the file's type is specified. The following are some of the file types supported on MPE.

standard files

By far the most common type of file on MPE systems is the standard file. When you create a file on an HP e3000, if you do not specify a file type, it will default to STD (standard file). A standard file consists of a group of records beginning with record 0 and ending with record $n-1$ (where n is the maximum specified in the *filesize* option).

Standard files may be read or written using either sequentially or using direct access. An example will suffice to explain how it works.

- Suppose a program opens a standard file for sequential output access. The records that it writes will be stored in this output file in the order in which they were written.
- If a program opens a standard file for sequential input, the records will be read in the order they're stored, (i.e. the order in which they were written).
- Programs may also open standard files using direct access. This permits a program to read or write a record based on its relative record number. So, for example, a program can read the n th record in a file without having to first read the $n-1$ records preceding it.

RIO files (relative I/O)

RIO is an enhanced form of relative access. If a program opens a standard file for direct access, it must know the relative record number of the record it's interested in. If multiple processes are accessing the same file, relative record numbers can change dynamically. For example, if process "A" deletes relative record number n , then all the records after n will have their relative record number adjusted by one. This can make things very difficult for a hypothetical process "B" that is trying to access the same file at the same time. The relative record numbers of the records that "B" is interested in could be changing even as "B" is accessing them.

This problem led to the development of RIO files. RIO is a random access method that permits individual file records to be deactivated, rather than deleted. These inactive records retain their relative position within the file. RIO files are intended for use primarily by COBOL programs because there is a COBOL standard for Relative I/O. RIO is available with Cobol compilers on MPE and on the recommended target platforms. They can be accessed using standard COBOL statements.

standard files

rio files (relative i/o)



circular files



message files



circular files

Circular files are wrap-around structures that behave just like standard files until they are full.

- When a standard file is created on MPE, a filesize is imposed, limiting the number of records that the file can contain. If the filesize is “ n ”, any attempt to write “ $n+1$ ” records to the file will result in an error.
- Similarly, when a **circular** file is created on MPE, there is a limit (“ n ”) on how many records the file can contain. As records are written to the circular file, they are appended to the end of the file. But writing record number $n+1$ to a circular file does not cause an error. Instead, it causes the block at the beginning of the file to be deleted and all other blocks to be logically shifted toward the head of the file.

Circular files have been used on MPE for logging purposes. An application program can write to a circular file forever, and the file will always contain a log of the last n records. However, circular files may not be simultaneously accessed by both readers and writers. When the file has been closed by all writers, it may be opened by a reading process. A reader takes records from the circular file one at a time, starting at the beginning of the file.

By default, circular file functionality is not part of any of the recommended target platforms. If you need circular file functionality you will either have to simulate it in your program, or use an emulator. Some of the MPE emulator provide simulated circular files.

message files

Message files (type MSG) are used for interprocess communication (IPC) under MPE. IPC is a facility of the file system that permits processes to communicate with one another in an easy and efficient manner. Message file functionality requires the use of the FREAD and FWRITE intrinsics. Message files act as first-in-first-out, queues of records. An example will explain how they are used.

Consider two processes, “A” and “B”. Process “A” needs to send information to process “B”, which is running at the same time as “A”.

- Suppose Process “A” writes a record to a message file using the FWRITE intrinsic.
- The record left by process “A” may now be read by other processes. Process “B” can do this using the FREAD intrinsic.
- Message file reads are *destructive*. This means that when process “B” reads the record written by “A”, that record will be deleted from the message file. This prevents records in message files from being read more than once.

Message files functionality can be provided on the recommended target platforms through a variety of methods. Most other operating systems support **local sockets** (also called UNIX sockets). Local sockets are not part of the file system, but they are functionally similar to message files. Other methods include pipes and fifo files. All of these methods provide process to process communication but they don’t provide all the features of Message files. (such as non-destructive reads and writing with no readers). Some of the emulator providers offer replacements for most of the message file functionality.

memory mapped files



memory mapped files

On MPE/iX, it is possible to map entire files into virtual storage. This feature is called *mapped access*, and it allows application programs to access the records in a file by moving a pointer through virtual storage, rather than by calling file system intrinsics. The benefit of mapped access is that it is potentially much more efficient than access through conventional file I/O or intrinsics. This is because mapped access generates no file system overhead. It is especially useful when accessing smaller files randomly rather than sequentially.

HP e3000 applications that use mapped access declare a short (32-bit) or long (64-bit) pointer variable. Pointers are special data items used for holding virtual memory addresses. This variable is passed as a parameter to the HPFOPEN intrinsic, which returns the virtual memory address of the opened file. Records in the file can be accessed by incrementing the address in the pointer variable. A record may be written to the file simply by moving the desired data to this address.

Mapped file access is an advanced programming technique, used when it's necessary to squeeze every available bit of performance out of the server. Mapped access is only available in programming languages that support pointers. On the HP e3000, these languages include Pascal and C. COBOL does not support the use of pointer variables. If you are porting an e3000 program that uses mapped file access to one of the recommended target platforms, you should first investigate whether or not a more standard way of accessing the file will be fast enough for your ported application. If you need to continue to use mapped access, then you'll need to modify the program significantly in order to get the same functionality on the target platform.

Features of the target platforms that are analogous to mapped access include *shared memory*. On UNIX systems, shared memory is a way for different processes to access the same locations in memory simultaneously. A process requests a shared memory segment allocation by means of the `shmget()` system call, specifying the segment size in one of the function parameters. One or more processes can then attach to the allocated segment by using the `shmat()` system call and detach when finished by the `shmdt()` system call. The `shmctl()` system call is used to obtain information about the segment, and to remove the segment when it is no longer needed.

spooled devices



spooled devices

By default, a printer is considered to be a non-sharable device by MPE, which means only one user could use it at a time. To overcome this problem, HP e3000 printer devices are **spooled**. *Spool* is an abbreviation for *simultaneous peripheral operation online*. A spooler allows numerous user processes requiring a printer to run simultaneously. Their printed output is re-routed to disk files called **spoolfiles**. The spooler then prints the contents of these spoolfiles one at a time.

The spooling process for a printer is activated by the *SPOOL* console command. Once the spooler is activated for a printer all output that is sent to that device is automatically spooled. MPE's spooler provides a rich set of functionality for managing spoolfiles:

- Spoolfiles are queued up in the order in which they are received by the spooler. Users can assign output priorities to their spoolfiles, which can affect their position in the queue.
- Users can also assign attributes to their spoolfiles, including the number of copies that will be printed, or whether or not special forms be mounted on the printer when it's their spoolfile's turn to print.
- Users can view the contents of spoolfiles before they print, or alter the spoolfile's attributes.
- The MPE spooler supports "Page Level Recovery". This means that should the spooler be interrupted by an error, (such as a system "crash"), it can pick up at the point at which it left off when the system is restarted.

On HP-UX and Linux the spooler process does not provide such a rich set of functionality. The spooler is not associated with a particular device file (printer) when it is started. The spooler process is invoked directly and it then manages the output. An example will help explain some of the differences between MPE's approach to spooling and that of the recommended target platforms.

Suppose an HP e3000 has a spooled printer configured with the device class name "LP". Assuming that this device has been spooled (i.e., that the *SPOOL* console command was issued), this user could issue the following file equation:
:FILE PRINT;DEV=LP

From this point forward, any program that opens a file named PRINT will be opening a spoolfile. Any records written to the file named PRINT will be sent to the spooler. The user has the options of specifying spoolfile attributes (e.g. number of copies, output priority), by specifying appropriate parameters on the :FILE command.

Let's contrast this MPE spooler example with the HP-UX spooler. On HP-UX if a process opens /dev/lp the process would "own" the printer until it closed it. To get your output spooled you would have to send it to the spooler process, instead of to the printer.

For example you might issue the shell command: **cat file > lp.**

Lp is a UNIX function that sends output to a spooler daemon that then sends the output to a printer.

In addition to `lp` there is the `pdpr` command that uses the HP Distributed Print Services (HPDPS). On HP-UX the System Administration Manager (SAM) can be used to configure the spoolers. It's beyond the scope of this white paper to fully explain the spooler capabilities of the recommended target platforms. Additionally documentation can be found at www.docs.hp.com, and HP-UX fundamentals training (which includes more information about the spooler) is available on the web at www.education.hp.com/curr-mpe-e3000.htm.

Some compilers (for example COBOL) handle the sending of output to spooled printers automatically. Other compilers require some special programming to send output to a printer. With C it is common to send output to `stdout`. When the program is run, `stdout` is redirected or "piped" to a spooler process. For example for a program called `foo`, you'd issue the following shell command:
foo | lp.

Another way to send output to a spooler is to open a pipe using `popen()` in your program to the spooler process. This allows output to be sent directly to the spooler.

The bundled spoolers available on the recommended target machines vary in capabilities and most do not have the robust feature set of the MPE spooler (e.g. page level recovery). Third party spoolers are available that offer functionality beyond the basic capabilities. Additionally some of the emulator solutions include spooling capabilities.

file domains

MPE file can exist in three different domains, New, Temporary and Permanent.

- A **new file** exists only for the duration of the process that created it. It cannot be opened, accessed or seen by another process. When a process terminates on MPE, any files that it created in the new domain will be automatically deleted.
- A **temporary file** is one that exists only for the duration of the Job or Session that created it. It can only be opened, accessed or seen by other processes in the creating Job or Session. When a job or session logs off the system, any temporary files that they created are deleted automatically
- A **permanent file** exists as a file in the system file domain. Its existence is not limited to the duration of its creating Job or Session, and depending on security restrictions; it may be accessed by jobs or sessions other than the one that created it. Permanent files are visible to any and all users on the system⁴. There can only be one permanent file on the system with a given fully qualified file name.

It is common practice on MPE systems to create temporary work files in the temporary domain. The fact that the temporary domain is only visible to the creating job or session means that multiple instances of a job can run concurrently without fear of interfering with one another.

⁴ Note the distinction between a file's *visibility* and its *accessibility*. When you create a file, it's visible to everybody on the system. That is, any user can use an MPE command such as `:LISTFILE` to see that the file exists. However, the file may not be accessible to everybody on the system. Depending on how the system administrator has set up file security, file access may be restricted to the creator alone, or to users in the same group or account.

file domains



For example, suppose a user streams a batch job called J1. A program being run by J1 creates a scratch file called WORK, and places it in the temporary domain. If the user streams the job again, it will create a second job - we'll call it J2. J2 runs the same program as J1, so it too will create a scratch file called WORK. Because each job has its own temporary domain, the two WORK files are completely separate from one another. The files in J1's temporary domain are invisible to J2 and vice-versa.

Temporary files are automatically removed from HP e3000 systems when the creating Job or Session is completed, even if it is an abnormal termination.

On the recommended target platforms, there is only one file domain: permanent. This means that programmers must make certain there are no naming conflicts in the file names that are used. Some programmers achieve this by creating unique file names (using the Process ID or PIN as part of the file name or putting in a time stamp in the name).

There is a convention that is often used on UNIX system of putting temporary files into a directory called temp or /tmp. It is commonly agreed that files in the temp directory don't need to be retained. System administrators periodically purge any files that are left in the temp directory. Some of the MPE emulation environments simulate MPE's temporary domain by generating unique names and placing them in the temp directory, or purging them automatically.

FILE equations

FILE equations are a unique feature of the MPE file system. FILE equations are a way to declare file attributes outside of a program. These attributes are used at the time the file is opened. They may be used to override programmatic or the system's default file specifications. FILE equations are Job or Session specific.

There are three basic uses of FILE equations:

1. **Change the attributes of a file that is opened in a program.** For example suppose a program has been written to create a scratch file that can hold 2000 records. When the program terminates, this file is deleted automatically. As your company grows, the program is called upon to process more data until one day, you find that 2000 records is no longer large enough. The program need a larger scratch file. Even though the limit is hard coded in the program, it can be overridden using a FILE equation. Issue a FILE command associating the name of the scratch file with a limit of 4000 records. When the program creates the file, (by opening it), the attributes specified in the :FILE equation override the attributes specified in the program. For example, some of these attributes include:

- The **capacity** of the file (which determines the maximum amount of data that the file can contain)
- The **structure** of the file (which determines the size of the records that make up the file)
- The file **domain**, (which determines how long the file remains on the system)
- The **file type**



2. **Point a file to a different file type or device.** For example suppose a program normally writes a report called REPORT to the system line printer. One day, you have a need to route the report to your terminal. This can easily be achieved by issuing a FILE command before you run the program that associates REPORT with \$STDLIST (the filename associated with each user's terminal). When the program opens the REPORT, the attributes specified in the FILE equation will override those in the program. Each record that is written to the REPORT file will be directed to \$STDLIST instead, and be displayed on the user's terminal.
3. **Change the name of a file.** For example a program creates an output file called OUTPUT but with a FILE equation you can redirect the records written to this file to a different file: ORD0001D.WORK.ACCT.

There are more things you can do with FILE equations but they are variations of these three basic uses. What is common to all its uses is the ability to change aspects of a file without having to change the compiled code, all FILE equations are issued in JCL, command files, UDCs and the Command Interpreter (CI, the MPE shell).

FILE equations do not exist on any of the recommended target platforms so the functionality has to be implemented in different ways. Let's go through the three basic uses:

1. **Change the attributes of a file that is opened in a program.** File systems on Linux, UNIX and Windows don't offer the rich set of file attributes offered by MPE. Basically, file structures are very simple. There's no record structure, and only one type of file. This means that there are no attributes to change. Therefore, this functionality is not needed on the recommended target platforms.
2. **Point a file to a different file type or device.** On the recommended target platforms, changing device types needs to be anticipated and programmed for. The device being used by the program can be defined in a configuration file or be set up in an environment variable.
3. **Change the name of a file.** Changing the name of a file can also be handled by using a configuration file or an environment variable. Often what a name change is doing is putting a file in a specific group and account or directory. Using a path variable can perform this same functionality. For example, suppose a program creates a work file called ord0001d. For this application all work files are located in /appl/work/. A path environment variable called WPATH can be set to "/appl/work/". The program can then be changed to get the WPATH variable and concatenate it with the file name to establish the full file name (/appl/work/ord0001d). Another way to redirect a file name is with symbolic links with the ln command.

FILE equation functionality is also available in many of the emulation environments.

ksam

KSAM is the Keyed Sequential Access Method database system that comes with the MPE/iX operating system. The KSAM access method makes it possible to access files sequentially, or directly, or using Keyed Access. Sequential access and direct access is similar to the functionality associated with standard files so we won't explore that any further here. However, KSAM files can also be accessed using keys. Keyed access requires that a field in each record be identified in advance as a key field. When data is written to a KSAM file, the KSAM access method not only writes the data, it also writes the key field to an index, which can then be used to retrieve the data record. This allows applications to access records from the middle of a file without having to search the file sequentially, or keep track of relative record numbers.

The TurboIMAGE DBMS also provides keyed access to data. KSAM is not a database management system. It is a simple keyed access method — compatible with the "ACCESS IS INDEXED" features of COBOL. These allow KSAM files to be accessed from COBOL programs without the need for explicit intrinsic calls.

KSAM has evolved over the years so that there 3 different versions of the KSAM access method available on the HP e3000.

- There is CM KSAM, the original compatibility mode KSAM. This dates back to the "classic" HP 3000 architecture. When MPE was re-written for the PA-RISC architecture, CM KSAM was brought over from the earlier 16-bit architecture in compatibility mode. Although it worked, it was subject to many of the limitations of the older 16-bit architecture. It is normally maintained using a utility program called KSAMUTIL, and accessed using the CKxxx intrinsics.
- KSAM XL is a native mode implementation of KSAM that was implemented on the PA-RISC models of the HP 3000 as the limitations of CM began to become problematical for customers who were using CM KSAM. Programmatically speaking, KSAM XL is compatible with CM KSAM, but it is more resilient against system problems and application aborts.
- KSAM 64 a newer native mode implementation of KSAM that supports larger files.

As KSAM evolved, HP maintained a high level of compatibility, so that applications could be migrated to new versions without having to be rewritten. All three kinds of KSAM files can be created using MPE's :BUILD command and accessed using the file system intrinsics (HPFOPEN, FREAD...). KSAM files can be accessed from COBOL using the standard COBOL file I/O verbs (READ, WRITE, DELETE...) if the file is declared to be "Indexed" with the SELECT clause in the INPUT-OUTPUT SECTION.



MPE applications that make use of one of the KSAM access methods can be ported to any of the recommended target platforms. However, no KSAM access method is bundled with HP-UX, Linux or Windows. There are three replacements for KSAM,

1. **Use another Sequential Access Method database.** C-ISAM from Informix is the most popular of these. The user interfaces to the other Sequential Access Method databases are different than KSAM so there would require some programming changes.
2. **Use a relational database as the data store.** You can set this up by using a Table for the file and having a unique index as the key. You can access the database directly through SQL or there are emulator wrappers that will translate the KSAM access to the relational database.
3. **Use the Indexed file access available with the COBOL compilers.** Because indexed access is part of standard COBOL, the COBOL compiler for your selected target platform almost surely supports an Indexed file access and some support. This is often achieved using C-ISAM as the indexed file system.

table 1. shows sources of KSAM replacement technologies

product	comments/contacts	operating system
AMXW	Provides KSAM wrappers www.neartek.com	HP-UX, Windows
Transport	Provides KSAM wrappers www.transport.bi-tech.com	HP-UX, Windows
Omni Access	Provides KSAM wrappers to relational database www.disc.com	HP-UX, Windows
AcuCOBOL	Provides indexed files with compiler www.acucorp.com	HP-UX, Windows, Linux
MF COBOL	Provides indexed files with compiler also can use C-ISAM www.microfocus.com	HP-UX, Windows
C-ISAM	A popular Sequential Access Method database www.informix.com	HP-UX, Windows, Linux

migrating programs that use intrinsics



Most MPE applications make use of intrinsics — either to access features of the file system, or to access some other part of MPE. To migrate these programs, the code referencing the intrinsics either needs to be changed, or you need to emulate the intrinsic.

Remember that just because the programmer who wrote the application being ported used an intrinsic does not mean that the functionality represented by the intrinsic is not available on your target platform. For example, in the early days of MPE, the only way to access KSAM files from a COBOL program was using the CKxxxx intrinsics. However, today's COBOL compilers can provide the same indexed functionality (on MPE as well as on the recommended target platforms) without making specialized system calls.

When you encounter calls to MPE intrinsics, be sure to analyze the functionality that the intrinsics are providing to the program. Then investigate the best way to provide that functionality on your selected target platform.

There are two aspects to be considered when migrating applications that use MPE intrinsics:

1. Providing the function being called and
2. Addressing the special attributes of the intrinsic call (e.g. optional parameters and condition code).

There are four different ways to address intrinsics migrations:

1. **Manually convert them to target native interfaces.** Find all intrinsics calls in your application's source code and then identify the corresponding system functions on your selected target platform. A cross-reference tool that can help with identifying the appropriate Unix system calls is available off of the www.education.hp.com/curr-mpe-e3000.htm web site. Because you will be using totally different functions calls addressing the special intrinsics attributes will be handled automatically. Some intrinsics functions do not have a corresponding Unix system function so if your application uses these some additional engineering will be necessary
2. **Use a commercially available intrinsics library.** There are MPE/iX intrinsics libraries available from third parties. These libraries provide substitutes for many of the MPE-iX intrinsics. In some cases, not all intrinsic functionality is provided but they can provide a quick way to migrate your code if it works for you.
3. **Use a full feature intrinsics library that is part of a whole emulation environment.** The emulation solution providers attempt to support the majority of the MPE/iX intrinsics. They also address the special attributes of the intrinsic calls. These intrinsics depend on the other components of emulation, including pre-compilers, accounting structure, and daemons.
4. **Automatically convert them to native interfaces.** Between system functions, language extensions and some generated code, the program code can be converted using an automatic conversion tool. These tools are utilized in an overall migration engagement.

table 2. identifies some of the tools that may be helpful in porting applications that use MPE/iX intrinsics and file system to one of the recommended target platforms

	product	comments/contacts	operating system
basic file system intrinsics	Intrins/iX	A Subset of the MPE/iX intrinsics www.allegro.com	HP-UX, Windows, Linux
emulated file system	AMXW	Part of there full emulation environment www.neartek.com	HP-UX, Windows
	Transport	Part of their full emulation environment www.transport.bi-tech.com	HP-UX, Windows
automatic file system conversion	Intelligent Adapters	Part of their migration services www.transoft.com	HP-UX, Windows, Linux
	ViaNova 3000	Part of their ViaNova migration package www.ordina-denkart.com	HP-UX, Windows

appendix a: more information about intrinsic



When a user-written program calls an intrinsic, it also passes information to it via **parameters**. Parameters are data items which are acted upon by the intrinsic. When the intrinsic returns control to the calling program, a value or values may be returned to the calling program. Each intrinsic has a unique parameter list that it expects to have passed to it. The MPE intrinsics and parameter lists are all documented in MPE manuals that are listed in the tables in this appendix.

Some of the attributes of intrinsics that differentiate them from ordinary procedures include the following:

- Since intrinsics are part of the MPE/iX operating system, they execute in **privileged mode**. Privileged mode (often abbreviated “PM”) is necessary to perform operating system tasks.
 - HP **does not support** the use of privileged mode application programs on the e3000. This is because an application program that is running in privileged mode can (through an error) cause a system abort.
 - HP **fully supports** the use of intrinsics. The intrinsics are HP’s supported way of providing privileged mode operating system functionality to application programs. This only exception to this rule are the privileged mode intrinsics (see table A13)
- The parameters passed to an intrinsic determine to some degree what that intrinsic will do. Therefore, each MPE/iX intrinsic performs extensive type and bounds checks on the parameter values passed to it before it uses them. This prevents intrinsics from engaging in destructive behavior because of invalid parameters.
- Each intrinsic has a documented list of permitted parameters and parameter values. These are documented in the MPE/iX Intrinsics Reference Manual, which is available on the web, or in the manuals described in the following tables. (See <http://docs.hp.com/mpeix/all/index.html>).
- Some intrinsic parameters are optional. (This differentiates intrinsics from the system calls used on most other operating systems, where optional parameters are not allowed.)
- Some intrinsics support the **condition code status** mechanism. The Condition code is a means of returning an indicator of the success or failure of the called intrinsic. The condition code is unique to MPE and is not supported on any other operating system.

The remainder of this appendix is made up of a series of tables, which list the MPE/iX intrinsics that are used for the following purposes.

1. Table A1 contains a list of the intrinsics that are used for Accessing Files
2. Table A2 shows a list of the intrinsics that are used for accessing features of the MPE/iX Command Interpreter
3. Table A3 contains a list of the intrinsics that are used for Managing Processes
4. Table A4 contains a list of the intrinsics that allow user programs to manage certain system Resources
5. Table A5 shows a list of the intrinsics that are used for Programming for Localization
6. Table A6 contains a list of the intrinsics that are used for Managing Message Catalogs
7. Table A7 contains a list of the intrinsics that may be used for Converting Data from one data Type to another
8. Table A8 contains a list of the intrinsics that can be used to invoke the MPE/iX sort/merge utility for the Sorting and Merging of Data
9. Table A9 contains a list of the intrinsics that are used for Handling Traps.
10. Table A10 contains a list of the intrinsics that are used to manage Logging Facilities
11. Table A11 contains a list of intrinsics that can be used to aid in the Debugging of Applications
12. Table A12 contains a list of the intrinsics that can be used to obtain Information about devices which are attached to the system
13. Table A13 contains a list of intrinsics that can be used for Programming in Privileged mode.
14. Table A14 contains a list of intrinsics that may be used for Managing USL Files.
15. Table A15 contains a list of intrinsics used for Managing Data Segments.
16. Table A16 contains a list of intrinsics which can be used for Changing the Stack Size.
17. Table A17 contains a list of intrinsics used to allow programs to switch back and forth between native mode and compatibility mode.
18. Table A18 contains a list of intrinsics used for Controlling Asynchronous Devices (typically terminals)
 - The intrinsics used for accessing TurboIMAGE databases are not listed here. For more information on TurboIMAGE, see the fourth white paper in this series: ***“HP e3000 Solution Transition Advisor – TurboIMAGE and Databases”***.
 - The intrinsics used for using VPLUS are not listed here. For more information on VPLUS, see the third white paper in this series: ***“HP e3000 Solution Transition Advisor – Compilers and User Interfaces.”***
 - The manuals listed in the following tables are all available on the web at **www.docs.hp.com**

table A1. contains a list of the intrinsics that are used for Accessing Files. The file handling intrinsics are perhaps the most widely used intrinsics

function	intrinsic	manual
Opening a file	FOPEN FPARSE HPFOPEN	Accessing Files Programmer's Guide*
Closing a File	FCLOSE	Accessing Files Programmer's Guide*
Writing data to a file	FDEVICECONTROL FSETMODE FUPDATE FWRITE FWRITEDIR FWRITELABEL PRINT PRINTOP PRINTOPREPLY	Accessing Files Programmer's Guide*
Reading data from a file	FREAD FREADBACKWARD FREADDIR FREADSEEK FSETMODE READ READX	Accessing Files Programmer's Guide*
Controlling record pointer movement	FCONTROL FPOINT FSPACE	Accessing Files Programmer's Guide*
Accessing a mapped file	HPFOPEN HPFMovedATA HPFADDDTOPOINTER HPFMovedATALTOR HPFMovedATARTOL HPFFILLDATA	Accessing Files Programmer's Guide*
Sharing a file	FOPEN FLOCK FUNLOCK HPFOPEN	Accessing Files Programmer's Guide*
Maintaining file security	FOPEN HPFOPEN HPACDINFO HPACDPUT	Accessing Files Programmer's Guide* No manual No manual
Getting file information	FCHECK FERRMSG FFILEINFO FGETINFO FLABELINFO FRELATE FRENAME	Accessing Files Programmer's Guide*
Error Checking	HPERRDEPTH HPERRREAD HPERRMSG PRINTFILEINFO	Accessing Files Programmer's Guide*
Accessing an RIO file	FOPEN FDELETE FREAD FWRITE HPFOPEN	Accessing Files Programmer's Guide*

* www.docs.hp.com/mpeix/onlinedocs/32650-90885/32650-90885.html

table A2. shows a list of the intrinsics that are used for accessing features of the MPE/iX Command Interpreter. The MPE/iX command interpreter is discussed in more detail in the sixth paper in this series, "HP e3000 Solution Transition Advisor – MPE Commands and Networks."

function	intrinsic	manual
Using commands programmatically	COMMAND HPCICOMMAND	Command Interpreter Access and Variables Programmer's Guide*
Controlling variables	HPCIDELETEVAR HPCIGETVAR HPCIPUTVAR	Command Interpreter Access and Variables Programmer's Guide*
Controlling job control words (JCWs)	FINDJCW GETJCW PUTJCW SETJCW	Command Interpreter Access and Variables Programmer's Guide*
Identifying Parameter Input	MYCOMMAND SEARCH	Command Interpreter Access and Variables Programmer's Guide*

* www.docs.hp.com/mpeix/onlinedocs/32650-90493/32650-90493.html

table A3. contains a list of the intrinsics that are used for Managing Processes

function	intrinsic	manual
Activating a Process	ACTIVATE	Interprocess Communication Programmer's Guide*
Deactivating/Suspending a Process	ABORTSESS CAUSEBREAK IODONTWAIT IOWAIT KILL PAUSE PROCINFO PROCTIME QUIT QUITPROG STARTSESS SUSPEND TERMINATE	Interprocess Communication Programmer's Guide*
Creating a Process	CREATE CREATEPROCESS	Interprocess Communication Programmer's Guide*
Obtaining Process Information	FATHER GETINFO GETORIGIN GETPRIORITY GETPROCID GETPROCINFO JOBINFO	Interprocess Communication Programmer's Guide*
Obtaining Mail Information	MAIL RECEIVEMAIL SENDMAIL	Interprocess Communication Programmer's Guide*

* www.docs.hp.com/mpeix/onlinedocs/32650-90019/32650-90019.html

table A4. contains a list of the intrinsics that allow user programs to manage certain System Resources

function	intrinsic	manual
Managing Global RINs	LOCKGLORIN UNLOCKGLORIN	Resource Management Programmer's Guide*
Managing Local RINs	FREELOCRIN GETLOCRIN LOCKLOCRIN LOCRINOWNER UNLOCKLOCRIN	Resource Management Programmer's Guide*
	HPFIRSTLIBRARY HPGETPROCLABEL HPMYFILE HPMYPROGRAM	Resource Management Programmer's Guide*

* www.docs.hp.com/mpeix/onlinedocs/32650-90024/32650-90024.html

table A5. shows a list of the intrinsics that are used for Programming for Localization

function	intrinsic	manual
Retrieving information	ALMANAC NLGETLANG NLINFO	Native Language Programmer's Guide*
Handling characters	NLCOLLATE NLFINDSTR NLJUDGE NLKEYCOMPARE NLMATCH NLMATCHINIT NLREPCAR NLSCANMOVE NLSUBSTR NLSWITCHBUF NLTRANSLATE	Native Language Programmer's Guide*
Formatting time and date	NLCONVCLOCK NLCONVCUSTDATE NLFMTCALENDAR NLFMTCLOCK NLFMTCUSTDATE NLFMTDATE NLFMTLONGCAL	Native Language Programmer's Guide*
Formatting numbers	NLCONVNUM NLFMTNUM NLNUMSPEC	Native Language Programmer's Guide*
Using application message catalogs	CATCLOSE CATOPEN CATREAD NLAPPEND	Native Language Programmer's Guide*

* www.docs.hp.com/mpeix/onlinedocs/32650-90207/32650-90207.html

table A6. contains a list of the intrinsics that are used for Managing Message Catalogs

function	intrinsic	manual
	CATCLOSE CATOPEN CATREAD GENMESSAGE	Message Catalogs Programmer's Guide*
Converting binary numbers	ASCII DASCII	Data Types Conversion Programmer's Guide**
Converting numeric ASCII strings	BINARY DBINARY	Data Types Conversion Programmer's Guide**
Translating ASCII/EBCDIC or JISCII/EBCDIK	CTRANSLATE	Data Types Conversion Programmer's Guide**
Converting floating-point formats	HPFP_CONVERT	Data Types Conversion Programmer's Guide**

* www.docs.hp.com/mpeix/onlinedocs/32650-90021/32650-90021.html

** www.docs.hp.com/mpeix/onlinedocs/32650-90015/32650-90015.html

table A7. contains a list of the intrinsics that may be used for Converting Data from one Data Type to another

function	intrinsic	manual
Converting binary numbers	ASCII DASCII	Data Types Conversion Programmer's Guide*
Converting numeric ASCII strings	BINARY DBINARY	Data Types Conversion Programmer's Guide*
Translating ASCII/EBCDIC or JISCII/EBCDIK	CTRANSLATE	Data Types Conversion Programmer's Guide*
Converting floating-point formats	HPFP_CONVERT	Data Types Conversion Programmer's Guide*

* www.docs.hp.com/mpeix/onlinedocs/32650-90015/32650-90015.html

table A8. contains a list of the intrinsics that can be used to invoke the MPE/iX sort/merge utility for the Sorting and Merging of Data

function	intrinsic	manual
Creating core merge routines (NM)	HPMERGEEND HPMERGEERRORMESS HPMERGEINIT HPMERGEOUTPUT	SORT-MERGE/XL Programmer's Guide*
Creating core merge routines (CM)	MERGEEND MERGEERRORMESS MERGEINIT MERGEOUTPUT	SORT-MERGE/XL Programmer's Guide*
Getting merge information (NM)	HPMERGESTAT HPMERGETITLE	SORT-MERGE/XL Programmer's Guide*
Getting merge information (CM)	MERGESTAT MERGETITLE	SORT-MERGE/XL Programmer's Guide*
Creating core sort routines (NM)	HPSORTEND HPSORTERRORMESS HPSORTINIT HPSORTINPUT HPSORTOUTPUT	SORT-MERGE/XL Programmer's Guide*
Creating core sort routines (CM)	SORTEND SORTERRORMESS SORTINIT SORTINPUT SORTOUTPUT	SORT-MERGE/XL Programmer's Guide*
Getting sort information (NM)	HPSORTSTAT HPSORTTITLE	SORT-MERGE/XL Programmer's Guide*
Getting sort information (CM)	SORTSTAT SORTTITLE	SORT-MERGE/XL Programmer's Guide*

* www.docs.hp.com/mpeix/onlinedocs/32650-90080/32650-90080.html

table A9. contains a list of the intrinsics that are used for Handling Traps. On MPE/iX, traps are a technique for trapping errors that might otherwise cause application programs to abort.

function	intrinsic	manual
	ARITRAP FINTEXT FINSTATE HPENABLTRAP RESETCONTROL XARITRAP XCONTRAP XLIBTRAP XSYSTRAP	Trap Handling Programmer's Guide*

* www.docs.hp.com/cgi-bin/doc3k/B3265090026.10457/1

table A10. contains a list of the intrinsics that are used to manage Logging Facilities

function	intrinsic	manual
Marking a logical transaction	BEGINLOG	User Logging Programmer's Guide*
Closing a log file	ENDLOG	User Logging Programmer's Guide*
Flushing the logging buffer	CLOSELOG	User Logging Programmer's Guide*
Getting information from the log file	FLUSHLOG	User Logging Programmer's Guide*
Opening a log file	LOGINFO	User Logging Programmer's Guide*
Writing to a log file	LOGSTATUS	User Logging Programmer's Guide*
	OPENLOG	User Logging Programmer's Guide*
	WRITELOG	User Logging Programmer's Guide*

* www.docs.hp.com/mpeix/onlinedocs/32650-90027/32650-90027.html

table A11. contains a list of intrinsics that can be used to aid in the Debugging of Applications

function	intrinsic	manual
Entering the debug facility	DEBUG	MPE/iX System Debug Reference Manual*
Disarming a debug call	HPDEBUG	MPE/iX System Debug Reference Manual*
Arming a debug call	HPRESETDUMP	MPE/iX System Debug Reference Manual*
Producing a full stack trace	RESETDUMP	MPE/iX System Debug Reference Manual*
	HPSETDUMP	MPE/iX System Debug Reference Manual*
	SETDUMP	MPE/iX System Debug Reference Manual*
	STACKDUMP	MPE/iX System Debug Reference Manual*

* www.docs.hp.com/mpeix/onlinedocs/32650-90824/32650-90824.html

table A12. contains a list of the intrinsics that can be used to obtain Information about devices which are attached to the system

function	intrinsic	manual
Obtaining volume information	HPVOLINFO	
Accessing peripheral functionality	HPDEVCONTROL	

table A13. contains a list of intrinsics that can be used for Programming in Privileged mode. Note that HP will not support applications that use these intrinsics.

function	intrinsic	manual
Starting privileged mode	GETPRIVMODE	Introduction to MPE XL for MPE V Programmers*
Ending privileged mode	GETUSERMODE	Introduction to MPE XL for MPE V Programmers*

* www.docs.hp.com/cgi-bin/doc3k/B3036790005.10124/1

table A14. contains a list of intrinsics that may be used for Managing USL Files. On older "classic" models of the HP 3000, a USL file is a library of compiled code. USL files were used as input by a utility called the "segmenter", which created executable program files. On the newer PA-RISC models of the HP e3000, USL files and the segmenter are only used in compatibility mode. MPE/iX the linkage editor to create native mode application programs.

function	intrinsic	manual
Changing USL files	ADJUSTUSLF EXPANDUSLF	MPE Segmenter Reference Manual*
Creating USL files	CLEANUSL	MPE Segmenter Reference Manual*
Initializing USL files	INITUSLF	MPE Segmenter Reference Manual*

* www.docs.hp.com/mpeix/pdf/30000-90011.pdf

table A15. contains a list of intrinsics used for Managing Data Segments. By default, each process on the older "classic" models of the HP e3000 is allocated a certain amount of memory. Additional memory may be allocated in the form of data segments.

function	intrinsic	manual
	ALTDSEG DMOVIN DMOVOUT FREEDSEG GETDSEG SWITCHDB	Introduction to MPE XL for MPE V Programmers*

* www.docs.hp.com/cgi-bin/doc3k/B3036790005.10124/1

table A16. on older "classic" models of the HP e3000, each process is allocated a memory structure called "the stack". The following intrinsics can be used for Changing the Stack Size.

function	intrinsic	manual
Changing the stack size	DLSIZE ZSIZE	Introduction to MPE XL for MPE V Programmers*

* www.docs.hp.com/cgi-bin/doc3k/B3036790005.10124/1

table A17. When the PA-RISC models of the HP e3000 were introduced, they were able to execute binaries from the earlier "classic" models using a feature called "compatibility mode" (CM). Programs that were written specifically for the PA-RISC models are called "native mode" (NM) programs. The following intrinsics are used to allow a program to switch back and forth between CM and NM, thus allowing complex programs to be ported from CM to NM in stages.

function	intrinsic	manual
Loading a CM procedure	HPLOADCMPROCEDURE LOADPROC	Switch Programming Guide*
Unloading a CM procedure	HPUNLOADCMPROCEDURE UNLOADPROC	Switch Programming Guide*
Loading a NM procedure	HPLOADNMPLABEL	Switch Programming Guide*
Switching from CM to NM	HPSWTONMNAME HPSWTONMPLABEL	Switch Programming Guide*
Switching from NM to CM	HPSWITCHTOCM HPSETCCODE	Switch Programming Guide*

* www.docs.hp.com/cgi-bin/doc3k/B3265090014.70/1

table A18. the following intrinsics are used for Controlling Asynchronous Devices (typically terminals)

function	intrinsic	manual
Controlling system breaks	FCONTROL CAUSEBREAK	Asynchronous Serial Communications Programmer's Reference Manual*
Controlling subsystem breaks	FCONTROL XCONTRAP RESETCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*
Specifying carriage control directives	FCONTROL FWRITE	Asynchronous Serial Communications Programmer's Reference Manual*
Specifying an EOR character	FCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*
Enabling/Disabling echo	FCONTROL FSETMODE	Asynchronous Serial Communications Programmer's Reference Manual*
Specifying line deletion echo response	FCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*
Setting editing mode	FCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*
Setting transmission mode	FCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*
Specifying and enabling parity	FCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*
Specifying terminal type	FCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*
Setting a read timeout	FCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*
Timing a read	FCONTROL	Asynchronous Serial Communications Programmer's Reference Manual*

* www.docs.hp.com/mpeix/onlinedocs/32022-90052/32022-90052.html

appendix b: device independence



Since all input/output operations on the HP e3000 are done through the mechanism of files, application programs can access very different devices in a standard, consistent way. It does not make much difference to the application program (or the programmer) whether a file that is being accessed is stored on a disk device, from a magnetic tape, or on a device that's connected to another HP e3000 across a network. The MPE file system treats all files in the same way.

Because of this characteristic of the MPE file system, MPE application programs are said to be *device independent*. This means that when a program creates a file on the HP e3000, certain characteristics are assigned to the file, such as the type of device upon which the file resides. But these characteristics do not need to be hard-coded in the creating program itself. In fact, each time the program is executed, the file it creates can reside on a different device (or even on a different *kind of device*) without any need to modify the program. The same is true of programs that read files. The same application program can read data from a disk device, a magnetic tape device, or from a device that's connected to another HP e3000 across a network. No modifications need to be made to the program to enable it to access files on a new or different device, or a new kind of device.

If a user wishes to change the behavior of an application program, so that instead of creating a file on a disk, it creates a file on another device (or kind of device), this is achieved using an MPE command called the `:FILE` command. MPE commands will be discussed in more detail in the sixth paper in this series, "*HP e3000 Solution Transition Advisor — MPE Commands and Networks*". For the moment, suffice it to say that the `:FILE` command is available to authorized end users, and may be used in conjunction with an application program to create and manipulate files.

Essentially, the `:FILE` command is used to specify file characteristics, overriding those that may (or may not) be specified in an application program. For example, an application program may contain a call to the `FOPEN` intrinsic that creates a new file called `MYFILE`. If the application program does not specify what kind of device this file is to reside upon, MPE will assume certain defaults. By default, new files are created on the first available disk device.

However, if the user issues a `:FILE` command before executing the application program, this default will be overridden. For example, the command: `:FILE MYFILE; DEV=12` specifies that the file `MYFILE` is to reside, not on any available device, but specifically on the device identified as logical device number 12. (On MPE, every peripheral device is identified by a number, called the "logical device number" or "LDEV" for short). When a `:FILE` command is executed by a session, the characteristics specified in it apply to that file name for the remainder of the session. So, if at any time during the remainder of that session⁵, an application program calls `FOPEN` or `HPFOPEN` in order to create a file called `MYFILE`, MPE will create that file on logical device 12.

⁵The file equation is in effect for the remainder of the session by default. It can be taken out of effect using another MPE command: `RESET`.

device files

device files

The MPE file system also permits you to access nonshareable devices. Any peripheral device except a disk is considered nonshareable.

Because all file open operations are accomplished through the file system, you can open files on very different kinds of devices in a standard, consistent way, using the HPFOPEN or FOPEN intrinsics. Furthermore, the name and characteristics 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. In these cases, the file system temporarily overrides the programmatic characteristics with those characteristics required by the device.

When the HP e3000 is configured the different peripherals are assigned logical device numbers (LDev) and optionally device class names. These peripherals can now be accessed through the file system by their LDev or device class. In a FILE equation this is the DEV= component.

For example, consider this file equation: **:FILE T;DEV=TAPE.**

This associates the filename "T" with devices that have been assigned to the device class "TAPE", (presumably magnetic tape drives). A program that opens a file called "T" while this file equation is in effect will automatically call for a tape to be mounted on the first available TAPE device.

HP-UX and Linux also provide support for device files that look like regular files that point to the non-shareable devices. These files normally are located in the /dev directory. The device files are created using *mksf()* or *mknod()*. These device files are created once when the device is installed.



All brand and product names are trademarks or registered trademarks of their respective companies.

notice

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

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

Restricted Rights Legend

Use, duplication, or disclosure is subject to restrictions as set forth in contract subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause 52.227-FAR14.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304, USA

© Copyright Hewlett-Packard Company 2003.
All rights reserved. Reproduction, adaptation or translation of this document is prohibited without prior written permission of Hewlett-Packard Company, except as allowed under the copyright laws.

Printed in USA 7/03
5981-3063EN rev. 1