Introduction to MPE/XL for MPE V Programmers

Series 900 HP 3000 Computer Systems

Manufacturing Part Number: 30367-90005
E1089

U.S.A. October 1989

Notice

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

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

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

Restricted Rights Legend

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

Acknowledgments

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

1

Introduction

The chapters of this manual present the differences between the operating systems organized by common tasks. The purpose of the manual is to help you predict where changes are likely, and how to best adjust your MPE V/E programming techniques to adapt to the MPE XL environments.

This chapter presents the differences between the MPE V/E and the MPE XL operating systems as general concepts.

The following chapters are organized on a task-by-task basis. Each chapter discusses one task. In the beginning, a list is presented of the features, intrinsics, and commands that are new, changed, unsupported, and unchanged in MPE XL from MPE V/E. The new, changed and unsupported items are then discussed individually.

For help in locating a specific task, intrinsic, command, or topic, see the Preface, the Table of Contents, and the Index in this book.

Overview of Differences: MPE V/E and MPE XL

All HP 3000 models run under the Multiprogramming Executive (MPE) operating system, which manages all system resources and coordinates the execution of all programs running on the system. The MPE V/E operating system is designed to run on all HP 3000s except the 900 series. The MPE XL operating system is designed to run on the 900 series.

The architecture of the 900 series is based on RISC (Reduced Instruction Set Computer) concepts. Research shows that conventional microcode machines spend about 80% of the time executing about 20% of the instructions. It makes sense, then, to design the computer to directly and quickly implement the most common instructions directly in hardware, instead of supporting bulky microcode.

MPE XL is a register-based machine. CPU (central processing unit) registers hold the most frequently used data and instructions. This method requires less chip space and system overhead than microcode.

HP-PA accesses files with demand-paged virtual memory. This eliminates the need for code and data segmentation. Disk files are mapped into virtual address space, and file system buffering is no longer needed.

MPE XL has two modes: Compatibility Mode (CM) and Native Mode (NM). CM provides backward compatibility with MPE V/E in CM. NM takes full advantage of the 900 Series Hewlett-Packard Precision Architecture (HP-PA). Both are transparent to the user.

HP-PA instructions are pipelined. The overlapping execution of multiple instructions for efficiency is made possible by uniform instructions and cycle-time regularity. Optimizing compilers generate efficient object code, allocate registers, and schedule instruction sequences to maintain efficient pipeline operation.

New

MPE XL features that are not available on MPE V/E include:

- Demand-paged virtual memory
- Dual programming modes

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- Word size
- Priv Mode
- Segmenter

Demand-Paged Virtual Memory (new)

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

File Mapping

File mapping uses MPE XL demand-paged virtual memory to make a large amount of virtual memory directly available to you. When a file is opened, it is logically mapped into virtual memory. Each byte of each opened file has a unique virtual address. The system actually stores files page by page at physical addresses.

The operating system keeps a map, a table of correspondences between virtual addresses and physical addresses, and translates between them as needed. System hardware resolves addresses through the Translation Lookaside Buffer (TLB). This translation process is transparent to the user. You reference an open file and its contents by virtual address.

The MPE XL memory manager fetches several adjacent pages directly from disk as required. It places them in the user's area in memory.

If you program in a language with pointers, you can access mapped files directly. You can write programs that address files through virtual memory, instead of calling file system intrinsics for disk reading and writing. Using a pointer data type, you can open and close user-mapped files with with faster `LOAD` and `STORE` on file references. You can get the advantage of file system naming and data protection for accessing array-type structures and developing specialized access methods.

File mapping improves I/O performance without imposing additional CPU overhead or sacrificing data integrity and protection. HP-PA uses the Translation Lookaside Buffer to resolve addresses in the hardware. Traditional disk caching schemes for increasing I/O performance impose a CPU overhead

penalty. The 900 Series hardware and system architecture allow MPE XL to perform file mapping without this penalty.

Dual Programming Modes (new)

For the programmer, the MPE XL seems to have two different operating systems: Compatibility Mode (CM) and Native Mode (NM). CM is designed to look familiar to MPE V/E programmers. NM is designed to take full advantage of the HP-PA architecture. The two can communicate.

You can develop new programs on CM or on NM. If you are familiar with programming in MPE V/E, you may find it easier, especially at first, to program in CM. Programmers who try NM, however, find that, in many cases, it takes less programming effort to accomplish the same task.

In CM, the program development cycle appears to be like that of MPE V/E. The process is different in MPE XL NM. The largest impact is virtual memory, which eliminates the need for segmentation. The following table shows a comparison.

Table 1-1. MPE XL and MPE V/E Program Development

Development Step	MPE V/E Tool	MPE XL CM Tool	MPE XL Tool
Edit Source Code	Edit/3000, TDP/3000, Toolset	Editor, TDP	Editor, TDP, Toolset/XL
Compile	MPE V/E Compiler Commands	MPE V/E Compiler Commands (plus Object Code Translator)	MPE XL NM Compiler Commands (includes Optimizer)
Linking	Segmenter	Segmenter	HP Link Editor/XL
Link Edit a Program	`PREP`	`PREP`	`LINK`
Create a Relocatable Library	`SEGMENTER`	`SEGMENTER`	`LINKEDIT`
Create a Segmented Library or Executable Library	`SEGMENTER` (SL)	`SEGMENTER` (SL)	`LINKEDIT` (XL)

CM and NM Compilers

Separate high-level language compilers are provided for CM and NM. You can usually change between modes simply by recompiling. You may experience some difficulty between modes with data alignment, especially floating-point numbers. Also, some intrinsics are different.

The following table shows which language compilers are available.

Table 1-2. MPE Programming Languages

MPE V/E	MPE XL CM	MPE XL NM
N/A	N/A	HP C/XL
COBOL II/V	COBOL II/V	COBOL II/XL
HP FORTRAN 77/V	HP FORTRAN 77/V	HP FORTRAN 77/XL
FORTRAN 66/V	FORTRAN 66/V	N/A
Pascal/V	Pascal/V	HP Pascal/XL
RPG/V	RPG/V	RPG/XL
HP Business BASIC/V	HP Business BASIC/V	HP Business BASIC/XL
SPL/V	SPL/V	N/A

Note Although a limited number of compilers are available in Compatibility Mode (CM), MPE XL provides run-time support, via CM libraries, for all HP-supported MPE V/E-based languages. Thus, any program that was compiled on an MPE V/E-based system using an HP supported compiler, can be executed in CM on an MPE XL-based system.

Switching between CM and NM

The Switch subsystem helps you create a program that alternates operation between CM and NM. You can make calls from one mode to a routine in the other mode. This way, your NM program can call a CM procedure, or vice versa.

A switch stub is a routine that acts as an intermediate step between a calling procedure in one operating mode and a called procedure in another. A call to a switch stub looks exactly like a call to the actual procedure. Thus, making cross-mode procedure calls is transparent to the calling application and its users.

The Switch Assist Tool (SWAT) helps you create switch stubs from NM to CM. For more information on creating switch stubs and SWAT, refer to *Switch Programming Guide* (32650-90014).

Emulating or Translating Migrated MPE V Code

Programs originally written on an MPE V/E system can be migrated to MPE XL CM. You decide whether to emulate or translate MPE V/E compiled source code in order to use it on MPE XL.

Emulating is done by the emulator. It uses a branch table to execute HP-PA instructions that are equivalent to the MPE V/E code. MPE V/E files migrated to MPE XL will be emulated each time they are run, unless you translate them.

Translating is done by the Object Code Translator (OCT). It translates the code and appends the translation onto a copy of the original file. Translated code usually runs faster than emulated code. Because both the MPE V/E version and the MPE XL version is included, it is flexible. However, the appended file is larger, and usually requires more than twice as much storage space as the original.

Word Size (changed)

MPE V/E and MPE XL have different native word sizes. An MPE V/E standard word is 16 bits long, and an MPE XL standard word is 32 bits long.

Applications may have differences in data alignment. Data compatibility is especially troublesome with floating-point representation of real numbers because CM and NM use different formats.

Priv Mode (changed)

MPE V/E file security recognized two modes: user and priv (privileged). MPE XL recognizes user mode (level 3) and three levels of privileged mode (levels 2, 1, and 0). Only one of these, level 2, is available to programmers. Levels 0 and 1 are reserved for the operating system, and are not accessible.

Segmenter (changed)

MPE XL retains the MPE V/E segmenter program in CM for backward compatibility. The commands and intrinsics will be familiar to the MPE V/E programmer.

Because of virtual memory, segmenting is no longer necessary. MPE XL emulates segmentation in CM. MPE XL does not support segmentation at all in NM.

Additional Information

For further information about the MPE XL system, refer to *Getting Started as an MPE XL Programmer* (32650-90008), and *MPE XL Intrinsics Reference Manual* (32650-90028).

2

Preparing a Program for Execution

This chapter discusses program development in NM (Native Mode) including writing, compiling, linking, running, and handling errors. Developing programs across modes is discussed, including emulating or translating MPE V/E code, and switching between CM (Compatibility Mode) and NM (Native Mode) code.

Overview of Differences: MPE V/E and MPE XL

There are major differences in developing programs in MPE V/E and in MPE XL. These differences may not be apparent at first glance; the model is the same, but components are different.

One difference is that you have two programming modes, NM and CM. During operation, a program can switch between the modes. MPE XL CM emulates MPE V/E for backward compatibility. It maintains the appearance of segmenting, but the virtual memory of MPE XL has made segmenting unnecessary.

New

MPE XL features that are not available on MPE V/E include:

- Dual programming modes
- Native Mode program development

- Mixed mode programs

Dual Programming Modes (new)

One major change from MPE V/E is that there are two programming evironments in MPE XL: CM (Compatibility Mode) and NM (Native Mode). CM is provided for backward compatibility with MPE V/E and cross-mode development. That is, a program developed in MPE V/E can be run on MPE XL, and a program can be developed in MPE XL CM to run on MPE V/E.

CM provides an environment familiar to the MPE V/E programmer.

NM is designed to take full advantage of HP-PA, the architecture of the 900 Series HP 3000.

The two programming environments can communicate and cooperate with each other. Their operations are transparent to the programmer and to the end user. You have several options in developing programs:

- You can develop and compile programs in MPE V/E and then migrate them to run on MPE XL CM.
- You can develop and run programs entirely in MPE XL CM. CM will probably be more familiar to you than NM.
- You can develop and run programs completely in MPE XL NM. NM will probably be more efficient than CM.
- Your programs can switch between CM and NM. For example, an NM program could call a CM procedure.

Because of the extended large address space available in MPE XL, paging is more efficient than segmentation. Therefore, segmentation is no longer needed.

Since the operating system pipelines instructions, MPE XL NM compilers can re-order commands to optimize efficiency. The programmer can choose optimization levels.

MPE V/E and MPE XL CM use the same processes for program development. MPE XL NM uses different, but analogous, processes. Table 2-1, following, shows the correspondences.

Table 2-1. MPE Program Development

MPE CM and MPE V/E	MPE XL NM
Compiler invocation commands	Compiler invocation commands
`PREP` command	`LINK` command
Segmenter	Linkeditor
Relocatable library (RL)	NM relocatable library (RL)
Segmented library (SL)	Executable library (XL)
Relocatable binary module (RBM)	Relocatable object module
User-defined segmented library file (USL)	Object file (OBJ)
`RUN` command	`RUN` command

Some new commands for compiling, linking, and running specific languages have been added; they are listed in Chapter 5, Using the Command Interpreter.

For further information about program development on MPE V/E and MPE XL CM, refer to *MPE Segmenter* (30000-90011). For information about program development on MPE XL, refer to *HP Link Editor/XL Reference Manual* (32650-90030).

Native Mode Program Development (new)

In Native Mode, you typically follow these steps in developing a program:

- You enter source code statements in a text file, using an editor.
- You invoke one of the MPE XL NM optimizing compilers to translate the file into machine-readable object code.
- The link editor joins this file with necessary library files, and produces an executable program file. (MPE V/E and MPE CM are somewhat different from MPE XL NM in the way they handle libraries.)
- You load and run the executable program file.

- If your program is interrupted, you can use the debugging facilities to determine what went wrong, and whether to continue.

You can compile, link, and execute with one command in most languages. There are also commands to do each of the steps separately.

Writing Source Code

Writing source code has not changed a great deal from MPE V/E. Before planning program modularity, read the section on HP Link Editor/XL, to understand how modules are linked.

Read the section on Mixed Mode Programs in this chapter if you are planning to develop a program that will switch between NM and CM. On MPE V/E and in CM the standard word is 16-bits long; in NM the standard word is 32-bits. You must be aware of differences in data alignment and floating-point real number formats.

Remember that MPE V/E supports Privilege Mode, but MPE XL NM does not.

Coding for Performance and Optimization

MPE XL NM has optimizing compilers that can reschedule machine instructions to use the system's resources more efficiently. It can not, however, alter the algorithm you use in the source code. Designing and coding your program carefully can enhance or reduce the performance of your program.

Programming Languages

- New. The following programming language is available in NM only:
  HP C/XL (NM)
- Changed. Programmers in the following languages will find compilers in both CM and NM versions:
  HP Business BASIC/V (CM) and HP Business BASIC/XL (NM)
  COBOL II/V (CM) and COBOL II/XL (NM)
  HP FORTRAN77/V (CM) and HP FORTRAN77/XL (NM)
  Pascal/V (CM) and Pascal/XL (NM)
  RPG/V (CM) and RPG/XL (NM)
- Not Used. The following MPE V/E languages are available in CM, but not in NM:
  FORTRAN 66/V (CM)
  SPL/V (CM)

Table 2-2 summarizes the program development tools available in MPE V/E and their availability in MPE XL.

Table 2-2. HP Products for Programmers on MPE XL

MPE V/E	MPE XL CM	MPE XL NM
		HP C/XL
COBOL II/V	COBOL II/V	COBOL II/XL
HP FORTRAN 77/V	HP FORTRAN 77/V	HP FORTRAN 77/XL
FORTRAN 66/V	FORTRAN 66/V	
Pascal/V	Pascal/V	HP Pascal/XL
RPG/V	RPG/V	RPG/XL
HP Business BASIC/V	HP Business BASIC/V	HP Business BASIC/XL
SPL/V	SPL/V	
NS3000/V		NS3000/XL
TurboIMAGE/V	TurboIMAGE/V	TurboIMAGE/XL
IMAGE/V		
HP SQL/V		HP SQL/XL
		ALLBASE/XL
KSAM/V	KSAM/V	KSAM/XL
Toolset/V		Toolset/XL
Transact/V	Transact/V	
Report/V	Report/V	
VPLUS/V	VPLUS/V	
System Dictionary/V	System Dictionary/V	System Dictionary/XL
Inform/V	Inform/V	
HP Access Central	HP Access Central	

MPE XL provides run-time library support for user callable library procedures. This support includes all MPE V/E and MPE XL libraries. Also, any programs compiled on MPE V/E can be executed in MPE XL CM.

Please note that in NM Toolset/XL will debug COBOL II/XL, HP Pascal/XL or HP FORTRAN 77/XL programs only. It cannot be used with HP C/XL programs.

Toolset XL cannot be used to debug CM programs. Therefore, COBOL or Pascal programs that are migrated from MPE V systems can be edited with Toolset/XL, but when they are compiled within Toolset/XL, they will be compiled with the NM version of the compiler, and only the NM version of the programs can be debugged in Toolset/XL.

If you want to develop applications on MPE XL for both MPE V and MPE XL systems, you can develop, compile, and debug your programs in NM with Toolset/XL and then recompile the source code in CM and test. Any debugging of the CM code can be done with the (non-symbolic) system debugger DEBUG.

Compiling

All HP-supported MPE V/E-based languages have run-time support, via CM libraries, on MPE XL. Some NM languages have slightly different versions than their MPE V/E counterpart compilers; all NM languages have optimizing compilers.

MPE XL Compilers

Using an MPE V/E compiler on MPE XL produces CM object code. The output of an MPE V/E compiler is a user segmented library file (USL) of relocatable binary modules (RBMs). Each RBM contains one procedure, either the main program or one of its subroutines. The compiler puts each procedure from the source into a separate RBM.

Using an MPE XL compiler produces NM object code. An MPE XL compiler produces object files composed of relocatable object modules containing common code and data for all procedures in the source file. An object module is the smallest unit that a compiler can produce and that HP Link Editor/XL can manipulate.

The object module corresponds to an RBM on MPE V/E, with the following distinctions:

- An RBM contains one, and only one, procedure. An object module can contain one or more procedures.
- An RBM can exist only as part of a USL or part of an RL. An object module is complete in itself and can stand alone as an independent file.

An object file contains one relocatable object module. Each compilation produces a single relocatable object module. A relocatable library, on the other hand, can contain multiple relocatable object modules.

Since each invocation of an NM compiler produces one object module, in order to separate object modules for two procedures into different object files, you must put the procedures in separate source files and compile them individually. You can gather them together at link time.

Compiler Libraries

On MPE V/E, the compiler run-time libraries needed by every program are stored in the system segmented library (SL). A programmer is never required to explicitly specify this library.

On MPE XL, compiler libraries are stored in a separate library, XL.PUB.SYS, the system executable library. You can specify other library files when linking the program. When you do an implicit link by using one of the following types of commands, XL.PUB.SYS is automatically included in your program's library search path:

- A command that compiles and links, such as `COB85XLK`.
- A command that compiles, links, and runs, such as `COB85XLG`.

If the compiler libraries are the only run-time libraries you need, you do not need to use the `XL` parameter at link or run time. If you do specify libraries, you do not have to mention XL.PUB.SYS; it will be searched, by default, after the names in your list. But, if you do specifically mention XL.PUB.SYS in your list, it must be the last library in your search string.

MPE XL Optimizer

There is no counterpart on an MPE V/E system to the MPE XL Optimizer, which is an integrated part of all MPE XL NM compilers.

The optimizer makes the scheduling of machine-level instructions and allocation of registers efficient. To increase efficiency, machine-level instructions may be re-ordered in your program to achieve optimal use of resources.

The three levels of code optimization are:

Level 0	(default level) provides only very simple optimizations. Use Level 0 when debugging a program that will not be used many times.
Level 1	performs local optimizations. Use Level 1 to achieve some optimization without spending too much time compiling.
Level 2	performs global optimization. It provides the greatest saving of space and time achievable with the optimizer of all the levels. The programs it produces are the most compact and run the quickest. Use Level 2 for final compilations of programs that will be run often.

In MPE XL Native Mode, HP Business BASIC /XL, HP C/XL, HP COBOL II/XL, HP FORTRAN 77/XL, and HP Pascal/XL provide optimizer options. Currently, COBOL and BASIC provide levels 0 and 1, and the others provide all three levels. Compilers set their own defaults for optimization levels.

Optimizer Assumptions

During compilation, the compiler gathers information about the use of variables and passes it to the optimizer. The 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:

- Common variables declared in this routine.
- Local variables (static and nonstatic).
- Parameters for this routine.

If you have code that violates these assumptions, the optimizer may change the behavior of the program in an undesirable way. The compiler assumes that the

program is the only process accessing its data. (HP FORTRAN 77/XL does have some exceptions.)

For detailed information on using NM optimizing compilers, refer to *HP Business BASIC/XL Reference Manual* (32715-90001), *HP C Reference Manual* (92434-90001) and *HP C/XL Reference Manual Supplement* (31506-90001), *COBOL II Reference Manual* (31500-90001) and *COBOL II/XL Reference Manual Supplement* (31500-90005), *HP FORTRAN 77/XL Reference Manual* (31501-90010), and *HP Pascal Reference Manual* (31502-90001).

Linking

In MPE XL CM, you use the MPE V Segmenter to link program modules. In NM, you link a program with the link editor. The link editor's input is an object module or a collection of object modules created by the NM compilers. The link editor's output is one of the following files:

- Executable library.
- Executable program file.
- Relocatable library.

HP Link Editor/XL

Many HP Link Editor/XL commands have counterparts in the MPE V/E Segmenter commands. However, there are some differences in the way the link editor behaves. For example, it is case sensitive and it does not implement version numbers. The counterpart of the MPE V/E `PREP` command is the MPE XL `LINK` command. `LINK` takes one or more object modules produced by Native Mode compilers and creates a program file. It invokes the HP Link Editor/XL and passes it the parameters you specify. The following are commands for the link editor, but not the segmenter:

- `COPYRL`, which copies an existing relocatable library.
- `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 library in use.
- `LINK`, a command within the HP Link Editor/XL subsystem, which works like the MPE XL system command `LINK` described below.
- `LISTPROG`, which prints the MAP of a program file to the display (by default) or to wherever the `LINKLIST` file has been directed.

The `LINK` and `PREP` commands have the same basic function. However, `LINK` has different options. `LINK` parameters can specify indirect files, which are themselves lists of other files. An indirect file must be an unnumbered file. In the `LINK` command, an indirect file can be a list of one of the following:

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

Relocatable Libraries

MPE V/E and MPE XL both use relocatable libraries (RLs). With `PREP` on an MPE V/E system, you can use only one relocatable library (RL). With `LINK` on MPE XL NM, however, you can use one or more relocatable libraries (RLs) to do the following:

- Resolve references at link time.
- Link RLs.

- Extract object modules from RLs.

The link editor can build new RLs several ways. It can use independent relocatable object modules. It can copy relocatable object modules from one RL to another. It can also extract copies of relocatable object modules from an RL and place them in a relocatable object file.

On 900 Series systems, an RL automatically expands until it reaches the maximum number of object modules it can contain. You can improve its structure and increase its size, if necessary. The link editor command `CLEANRL` can be used for garbage collection.

Unlike the MPE V/E Segmenter, when HP Link Editor/XL resolves an external reference, it merges the entire object module in the program file, even though only one procedure in that module may be required. To avoid including unreferenced procedures in an executable program file, you can compile each procedure from a separate source file to create separate object modules.

Executable Libraries

The counterpart to MPE V/E segmented libraries (SLs) are MPE XL executable libraries (XLs). SLs have the following characteristics:

- They must be in files named SL.*group.acct*, where *group* is the name of the group and *acct* is the name of the account in which the SL resides.
- The *;LIB=* parameter of the MPE V RUN command tells the order to search the SLs. Default is *LIB*=S to search system SL. Specifying *LIB*=P will search public, then system. Specifying *LIB*=G will search group, then public, then system.

XLs have the following characteristics:

- They can have any valid MPE XL filename.
- You can access many XLs when you run your program.

Program Auxiliary Header

The MPE XL program auxiliary header resides in the program files. It is generated by the link editor, and 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

If the program auxiliary header does not contain any of the above information, the loader uses the default values for the `RUN` command. For example, the program capabilities default to IA (interactive) and BA (batch) and cannot be overridden at run time.

Unlike MPE V/E, MPE XL allows you to change any of the specifications listed above at load time, except the primary entry point name. Libraries specified at run time override those specified at link time.

For detailed information, refer to *MPE Segmenter* (30000-90011) or *HP Link Editor/XL Reference Manual* (32650-90030).

Running a Program

Running a program includes creating a load module, loading the program, and executing it.

Creating a Load Module

The term "load module" refers to either an executable program file or an executable library module. It is the basic unit of code sharing.

Load modules are created by the link editor with the `LINK` command. Typical load modules are a program file, or a file containing one or more library routines.

Loading a Program

The MPE XL Loader performs the final step in preparing a program file for execution in either CM and NM. The loader accepts a program file and (optionally) a set of executable libraries as input. For CM programs, the loader accepts segmented libraries (SLs); for NM programs, it accepts executable libraries (XLs).

The loader initializes code and data and creates table entries needed to execute or access the code and data. It also creates linkages to connect program calls to procedures in the executable libraries.

The loader performs the following tasks:

- Converts code to an executable format. The load 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 XL; only read and execute access is granted to code pages. (On MPE V/E, read, write, and execute access is required because the Segment Transfer Table is written to the program file).
- Creates global data area.
- Copies global data initialization information into process private data space and sets appropriate register to point to it.

- Generates external reference list and attempts to locate all entries in the list.

The CM Loader simulates the MPE V/E loader. The following CM loader commands may be issued:

`ALLOCATE`	Issued from a session or program, loads a CM program or procedure into main memory.
`DEALLOCATE`	Deallocates a program or procedure previously loaded with the `ALLOCATE` command. A limited number of procedures can be allocated.

These commands function in the same way on MPE V/E and MPE XL CM. Some command syntax and parameter options are slightly changed. For more information, refer to Chapter 5, Using the Command Interpreter.

Using Executable Libraries

On MPE V/E you must specify libraries by choosing one of three search types: Group (G), Account (P), or System (S). The search order is: SL.Group.Account, SL.Pub.Account, SL.Pub.Sys. Specifying G, P, or S specifies where to begin the search. The default is S (System).

On MPE XL, there are two parameters to specify libraries in the `RUN` command: *LIB=* accepts the CM library search path (G, P, or S), and *XL=* accepts the NM library search path. When running CM programs, the *XL=* parameter is ignored. When running NM programs, an entry in the *XL=* parameter overrides any entry in the *LIB=* parameter.

You can specify any number of executable libraries in NM. All commands are restricted to a maximum of 280 characters, which means you may need to use an indirect file. To do this, make a list of all the library file names and save it as a text file (unnumbered). You can use this file of filenames (preceded by ^, a caret) as your parameter entry.

MPE V/E restricts file names. MPE XL allows you to use any valid file name you choose for an executable library.

In MPE XL, you do not have to list the system library; it is added to the end of the list by default. However, if it is specified, it must be the last library listed. XLs that you specify at run time override those specified at link time.

The following guidelines apply to searching executable libraries:

- The library list can be specified at link time, but is not actually used until run time.
- A run-time library name can appear only once in the `RUN` command.
- Libraries should be listed in order of increasing privilege level.
- If the system libraries are not specified, the system automatically adds XL.PUB.SYS and NL.PUB.SYS as the last libraries to search. XL.PUB.SYS, the system executable library, contains routines pertaining to purchased products, including compiler routines. NL.PUB.SYS, the native library, contains operating system routines.

UNSAT Procedure

If a program references external procedures that cannot be located, it can still be run without problems if you specify an UNSAT procedure. To do this, include the name of a dummy procedure in the UNSAT parameter of the `LINK` command. This procedure will be substituted for the missing library routine when the routine is called.

Set up the procedure with statements that facilitate program execution in the absence of the real routine. For example, the UNSAT procedure execution could print a statement informing you that it was called.

When the loader encounters the UNSAT procedure, it uses it to resolve all remaining unresolved references. A recommended programming practice is to compile the UNSAT procedure, place it in a separate library, and specify the library at the end of the *XL* parameter list. (Your UNSAT dummy must be before the compiler and system libraries. However, these libraries are not typically specified; they are added to the end of your list by default.)

Figure 2-1 shows an example of using an UNSAT procedure in a program.

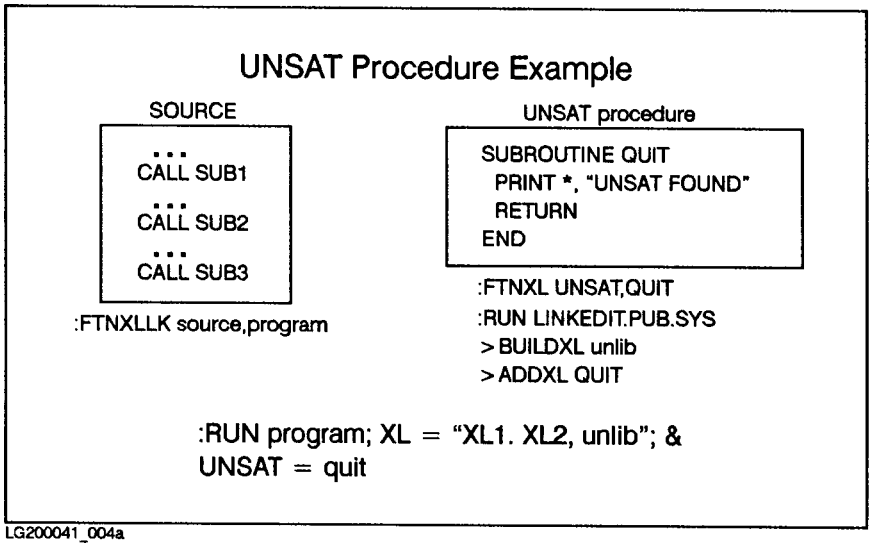

Figure 2-1. UNSAT Procedure

System Libraries

System libraries are loaded in system space. The CM system library is named SL.PUB.SYS. The NM system library is named NL.PUB.SYS. System library object modules can reference only themselves or other system library object modules. They cannot use UNSAT procedures.

Mixing Execution Modes

While the operating system is executing a process, it may switch between modes. It can alternate repeatedly between NM and CM.

MPE XL provides switch stubs to allow NM programs to access CM intrinsics. The operating system intrinsic call determines when to use the CM Segmented Library (SL) or the NM executable libraries (NL and XL). The CM intrinsic file is SPLINTR.PUB.SYS. The NM intrinsic file is SYSINTR.PUB.SYS.

You can set up an NM program to call procedures that are in a CM segmented library by using the switch intrinsics. This requires that the program specify the switch stub. The switch intrinsic uses the `LOADPROC` procedure to find the procedure. For detailed information on the Switch subsystem and

programmatic access through switch stubs, refer to *Switch Programming Guide* (32650-90014).

Errors, Aborts, and Debugging

The MPE XL System Debugger is very different from the MPE V/E debugger. It has powerful features and commands that are not available with the MPE V/E debugger.

The MPE XL debugger can be used interactively. It can be invoked with commands or intrinsics.

Your compiler lets you choose options for trap, error, and abort control. It is best to compile a program without optimization until it is debugged. Optimization can change the order of instructions, making the offsets misleading.

Common Errors

The following lists some common programming errors and offers some suggestions for correcting them.

- Bad syntax during compilation: Check manual.
- Linking problems: Check parameters.
- Missing externals at run time: Check XL list.
- Run time errors: Look for program logic error, bad data.

Not all errors produce aborts, but all reported errors should be checked.

Aborts

The decision to abort can be made by a user program, a library routine, an MPE intrinsic, or the hardware. Use the `QUIT` intrinsic to kill the current process, and take you back to the process that called it. Use `QUITPROG` to kill all the way back to the level of the command interpreter (CI).

The following lists some common abort situations and offers some suggestions for overcoming them.

- Integer overflow: Arithmetic operation needs larger target variable. May appear as value not within subrange.

- Bounds violation: Bad stack/code address. Look for bad parameters.
- Library file problems: Check *FILE* parameter to see if it was overlooked or entered incorrectly, review options/access.

When a process aborts, the MPE routines print a message or messages. Any time there is an abnormal termination, the CI prints the final line, resetting the *cierror* number. To find out more, see `HPCIERRMSG` in *Accessing Files Programmer's Guide* (32650-90017).

Run time aborts also produce a process abort trace to $STDLST. The trace starts with the procedure that terminated the program, and tracks callers back to the main program.

Offsets are given from the beginning of the procedure. Remember, the optimizer may have reordered some instructions. If you suspect this is producing confusing offset numbers, try recompiling with the optimizer off (or set to minimum). Run the program and check the offsets again.

Mixed Mode Programs (new)

Programs can be developed and compiled in MPE V/E, and migrated to run on MPE XL CM. Unless you specify otherwise, the instructions will be translated to HP-PA commands by the emulator. If you use the program very often, however, you will probably want the Object Code Translator (OCT) to make and permanently append translated code to your original program file.

The Switch subsystem allows calls to procedures or routines that operate in the alternate mode (that is, an NM program can call a CM procedure, or vice versa). A switch stub is a routine that acts as an intermediate step betweeen a calling procedure in one operating mode and a called procedure in another. The Switch Assist Tool (SWAT) can help you create switch stubs from NM to CM.

A call to a switch stub looks exactly like a call to the actual procedure in the mode of the calling procedure. Thus, cross-mode procedure calls are transparent to the calling application and its users.

For more information about creating switch stubs and SWAT, refer to *Switch Programming Guide* (32650-90014).

Running MPE V/E Programs on an MPE XL System

The MPE XL system determines whether or not to run a program in NM when the `RUN` command activates the MPE XL Loader. The loader checks the file code of the program file. If it is an NM program file or a CM program file, the loader loads the program.

If it is a CM program file, the loader provides information to the system indicating whether or not the file contains translated code. (Translated code is code that has been produced by the Object Code Translator (OCT).) The Emulator and OCT are described below. Table 2-3 describes the file codes, file names, and descriptions of MPE XL executable files.

Table 2-3. MPE XL Executable Files

FILE CODE	FILENAME	FILE CONTENT
1028	RL	Compatibility Mode Relocatable Library
1029	PROG	Compatibility Mode program file
1030	NMPRG	Native Mode program file
1031	SL	Compatibility Mode segmented library
1032	NMPRG	Native Mode executable library
1033	NMRL	Native Mode relocatable library
1034	PROG	MPE V/E translated program file
1461	NMOBJ	Native Mode Object File
1462	PASLB	HP-Pascal/XL Source Library

The system begins execution of a program file that is in NM after it is loaded. If the loaded program file is in CM, the system calls the Emulator to execute the CM code.

Emulating or Translating

If the code is not translated, then the emulator dynamically translates the MPE V/E code into equivalent HP-PA instructions.

OCT is used to translate an MPE V/E program file and produce two output files; a translated CM program file and a list file containing error messages. The translated program file is appended to a copy of the original program file, thus creating a larger file. The list file defaults to `$STDLIST`.

The primary advantage of running translated code instead of emulated code is faster performance. However, the file size is larger than that of the original untranslated file.

The MPE XL command to run OCT is `OCTCOMP`. For detailed information on translated code, refer to *Migration Process Guide* (30367-90007).

Errors, Aborts, and Debugging

Much error management and abort control depends on the language you use. Consult the manual for your programming language for techniques, statements, and translations for error messages. MPE XL subsystems produce their own error messages.

Condition code numbers for error checking are still used for some intrinsics, as in MPE V/E, but many MPE XL intrinsics use a *status* parameter instead. The *status* parameter returns a 32-bit value, interpreted as two 16-bit fields analogous to the condition code and the `PRINTFILEINFO` intrinsic. The first 16-bit field tells you if an error or warning occurred, using a code for the type of error. The second 16-bit field tells you the subsystem where the error occurred.

The *status* parameter is optional, but recommended. If you do not use the *status* parameter, and an error occurs, the program may cause an abort.

It is good programming practice to check for errors after each intrinsic call. However, you must be sure to check the right information. The *status* parameter does not effect the Condition Code setting. If you use an intrinsic that has a *status* parameter and then check the Condition Code, you will get information, but not about your intrinsic.

You can still use offsets to trace errors in programs. However, an optimizing compiler may reorder the sequence of instructions. For example, an invariant expression may have been moved outside a loop. If you get confusing results and suspect this is the case, try recompiling with the optimization level set to the minimum. This way, the sequence of instructions in your coding will be a better match to the sequence of instructions actually being executed.

The MPE XL System Debugger is entirely different from the MPE V/E Debugger. If you are debugging a CM program, *do not* follow in instructions in the MPE V/E Debug documentation. You should become familiar with *HP Symbolic Debugger User's Guide* (92435-90001) and *System Debug Reference Manual* (32650-90013).

Additional Information

For more information about intrinsics, refer to *MPE XL Intrinsics Reference Manual* (32650-90028). For more information about commands, refer to *Command Interpreter Access and Variables Programmer's Guide* (32650-90011) and *MPE XL Commands Reference Manual* (32650-90003).

For further information about program development on the MPE V/E and MPE XL CM, refer to *MPE Segmenter* (30000-90011) . For information about program development on the MPE XL NM, refer to *HP Link Editor/XL Reference Manual* (32650-90030) . For information about switching between the two, *Switch Programming Guide* (32650-90014) will be useful.

For detailed information on using NM optimizing compilers, refer to

- *HP Business BASIC/XL Reference Manual* (32715-90001)
- *HP C Reference Manual* (92434-90001) and *HP C/XL Reference Manual Supplement* (31506-90001)
- *COBOL II Reference Manual* (31500-90001) and *COBOL II/XL Reference Manual Supplement* (31500-90005)
- *HP COBOL II/XL Programmer's Guide* (31500-90002)
- *HP FORTRAN 77/XL Reference Manual* (31501-90010)
- *HP FORTRAN 77/XL Programmer's Guide* (31501-90002)

- *HP Pascal Reference Manual* (31502-90001)
- *HP Pascal Programmer's Guide* (31502-90002)
- RPG;

For further information about debugging, refer to *HP Symbolic Debugger User's Guide* (92435-90001) and *System Debug Reference Manual* (32650-90013).

3

Using Intrinsics

This chapter discusses changes in using intrinsics, such as the new access mechanism, the status parameter used for error checking, and data types. Because the word "intrinsic" has a generic meaning, as well as the specific one used in this chapter, a list of definitive criteria is also included.

You will find three lists at the end the chapter: those intrinsics that are new, changed, and not supported in MPE XL.

Overview of Differences: MPE V/E and MPE XL

A few intrinsics are supported on MPE V/E but not on MPE XL. There are also MPE XL intrinsics that are not supported on MPE V/E. For some intrinsics, there are distinct CM (Compatibility Mode) and NM (Native Mode) versions. Unless otherwise indicated, intrinsics function the same in both modes.

Most MPE V/E intrinsics are supported on MPE XL. The exceptions are `FCARD` (which supports card readers) and `PTAPE` (which supports paper tape). These have obsolete functions and are not callable in MPE XL from CM or NM. In addition, `SWITCHDB` cannot be called from NM.

On MPE V/E, you can access an intrinsic via an explicit external procedure declaration. On MPE XL, you access all system intrinsics via the intrinsic mechanism.

New

MPE XL features that are not available on MPE V/E include:

- Status Parameter (NM)

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- Intrinsic mechanism
- Generic data type mnemonics
- Intrinsics available

What is an Intrinsic?

The term **intrinsic** is often used in reference to any system or external subsystem. However, this term has a specific meaning and should be used with some care. If you are using intrinsics in an application that is intended to migrate between MPE V/E and MPE XL, you need to know which intrinsics are common to both.

To qualify as a true intrinsic (a Hewlett-Packard documented, user-callable intrinsic), an entry point must meet the following criteria:

- An intrinsic is a supported external interface to the operating system or subsystem services.
- An intrinsic interface performs type and bounds checks on parameter values before it uses them, thus protecting the operating system from the user and vice versa.
- An intrinsic interface is documented in a Hewlett-Packard manual.
- If an intrinsic is enhanced, its interface, capabilities and feature set remain backward compatible.
- An intrinsic should be callable from any Hewlett-Packard supported language. (Some data types are not available in some languages, however.)

- The callable interface of an intrinsic is different than that of other system library procedures. MPE V/E uses a SPLINTR file, and MPE/XL intrinsic mechanism uses the SYSINTR file.

Note Hewlett-Packard subsystems and applications may also provide intrinsics that meet the definition of an intrinsic. They may be documented in a separate manual and are not discussed in this manual. Refer to the *MPE XL Documentation Guide* (32650-90144).

Status Parameter (NM) (new)

It is good programming practice to check for status immediately after using each intrinsic. Be sure you check the appropriate place, or you may receive misleading information.

MPE V/E intrinsics typically used the condition code for error checking and the `PRINTFILEINFO` intrinsic for error information. The CM versions of these intrinsics use the same method. Some new intrinsics use the *status* parameter for error management instead.

Take advantage of the *status* parameter, when you use the new NM intrinsics beginning with "`HP`". Using the *status* parameter is analogous to using both condition codes and `PRINTFILEINFO`.

The *status* parameter will return a 32-bit number that you read in two parts. The first 16-bit field indicates the presence of a warning or error with a code number associated with an error message. If the first 16 bits are not zero, the second 16-bit field indicates the subsystem where the error occurred.

Generic Data Type Mnemonics (changed)

Data types used for intrinsics are represented by generic mnemonics, emphasizing that intrinsics can be called from all the programming languages. For example, if an intrinsic is listed as requiring an `I32V` parameter, you must pass, by value, a signed 32-bit integer.

The data types and their mnemonics are different in MPE V/E and MPE XL. The list of mnemonics and their meanings are listed in Table 3-1 and Table 3-2, following.

Table 3-1. Generic Data Type Mnemonics

MPE V/E SPL	MPE XL Mnemonic	Meaning
Integer	I16	16-bit signed integer.
	I32	32-bit signed integer.
	I64	64-bit signed integer.
Logical	U16	16-bit unsigned integer.
	U32	32-bit unsigned integer.
	U64	64-bit unsigned integer.
Real	R32	32-bit real.
Long	R64	64-bit real.
	R128	128-bit real.
	B	Boolean.
Byte	C	character.
	@32	32-bit address.
	@64	64-bit address.

The following mnemonics are defined to indicate parameter structures and usages:

Table 3-2. Generic Structure and Usage Mnemonics

MPE XL Mnemonic	Meaning
A	array.
REC	record.
UDS	user-defined structure.
R	passed by reference.
V	passed by value.

For more information, see Chapter 10, Converting Data Types in this manual.

Intrinsic Mechanism (changed)

The MPE XL intrinsic mechanism facilitates the declaration of system intrinsics. It is the way you gain flexible and convenient access to intrinsic routines from high-level programming languages.

An intrinsic's callable interface differs from that of other system library procedures. MPE V/E makes an intrinsic's procedure declaration available to other programming languages via its intrinsic file SPLINTR. MPE XL calls its file SYSINTR.

If you execute in Native Mode (NM), you are strongly encouraged to use the MPE XL Intrinsic Mechanism when calling system intrinsics. On MPE V/E you can also access an intrinsic via an explicit external procedure declaration, but the MPE XL intrinsic mechanism is designed to handle all access to intrinsics.

All MPE XL intrinsics are treated as external procedures by user programs. 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 can use the intrinsic mechanism to determine how to correctly invoke the routine. For example, the compiler receives information about the number of parameters, the type of each parameter, and the functional return type (if applicable).

Some advantages of using the intrinsic mechanism follow:

- Provides a consistent intrinsic interface.
- Reduces the burden and error potential of coding multiple external procedure declarations with all the parameters and type declarations required in some languages.
- Ensures proper data type conversion.
- Verifies proper data alignment.
- Provides proper indirect address references to data.
- Generates proper reference parameter addresses.
- Assigns values to default parameters and correctly resolves optional parameters omitted at the end of an intrinsic call.
- Allows parameter checking for languages that have no mechanism for detailed external declarations.
- Permits inter-language calls that might otherwise not be possible because compilers can generate the code needed to load parameters that could not otherwise be described in the language. (For example, some languages do not have call-by-value or certain data types.)
- Provides name translation for noncase-sensitive languages.
- Allows future extensibility without requiring source code changes.

Intrinsics Available (changed)

The following lists show changes to intrinsics on MPE XL.

Intrinsic Summary

The following lists show:

- new MPE XL intrinsics, not available on MPE V/E.
- intrinsics that have changed from their MPE V/E versions.
- intrinsics whose implementions are unchanged, but whose functionality may have changed due to peripheral dependencies.
- intrinsics that are not supported on MPE XL NM.

New. The following are new intrinsics on MPE XL. Unless otherwise indicated, they are available in NM only. A user with standard capabilities can call all the intrinsics in this table.

Table 3-3. New Intrinsics and Their Use

Intrinsic Name	Use/Purpose
HPCICOMMAND	Executes an MPE XL command programmatically; refer to Chapter 5, Using the Command Interpreter.
HPCIDELETEVAR	Removes an entry from the session-local variable table; refer to Chapter 5, Using the Command Interpreter.
HPCIGETVAR	Retrieves a specified, session-local variable value; refer to Chapter 5, Using the Command Interpreter.
HPCIPUTVAR	Sets the value of a session-local variable; refer to Chapter 5, Using the Command Interpreter.
HPDEBUG	Enters the system debugger and optionally executes a defined set of system debug commands; refer to Chapter 13, Debugging Applications.
HPENBLTRAP	Selectively enables or disables arithmetic traps; refer to Chapter 12, Handling Traps.
HPFIRSTLIBRARY	Returns the full name of the first library file in the binding sequence; refer to Chapter 8, Managing Resources.
HPFOPEN	Opens a file; refer to Chapter 4, Accessing Files.
HPFPCONVERT	Converts data between binary floating-point formats; refer to Chapter 10, Converting Data Types.
HPGETPROCPLABEL	Locates a procedure and returns its procedure label; refer to Chapter 8, Managing Resources.
HPLOADCMPROCEDURE	Loads the target procedure of an NM—> CM switch and returns the procedure's plabel; refer to *Switch Programming Guide* (32650-90014).
HPLOADNMPROC	Returns the procedure label of a Native Mode procedure; refer to *Switch Programming Guide* (32650-90014)
HPMERGEEND	Releases the HPMERGE program work area; refer to Chapter 11, Sorting and Merging Data.
HPMERGEERRORMESS	Retrieves and prints message if a fatal error occurs during the HPMERGE program; refer to Chapter 11, Sorting and Merging Data.
HPMERGEINIT	Initiates the HPMERGE program; refer to Chapter 11, Sorting and Merging Data.
HPMERGEOUTPUT	Retrieves records, one at a time, from the HPMERGE program; refer to Chapter 11, Sorting and Merging Data.

Table 3-3. New Intrinsics and Their Use (continued)

Intrinsic Name	Use/Purpose
HPMERGESTAT	Prints the HPMERGE program statistics on $STDLIST; refer to Chapter 11, Sorting and Merging Data.
HPMERGETITLE	Prints the version number and title of the HPMERGE subsystem on $STDLIST; refer to Chapter 11, Sorting and Merging Data.
HPMYFILE	Returns the full name of the program or library file that called it; refer to Chapter 8, Managing Resources.
HPMYPROGRAM	Returns the full name of the program being executed by this process; refer to Chapter 8, Managing Resources.
HPRESETDUMP	Disarms the system debugger call in a process abort; only the current process is affected; refer to Chapter 13, Debugging Applications.
HPSETCCODE	Sets the condition code for the calling process; refer to *Switch Programming Guide* (32650-90014).
HPSETDUMP	Arms the system debugger call for a process abort; the process can be the current process or any of its children created after calling HPSETDUMP; refer to Chapter 13, Debugging Applications.
HPSORTEND	Closes the scratch file and releases the HPSORT work area; refer to Chapter 11, Sorting and Merging Data.
HPSORTERRORMESS	Retrieves and prints message if a fatal error occurs during the HPSORT program; refer to Chapter 11, Sorting and Merging Data.
HPSORTINIT	Initiates the HPSORT program; refer to Chapter 11, Sorting and Merging Data.
HPSORTINPUT	Passes records, one at a time, to the HPSORT program; refer to Chapter 11, Sorting and Merging Data.
HPSORTOUTPUT	Retrieves records, one at a time, from the HPSORT program; refer to Chapter 11, Sorting and Merging Data.
HPSORTSTAT	Prints the HPSORT program statistics on $STDLIST; refer to Chapter 11, Sorting and Merging Data.
HPSORTTITLE	Prints the version number and title of the HPSORTLIB segment on $STDLIST; refer to Chapter 11, Sorting and Merging Data.
HPSWITCHTOCM	Makes possible NM—>CM mixed-mode procedure calls; refer to *Switch Programming Guide* (32650-90014).
HPSWTONMNAME	Makes possible CM—> NM mixed-mode procedure calls by name; refer to *Switch Programming Guide* (32650-90014).
HPSWTONMPLABEL	Makes possible CM—> NM mixed-mode procedure calls by

Changed. The following intrinsics have changed from their MPE V/E versions.

Table 3-4. Intrinsics with Implementation Differences

Intrinsic	Implementation Difference
ARITRAP (NM)	Parameter differences; refer to Chapter 12, Handling Traps.
ARITRAP (CM)	Operational differences; refer to Chapter 12, Handling Traps.
CATREAD	Parameter differences; refer to Chapter 9, Managing Message Catalogs.
COMMAND	Parameter differences; refer to Chapter 5, Using the Command Interpreter.
CREATE	Parameter and operational differences; refer to Chapter 7, Managing Processes.
CREATEPROCESS	Parameter and operational differences; refer to Chapter 7, Managing Processes.
DLSIZE (NM)	Operational differences; refer to Chapter 15, Changing Stack Sizes.
FCLOSE	Parameter differences; refer to Chapter 4, Accessing Files.
FCONTROL	Parameter differences; refer to Chapter 4, Accessing Files.
FFILEINFO	Parameter differences; refer to Chapter 6, Getting System Information.
FGETINFO	Parameter differences; refer to Chapter 6, Getting System Information.
FLABELINFO	Parameter differences; refer to Chapter 6, Getting System Information.
FOPEN	Parameter differences; refer to Chapter 4, Accessing Files.
FSETMODE	Operational differences; refer to Chapter 4, Accessing Files.
GENMESSAGE	Parameter differences; refer to Chapter 9, Managing Message Catalogs.
LOADPROC (NM)	Operational differences; refer to Chapter 8, Managing Resources.
MYCOMMAND (NM)	Byte pointer differences; refer to Chapter 5, Using the Command Interpreter.
SEARCH (NM)	Byte pointer differences; refer to Chapter 5, Using the Command Interpreter.
SETDUMP (NM)	Parameter differences; refer to Chapter 13, Debugging Applications.
STACKDUMP (NM and CM)	Parameter and operational differences; refer to Chapter 13, Debugging Applications.

Table 3-4. Intrinsics with Implementation Differences (continued)

Intrinsic	Implementation Difference
XARITRAP (CM)	Operational differences; refer to Chapter 12, Handling Traps.
XCONTRAP (NM)	Plabel differences; refer to Chapter 12, Handling Traps.
XLIBTRAP (NM)	Plabel differences; refer to Chapter 12, Handling Traps.
XLIBTRAP (CM)	Operational differences; refer to Chapter 12, Handling Traps.
XSYSTRAP (NM)	Plabel differences; refer to Chapter 12, Handling Traps.
XSYSTRAP (CM)	Operational differences; refer to Chapter 12, Handling Traps.
ZSIZE (NM)	Operational differences; refer to Chapter 15, Changing Stack Sizes.

Changed functionality. The following are intrinsics whose implementations are unchanged, but whose functionality may have changed due to peripheral dependencies:

Table 3-5. Intrinsics with Peripheral Dependencies

Intrinsic	Peripheral Dependency
FREADBACKWARD	Refer to Chapter 4, Accessing Files.
FWRITE	Refer to Chapter 4, Accessing Files.

Not Supported. FCARD, PTAPE, and SWITCHDB intrinsics are not supported on MPE XL NM.

Additional Information

For information on the intrinsics that are unmodified from their MPE V/E counterparts, refer to the MPE V/E *Intrinsics Reference* (32033-90007). For MPE XL intrinsics, refer to *MPE XL Intrinsics Reference Manual* (32650-90028).

4

Accessing Files

This chapter describes the differences between MPE XL and MPE V/E in the implementation of basic file operations. The advantages and disadvantages of mapped access are briefly discussed. A program example of access with mapped option is presented.

Overview of Differences: MPE V/E and MPE XL

Demand-paged virtual memory and a greater addressing range have caused modification to some intrinsics, but most MPE V/E file system intrinsics function the same in MPE XL. New services include access to mapped files through the Native Mode `HPFOPEN` intrinsic. Several MPE V/E file system features are not available on MPE XL based systems.

New

MPE XL features that are not available on MPE V/E include:

- Mapped access to files (NM)
- `HPFOPEN` intrinsic (NM)

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- Buffers and disk files
- Disk file placement
- Extents
- File Codes
- The following intrinsics:

FCARD (CM)	FCLOSE
FCONTROL	FFILEINFO
FGETINFO	FLABELINFO
FOPEN	FREADBACKWARD †
FSETMODE	FWRITE †
PTAPE (CM)	

† MPE V/E intrinsics whose implementations are unchanged, but whose functionality may have changed due to peripheral dependencies.

- The following commands:

LISTF
LISTFTEMP

Not Used

Existing MPE V/E features that are not supported on MPE XL include:

- Foreign disk facility
- Serial disk facility
- The following intrinsics:

FCARD (NM)
PTAPE (NM)

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

```
FCHECK                  FDELETE
FDEVICECONTROL          FERRMSG
FLOCK                   FPOINT
FREAD                   FREADDIR
FREADLABEL              FREADSEEK
FRELATE                 FRENAME
FSPACE                  FUNLOCK
FUPDATE                 FWRITEDIR
FWRITELABEL             IODONTWAIT
IOWAIT                  PRINT
PRINTFILEINFO           PRINTOP
READ                    READX
```

- `LISTEQ` command

Mapped Access to Files (NM) (new)

A major enhancement to the MPE XL file system is mapped access to files, an option that allows you to access a disk file directly through memory load and store instructions. You choose the mapped access option in NM (Native Mode programming environment) through two `HPFOPEN` intrinsic parameters:

- The *short mapped* option returns a 32-bit value of type address.
- The *long mapped* option returns a 64-bit value of type address.

Using the mapped option to access files requires the use of pointers, a datatype available in Pascal or C.

Accessing With Mapped Option

To use the mapped option to access files, declare a short (32-bit) or long (64-bit) pointer variable within a program, and pass that variable to the appropriate `HPFOPEN` parameter. When `HPFOPEN` returns the variable, it is pointing to the beginning of the data area of the opened file. After `HPFOPEN` returns the address of the file, you can simply reference the pointer as an array.

Some files restrict the type of access, like READ or READ/WRITE, that will be allowed with the mapped access option. Some can not be opened at all.

The following file types are allowed any type of access when they are opened using a mapped access option:

- standard files with fixed-length record formats.
- standard files with undefined-length record formats.

The following file types are restricted to READ-ONLY access when they are opened using a mapped access option:

- standard files with variable-length record formats.
- KSAM files opened with *copy mode* option enabled.
- standard files with variable-length record formats.

The following file types are not allowed to be opened at all with a mapped access option:

- circular (CIR) files.
- device files.
- message (MSG) files.
- relative input/output (RIO) files.
- any files with a negative file code, such as privileged files like Turboimage databases.

You can use all applicable file system intrinsics with a file that has been opened with the mapped access option, including all data transfer intrinsics. However, when mixing file system data transfer intrinsic calls (such as `FREAD` and `FWRITE`) with a mapped access option, you must consider the data type, the record format, and the record size of the file. Otherwise, the data you write to the file using the option may not make sense when it is read by `FREAD`.

When you open a file using a mapped access option and write data to that file, you must use the `FPOINT` and `FCONTROL` intrinsics to reset the EOF before

you close the file. Otherwise, all data you write to the file after the EOF will be lost when you close the file. In the case of a newly created file, the EOF initially points to record zero.

Note

When you access a file via mapped access option you are bypassing certain file system services that set various file system pointers automatically, including the EOF and the logical record pointer. You are responsible for resetting the EOF prior to closing a file you have accessed with map option.

You are also bypassing file system posting, so if data recovery is needed you should use `FCONTROL` *controlcodes* to post data and update the EOF periodically. Heavy use of the `FCONTROL` intrinsic to post data and set the EOF will degrade performance due to the overhead of the extra posting.

Advantages of Mapped Access Option

There are two perspectives you can take on mapped access to files:

- *A file is accessible as virtual memory.* The advantages from this perspective are high performance and fast response time from the file system.
- *Virtual memory is accessed through the file system.* Three advantages from this perspective are that virtual memory can be easily and permanently saved, it can be "checkpointed", and it can be easily shared through a common naming convention.

Accessing a file with a mapped option can be faster than accessing it through normal file system intrinsics. This is especially true when you are accessing a smaller file randomly rather than sequentially. With a mapped access option, there is no file system overhead associated with a specific reference to the file.

The only differences between using the mapped access option and using normal memory are the locality of the access and the protection strategy.

Disadvantages of Mapped Access Option

It is possible to show a degradation of performance if an application which accesses files sequentially is modified to access files via mapped option. Normal file system reads are buffered to pre-fetch multiple records per read. The mapped access option does minimal pre-fetching of data, and consequently some performance penalty is paid by additional overhead on page faults.

You cannot access a loaded program file or a loaded library file using mapped access option. In addition, you cannot load or execute a program file that is currently being accessed with the mapped options.

Short or Long Mapped Access?

The *short mapped* option is available in the `HPFOPEN` intrinsic to provide you with shared virtual memory. A short pointer is returned in the optional *item* parameter. You can use the pointer as a large array of any type to efficiently access the file.

The *long mapped* option is available in the `HPFOPEN` intrinsic to provide you with access to shared virtual memory. You can use the pointer as a large array of any type to efficiently access the file.

A file opened using the *short mapped* option can be up to four megabytes in size. A process can have up to six megabytes of files open at the same time that were created using the *short mapped* option. If you need more, use the *long mapped* option. A file opened using the *long mapped* option can be up to two gigabytes in size.

Using the *long mapped* access option has two advantages over the *short mapped* option. You can access much larger files. You can open files that were opened previously with any options (as long as the exclusive status of the file is not violated).

The disadvantage of using the *long mapped* access option is that it may be slower than using the *short mapped* option because of the need to load a space register to access the long pointer space. Long mapped access can be as much as four times slower than short mapped access, although the *long mapped*

option can still operate faster than accessing a file through the intrinsics for file system data transfer. The performance degradation can be minimized if you make the references to the long pointer space in a localized part of your code.

You cannot use either mapped access option to access a loaded program file or a loaded library file. In addition, you cannot load or execute a program file that is currently being accessed with either mapped access option.

If you attempt to use the *short mapped* option to open a file that has been previously opened normally or with the *long mapped* option, you will receive an error.

Mapped Access Example

The following example illustrates how a file is created and opened with the *short mapped* access option. The Pascal XL procedure opens the file, then writes data to the file via assignments to the array structure. The procedure then sets the EOF and closes the file. The file is then reopened with *short mapped option* and data is retrieved before the file is closed and purged.

```
procedure Mapped_File_Example;  {for use within a program}

type

  {** defines an 80 byte record **}
  record_t = record
               a_record : packed array [1..80] of char;
             end;

  {** define a 4,000,000 byte array **}
  file_t        = array [1..50000] of record_t;

var
  access         : integer;
  domain         : integer;
  dummy          : shortint;
  file_name      : packed array [1..20] of char;
  file_number    : integer;
  file_ptr       : ^file_t;                          {** pointer to the file **}
  filesize       : integer;
  index          : integer;
  rec            : integer;
  create_domain_perm: integer;
  domain_OLD     : integer;
  read_write_access: integer;
  status         : record
                     case integer of
                       0:    (all:    integer);
                       1:    (info:   shortint;
                              subsys:  shortint);
                   end;


const
  itemnum_2    =  2;                                 {** file name option **}
  itemnum_3    =  3;                                 {** domain option **}
  itemnum_35   =  35;                                {** filesize option **}
  itemnum_18   =  18;                                {** short mapped option**}
  itemnum_11   =  11;                                {** access type option **}
```

```
begin                                  {** initialize item values for HPFOPEN **}
     file_name           := '%EXAMPLE%';
     create_domain_PERM  := 4;
     domain_OLD          := 3;
     filesize            := 15265;
     read_write_access   := 4;


     HPFOPEN (                              {** create a short mapped file **}
               file_number, status,
               itemnum_2, file_name,
               itemnum_3, create_domain_PERM,
               itemnum_35, filesize,
               itemnum_18, file_ptr,
               itemnum_11, read_write_access,
             );


     for rec := 1 to 100 do               {** put some data into the file **}
          for index := 1 to 80 do
               file_ptr^[rec].a_record[index] := Chr (((rec - 1) Mod 26) + 65);


     FPOINT (file_number, 33);          {** set the logical record pointer **}
     FCONTROL (file_number, 6, dummy);                    {** set the EOF **}
     FCLOSE (file_number, 0, 0);                        {** close the file **}


     HPFOPEN (                        {** re-open the same short mapped file **}
               file_number, status,
               itemnum_2, file_name,
               itemnum_3, domain_OLD,
               itemnum_18, file_ptr,
             );


     for rec := 1 to 100 do          {** retrieve some data you put in file **}
        begin
           write ('Record#', rec:4, ' ');
           for index := 1 to 20 do
               write (file_ptr^[rec].a_record[index]);
           writeln;
        end;

     FCLOSE (file_number, 4, 0);              {** close and purge the file **}
end;
```

HPFOPEN (NM) (new)

The HPFOPEN intrinsic is an enhanced version of the FOPEN intrinsic. It is available in NM only. It establishes access to a file on any device and creates a file on any shareable device.

You can use the HPFOPEN intrinsic to access either a disk file or a device file. It enables you to create a new file on a shareable device and define the physical characteristics of that file prior to access. HPFOPEN returns to the calling process a file number that uniquely identifies the file. You can use the file number to reference the file in calls to other intrinsics.

HPFOPEN's optional parameters are a superset of options provided in the FOPEN intrinsic. For example, the mapped access option is available through HPFOPEN but not through FOPEN. In addition, some options available through FOPEN are enhanced in the equivalent HPFOPEN options:

- The *blockfactor* parameter allows you a maximum blocking factor of 32,767 (FOPEN allows you 255).
- The *domain option* allows you to create a file that is immediately placed in the PERMANENT file directory.
- The *access type option* allows you EXECUTE-READ access.

Note The FOPEN intrinsic is still available in MPE XL, but HPFOPEN is the recommended intrinsic for file access because it is more flexible and extensible than FOPEN. HPFOPEN is available in NM only.

HPFOPEN has an optional *status* parameter, a 32-bit signed integer passed by reference. Using the *status* parameter replaces using condition codes. The parameter has two fields. One reports the presence of errors and warnings, and the other indicates their source.

The *status* parameter is optional, but recommended. If an error or warning condition is encountered, and you did not specify the *status* parameter, your process will abort.

Buffers and Disk Files (changed)

In MPE XL, file system buffering is not performed for standard (*file type option* = STD) disk files in BUF mode. Instead, records are transferred directly between your local data area and the virtual space associated with your file. This means that, unlike MPE/V, using the blocking factor to alter the buffer size will not have an impact upon the performance of MPE XL applications.

Under MPE XL, data is transferred between the system and the data area defined in your program in the same way it was transferred under MPE/V. That is, for the standard disk files opened with the *inhibit buffering options* of `HPFOPEN/FOPEN` set to BUF (buffered mode), each read or write operation transfers one logical record. For standard disk files opened with NOBUF (unbuffered mode), each read or write operation transfers one physical record (block).

Because MPE XL does not implement system buffers for standard disk files, the *numbuffers option* of `HPFOPEN/FOPEN` is meaningless for standard disk files.

Disk File Placement (changed)

When you are using the `FOPEN` intrinsic and you specify the *deviceclass* option for disk devices in the *device* parameter, you are referring to a volume set member(s). That is, you are specifying a member, or members, of a set of disk packs, rather than a set of specific disk drives.

A file is bound to a disk pack and not to a disk device. In the `HPFOPEN` intrinsic, only volume (disk pack) or volume class may be specified for a file residing on disk.

Note When you specify *ldev* at file creation, *ldev* will be permanently associated with a specific volume that was mounted on the device specified by *ldev* at the time of file creation, and not

permanently associated with the physical device, as is the case in MPE V/E based systems.

Extents (changed)

MPE V/E and MPE XL allocate extents differently, as summarized below:

In MPE V/E:	In MPE XL:
When you build a file, at least one extent is allocated at build time.	Building a file does not necessarily allocate any disk space. Initial extents are only allocated if they are specified at creation time.
Every file takes up some disk space, even if it is empty.	Empty files take up no disk space, unless initial extents are specified when the file is created.
Files can be broken up into as many as 32 extents. The number and size of the extents is specified at the time the file is created.	Files can be broken into as many extents as are needed. The number and size of the extents need not be specified at the time the fiile is created. MPE XL will add extents to a file as required. Files can be built in one extent, or in a variable number of extents. Specifying a fixed number of extents (other than one) results in the allocation of a variable number of extents.
Every file is described by a file label, which is stored by the operating system along with the first extent.	Every file is described by a file label, which is part of a table called the File Label Table (FLT). There is one FLT for each disk volume. The file label is not part of any extent of the file.

File Codes (changed)

Following is a list of current MPE XL file codes with meanings that are predefined by Hewlett-Packard.

Table 4-2. Current MPE XL File Codes

Integer	Mnemonic	Meaning
1028	RL	Compatibility Mode Relocatable Library
1029	PROG	Compatibility Mode Program File
1030	NMPRG	Native Mode Program File
1032	NMXL	Native Mode Executable Library
1033	NMRL	Native Mode Relocatable Library
1461	NMOBJ	Native Mode Object File
1462	PASLB	Pascal XL Source Library

FCARD

The `FCARD` intrinsic is available to a program executing in CM, but the calling process is aborted when it attempts to execute the intrinsic code.

The `FCARD` intrinsic is not available to a program executing in NM.

FCLOSE (changed)

The `FCLOSE` intrinsic has been modified to incorporate enhancements to the *disposition* parameter described below.

With the *Domain Disposition* item, you can make the specified permanent standard disk file temporary (valid only for standard disk files with either fixed-length, variable-length, or undefined-length record formats). The file is moved from a permanent file directory to a temporary file directory. (You must be privileged to use this option.)

With the *Disk Space Disposition* item, you can specify that the disk space beyond the EOF marker be released to the system. The file limit remains in its current position. (Valid only for standard disk files with either fixed-length, variable-length, or undefined-length record formats.)

FCONTROL (changed)

Because of architectural differences between MPE XL and MPE V/E based machines, MPE V/E system hardware status information has no meaning in MPE XL. If you specify `FCONTROL` *controlcode* to be "read hardware status word," you will get a meaningless value returned to *param*.

You can pass a carriage control code to a line printer or terminal through the *buffer* parameter of the `FWRITE` intrinsic or the *param* parameter of the `FCONTROL` intrinsic. Some may be applicable only to peripheral devices that are not currently supported on the 900 Series HP 3000. Please be certain that the control code you pass is still applicable to the device supported on your 900 Series HP 3000.

For more information, see *MPE V/E Intrinsics Reference Manual* (32033-90007).

FFILEINFO, FGETINFO, FLABELINFO (changed)

Changes to the `FFILEINFO`, `FGETINFO`, and the `FLABELINFO` intrinsics are described in Chapter 6, Getting System Information.

FOPEN (changed)

The following `FOPEN` parameters have been modified for use on MPE XL based systems. For other `FOPEN` parameters, refer to . *MPE V/E Intrinsics Reference Manual* (32033-90007) or *MPE XL Intrinsics Reference Manual* (32650-90028).

formaldesignator — The formal file designator may contain command interpreter variables and expressions that are evaluated by `FOPEN` before the formal file designator is parsed and validated. For more information about command interpreter variables and expressions refer to the *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

foptions — Specifying an *foptions* value of zero behaves as if you did not specify *foptions*; the parameter is defaulted.

aoptions — Specifying an *aoptions* value of zero behaves as if you did not specify *aoptions*; the parameter is defaulted.

device — The `FOPEN` intrinsic does not access the special disk device classes of serial disk (SDISC) and foreign disk (FOREIGN), as neither are supported on MPE XL based systems.

When you specify the *deviceclass* option for disk devices you are referring to a member(s) of a volume set (set of disk packs) rather than a set of specific disk drives.

Also, a file is bound to a disk pack and not to a disk device. When you specify *ldev* at file creation, *ldev* will be permanently associated with a specific volume that was mounted on the device specified by *ldev* at the time of file

creation, and not permanently associated with the physical device, as is the case in MPE V/E based systems.

filesize — MPE XL NM maximum file size for Standard files is two gigabytes.

numextents, *initialloc* — When you specify a value of one in both *numextents* and in *initialloc*, the file will be created as one contiguous extent of disk space. Otherwise, a variable number of extents (with varying extent sizes) will be allocated on a need basis.

When you specify a value of one in both *initialloc* and in *numextents*, the file will be created as one contiguous extent of disk space. Otherwise, a variable number of extents (with varying extent sizes) will be allocated on a need basis.

MPE XL NM default for *initialloc* is zero extents (the default in MPE V/E is one extent). No extents are allocated to the file until the file is accessed and one or more extents are required.

MPE XL Intrinsics Reference Manual (32650-90028) lists equivalences between `FOPEN` parameters and `HPFOPEN` parameters.

FSETMODE (changed)

Two mode options, available through the *modeflags* parameter of the `FSETMODE` intrinsic, have been modified for use in MPE XL.

- You can choose to control whether or not program writes to a file are guaranteed to be completed in chronological order. In MPE XL, write requests to the file from all writers on the system can be placed on the MPE XL Serial Write Queue and guaranteed to be completed in chronological order. In MPE V/E, this would control only your program's writes to the file.

On MPE V/E, a disk file can be removed from the Serial Write Cue using `FSETMODE`. On MPE XL, a disk file that is placed on the Serial Write Queue remains on that queue until the file is purged from the system.

In MPE XL, you can choose to block program execution on each write request until the physical write operation is completed. In MPE V/E, in BUF mode, program execution will be blocked only when your write requests have filled a system buffer. Posting will by physical record (block), not per logical record.

- You can choose to have a carriage return and line feed not be issued to a terminal at the completion of each terminal read. In MPE V/E, only the issuance of the line feed will suppressed; only the carriage return is issued to the terminal after each terminal read (unless the read is in unedited mode).

FWRITE (changed)

You can pass a carriage control code to a line printer or terminal through the *buffer* parameter of the of the `FWRITE` intrinsic or the *param* parameter of the `FCONTROL` intrinsic. Some control codes may be applicable only to peripheral devices that are not currently supported on the 900 Series HP 3000. Please be certain that the control code you pass is still applicable to the device supported on your 900 Series HP 3000.

LISTF, LISTFTEMP (changed)

Changes to these commands are described in Chapter 6, Getting System Information.

FCARD, PTAPE (NM) (not used)

The `FCARD` and `PTAPE` intrinsics are not supported on MPE XL, which has no provisions for reading paper tape or cards. The intrinsics are not available in Native Mode. Although the intrinsics exist in compatibility mode, the calling process is aborted when it attempts to execute the intrinsic code.

Foreign Disc Facility (not used)

The foreign disc facility is not supported on MPE XL based systems.

FREADBACKWARD (not used)

You can use the `FREADBACKWARD` intrinsic only with magnetic tape drives that have "Read Reverse" capability. Because MPE XL does not currently support magnetic tape devices that have the "Read Reverse" capability, calls to `FREADBACKWARD` will result in an error (condition code set to CCL).

Serial Disc Facility (not used)

The serial disc facility is not supported on MPE XL-based systems.

Additional Information

For more information on programmatic control of terminal characteristics on MPE XL-based systems, refer to Appendix B of the *MPE XL Asynchronous Serial Communications System Administrator's Reference Manual* (32022-90001). An appendix discusses migration from the ATP or ADCC subsystem on an MPE V/E-based system to the Distributed Terminal Subsystem (DTS) on an MPE XL-based system.

5

Using the Command Interpreter

This chapter outlines differences between MPE V/E and MPE XL in accessing and using the programmatic interface Command Interpreter. Changes to the Command Interpreter and related intrinsics and commands are also discussed.

Three lists end the chapter: commands that are new, changed, and not supported in MPE XL.

Overview of Differences: MPE V/E and MPE XL

The Command Interpreter (CI) has undergone major changes. Enhancements to the CI make it more powerful, more flexible, and easier to use in MPE XL. Many tasks that require programs on MPE V/E can be done simply through the CI on MPE XL. You can create user command files to be accessed programmatically in MPE XL. Four new intrinsics have been added on MPE XL NM and several others are changed slightly.

New

MPE XL features that are not available on MPE V/E include:

- The following intrinsics:

`HPCICOMMAND` (NM)	`HPCIGETVAR` (NM)
`HPCIPUTVAR` (NM)	`HPCIDELETEVAR` (NM)

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- CI structure and implementation
- The following intrinsics:

  COMMAND (CM)
  MYCOMMAND (CM)
  SEARCH (CM)

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

  GETPRIVMODE
  GETUSERMODE

HPCICOMMAND (NM) (new)

You use the HPCICOMMAND intrinsic to execute an MPE XL command programmatically. Two parameters are required: a character string for the command name, and an integer for the error number. You should use the optional error parameter, as no condition codes are returned. The command in the parameter is executed as if it were directly entered in the CI.

The COMMAND intrinsic on MPE V/E and MPE XL (CM) is analgous to the HPCICOMMAND on MPE XL (NM), but HPCICOMMAND is more flexible and powerful. For example, you can not execute a UDC via COMMAND but you can with HPCICOMMAND.

HPCICOMMAND searches UDCs, command files, program files and uses the new implied RUN command. Users with PH capability can create and handle processes.

HPCIGETVAR (NM) (new)

You use the `HPCIGETVAR` intrinsic to retrieve a specified session-level variable value and/or attributes. There are two required parameters: the variable name in a character string, and an integer for error information. Optional keyword/keyvalue pairs are used to request and return information about the variable, such as its value, length or type.

HPCIPUTVAR (NM) (new)

You use the `HPCIPUTVAR` intrinsic to set the value of a session-level variable. The variable can be integer, character, or Boolean. One parameter is required: a character string containing the name of the variable you want to set. The optional keyword/keyvalue pairs specify the the value to be set and its format and type. You should use the optional error parameter, *status*, as no condition codes are returned.

HPCIDELETEVAR (NM) (new)

You use the `HPCIDELETEVAR` intrinsic to remove an entry from the session-level variable table. One parameter is required: a character string with the name of the variable you want deleted. You should use the optional error parameter, as no condition codes are returned.

CI Structure and Implementation (changed)

The MPE XL CI was designed to look the same as the MPE V/E CI, but there are some internal changes and some enhancements.

MPE V/E implemented the CI as a system process. In MPE XL, the CI is an executable program file instead, residing in PUB.SYS. This allows you to run CI.PUB.SYS as a program. Repeating the `RUN CI.PUB.SYS` command creates nested levels of CI.

Also, `CI.PUB.SYS` uses the *info=* and *parm=* parameters of the `RUN` command. The *info=* string may contain a command, and *parm=* a value that can control initiation and/or termination done by the CI.

Unlike MPE V/E, MPE XL handles scanning and parsing as a separate preliminary process. This way, errors can be detected before the CI attempts execution. String substitution is performed on each command line before the command is processed. This allows all variables, including UDC parameters, to be substituted before the command name is extracted.

Changes to the following CI features are discussed in this section:

- new command files and UDCs.
- new CI variables and JCWs.
- expression evaluator.
- command language.
- REDO facility and `REDO` command.

Command Files and User Defined Commands (changed)

There are two types of user commands that you can use to customize your environment or create personal command files. The UDCs (User Defined Commands) that exist on MPE V/E are enhanced on MPE XL. MPE XL also allows a type of user files called command files.

UDCs

UDCs are personalized files you build with sets of CI commands. UDCs are executed first, and can override or supersede MPE commands. You can build a set of files that invoke automatically at logon time to customize the user's environment.

UDC files are activated with the `SETCATALOG` command. On MPE V/E, you had to unset and then reset the catalog to change it. The new *append* and *delete* parameters on MPE XL allow you to directly add a new UDC file to a catalog or directly remove an existing UDC file from a catalog.

In MPE XL, you have two new options for UDCs: PROGRAM/NOPROGRAM and RECURSION/NORECURSION.

The PROGRAM/NOPROGRAM option allows you to choose whether your UDC is to be executable from an application.

The RECURSION/NORECURSION option determines where the CI will begin searching for a UDC. You must choose the RECURSION option for a UDC if you want it to call itself or to call any UDC that precedes it in its catalog.

The CI searches the catalog sequentially. When one UDC invokes another UDC in the session catalog *without recursion*, the CI begins searching at the end of the current UDC and continues toward the end of the file. This means it will never encounter the current UDC or any that precede it in the session catalog. When one UDC invokes another UDC in the session catalog *with recursion*, the CI begins searching at the beginning of the session catalog.

Command Files

In MPE XL, command files provide an additional method to create customized user command files.

Command files are similar to UDCs. They may accept parameters by defining them in the header line and they may use most options.

UDCs are different from command files.

- UDCs are searched before MPE commands; command files are searched after.
- UDCs have to be cataloged; command files do not.

- UDCs can be set to invoke automatically at logon; command files can not. You can cause this effect, however by having your logon UDC call a command file.

UDCs are entered in a catalog which, by default, is sequentially searched. The recursion option is required if any UDC is to call itself or any other UDC preceding in its catalog. Command file references are resolved by the file system. Each invocation of each file is treated as a separate entity, and recursion and search order is not an issue.

CI Variables (changed)

The only variables the MPE V/E CI uses are the JCWs (Job Control Words), the predefined integer variables which provide information about the status of program execution or system information.

On MPE XL, variables can be predefined or user-defined. They can be one of several types: Boolean, string, or integer.

MPE XL predefined variables provide more program execution information than on MPE V/E, including logon ID, capability lists, $STDIN and $STDLIST ldev, and CPU time used. Other predefined variables provide system information, such as time, date, job count and job fence.

Some predefined variables are used to control the user environment. Predefined variables are already defined by the system, but many can be set or changed by the user. When you set `HPAUTOCONT` to true, it is as if each command had a `CONTINUE` statement. You can set the default CI prompt with the `HPPROMPT` variable. You can time terminal reads by setting the `HPTIMEOUT` variable.

Three new commands help you manage the variables:

- `SETVAR` assigns values to variables.
- `SHOWVAR` displays variables and JCW information.
- `DELETEVAR` removes variables.

The `SETJCW` and `SHOWJCW` commands are the same in MPE V/E and MPE XL.

Expression Evaluator

The MPE XL CI has an expression evaluator, a special facility not available on MPE V/E. It evaluates arithmetic, Boolean, and string operations, variable functions, bit operations, data conversion, and some special file functions.

Dereferencing

When you pass a variable to a command, the variable may be evaluated, or dereferenced, by the expression evaluator. Every variable has both reference (a name), and a value that is assigned to it. When a variable is dereferenced, its value is substituted for its name.

Dereferencing can be implicit (done automatically) or explicit (done by request). The `CALC`, `IF`, `SETVAR` and `WHILE` commands implicitly dereference all variables you pass them. If you want to pass the value of a variable to another command, you can explicitly cause the variable to be evaluated by preceding the variable name with an exclamation point (!).

You inhibit the evaluation of character strings by enclosing them in quotation marks.

You can dereference variables recursively. You create layers of evaluation by nesting exclamation marks and quotation marks and by marking evaluation blocks with brackets. The variables will then be evaluated in sequence, left to right, in much the same way complex mathematical expressions are evaluated. Explicit dereferencing takes precedence in the sequence over implicit.

For more details on variables and a complete explanation of dereferencing, refer to *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

Calculating

Arithmetic operations include:

addition, subtraction, multiplication, and division
absolute value
modulo
exponentiation

String operations include:

concatenation
length
string extractions
case shifting

Bitwise operations include:

and
or
not
exclusive or
right or left shift

Numeric functions include converting numbers between decimal, octal and hexadecimal bases.

File information is provided via the `FINFO` intrinsic. Use it for checking existence, creation data, modification date, code number or *foptions* of a file.

Variables are evaluated by the expression evaluator, either explicitly or implicitly. The only commands that implicitly dereference expressions are `CALC`, `IF`, `SETVAR` and `WHILE`. Other expressions must explicitly request dereferencing.

Command Language

The MPE XL CI recognizes an implied `RUN` command. The command language has added some new structures, such as the while loop, recursion and a command to return program control to the calling environment. Commands have been added, and some have been changed from MPE V/E to MPE XL.

Implied Run

You can use implied `RUN` commands in MPE XL. When you enter a program file name, it acts as a command and creates the process. For example, simply entering "*progfile*" has the same effect as entering "`RUN` *progfile*".

You can control the implied `RUN` program file search and the Command file search paths used by the CI by setting the HPPATH variable. (This is not valid for data files.)

New Programming Structures (new)

MPE V/E has one conditional branch structure: the `IF..THEN..ELSE..ENDIF` set of commands. In MPE XL, the `ELSEIF` command is available as well. MPE XL enhancements also include a new `WHILE..DO..ENDWHILE` command set for loops.

In addition, UDCs in MPE XL have a *recursion* option. You can cause recursion in a command file by simply including a file name within the file itself so that the file calls itself. This is powerful, so code carefully to prevent "endless" loops.

You can use the new command `RETURN` in a UDC or command file to return control to the calling environment. Files with a `RETURN` command that are called from an application will complete and then return to the application; if the file was called from the CI, it will return to the CI.

When commands are entered, a history is kept on a stack. New commands allow you to manage the stack. `LISTREDO` displays the command-line history with each line numbered. The `DO` command accepts one of these numbers to repeat the corresponding command. For further information about `DO` and `REDO` consult *General User's Reference Manual* (32650-90002)

Command Summary

The tables below present the status of MPE XL commands as compared to MPE V/E commands for the HP 3000. The "MPE XL Status" column indicates the status of each command, and functions or features added to these commands in MPE XL.

Commands identified as "V/E" in the Mode column were functional for MPE V/E; most have been migrated to the MPE XL operating system. All can be used in the Compatibility Mode environment. If their function in Native Mode is assumed by a new command, the replacement is noted. Commands identified as "XL" are unique to MPE XL and function in the Native Mode environment.

Table 5-1. Command Summary

Command	Mode	MPE XL Status
()COMMAND LOGON	V/E	Replaced with *INFO=* parameter of HELLO command.
ABORT	V/E	Unchanged
ABORTIO (=ABORTIO)	V/E	Unchanged
ABORTJOB (=ABORTJOB)	V/E	Unchanged
ACCEPT	V/E	Unchanged
ALLOCATE	V/E	Unchanged
ALLOW	V/E	Unchanged
ALTACCT	V/E	The *volset* parameter replaced with the *volumesetname* parameter.
ALTGROUP	V/E	New *HOMEVS* parameter. The *volset* parameter replaced with *volumesetname* parameter.
ALTJOB	V/E	Unchanged
ALTLOG	V/E	Does not support SDISC or CTAPE.
ALTSEC	V/E	Unchanged
ALTSPOOLFILE	V/E	Unchanged
ALTUSER	V/E	New *acctname* parameter.
ALTVSET	V/E	Function moved to VOLUTIL.
ASSOCIATE	V/E	Unchanged
AUTOALLOCATE	V/E	Not supported

Command	Mode	MPE XL Status
BASIC	V/E	Unchanged
BASICGO	V/E	Unchanged
BASICOMP	V/E	Unchanged
BASICPREP	V/E	Unchanged
BBASIC	V/E	Unchanged. XL equivalent is BBXL.
BBASICGO	V/E	Unchanged. XL equivalent is BBXLGO.
BBASICOMP	V/E	Unchanged. XL equivalent is BBXLCOMP.
BBASICPREP	V/E	Unchanged. XL equivalent is BBXLLK.
BBXL	XL	Initiates execution of HP Business BASIC/XL interpreter.
BBXLCOMP	XL	Compiles an HP Business BASIC/XL program.
BBXLGO	XL	Compiles, links, and executes an HP Business BASIC/XL program.
BBXLLK	XL	Compiles and links an HP Business BASIC/XL program.
BREAKJOB	V/E	Unchanged
BUILD	V/E	Unchanged
BYE	V/E	Unchanged
CACHECONTROL	V/E	Not supported
CALC	XL	Evaluates an expression.
CCXL	XL	Compiles an HP C/XL program.
CCXLGO	XL	Compiles, links, and executes an HP C/XL program.
CCXLLK	XL	Compiles and links an HP C/XL program.
CHANGELOG	V/E	Does not support SDISC or CTAPE.
CHGROUP	XL	Changes the user's current group.

Command	Mode	MPE XL Status
COB74XL	XL	Compiles a COBOL II/XL program (1974 ANSI).
		New *xdbfilename* parameter.
COB74XLG	XL	Compiles, links, and executes a COBOL II/XL program (1974 ANSI).
		New *xdbfilename* parameter.
COB74XLK	XL	Compiles and links a COBOL II/XL program (1974 ANSI).
		New *xdbfilename* parameter.
COB85XL	XL	Compiles a COBOL II/XL program (1985 ANSI).
		New *xdbfilename* parameter.
COB85XLG	XL	Compiles, links, and executes a COBOL II/XL programs (1985 ANSI).
		New *xdbfilename* parameter.
COB85XLK	XL	Compiles and links a COBOL II/XL program (1985 ANSI).
		New *xdbfilename* parameter.
COBOL	V/E	Replaced by COBOLII.
COBOLGO	V/E	Replaced by COBOLIIGO.
COBOLPREP	V/E	Replaced by COBOLIIPREP.
COBOLII	V/E	Replaces COBOL. XL equivalent is COB74XL.
COBOLIIGO	V/E	Replaces COBOLGO. XL equivalent is COB74XLG.
COBOLIIPREP	V/E	Replaces COBOLPREP. XL equivalent is COB74XLK.
COMMENT	V/E	Unchanged
CONSOLE	V/E	Unchanged
CONTINUE	V/E	Unchanged
COPY	XL	Copies one disk file to another.

Command	Mode	MPE XL Status
DATA	V/E	Restricted to use in jobs only.
DEALLOCATE	V/E	Unchanged
DEBUG	V/E	New *commands* parameter.
DELETESPOOLFILE	V/E	Unchanged
DELETEVAR	XL	Deletes a specific MPE XL variable.
DISALLOW	V/E	Unchanged
DISASSOCIATE	V/E	Unchanged
DISCRPS	V/E	Unchanged
		New *value* parameter.
DISMOUNT	V/E	Unchanged. XL equivalent is VSRELEASE.
DO	XL	Reexecutes any command in the command line history stack.
DOWN	V/E	Unchanged
DOWNLOAD	V/E	Unchanged
DSTAT	V/E	Unchanged
ECHO	XL	Echoes a message to the standard list device.
EDITOR	V/E	Unchanged
ELSE	V/E	Unchanged
ELSEIF	XL	Provides an alternate sequence for an IF statement.
ENDIF	V/E	Unchanged
ENDWHILE	XL	Ends a WHILE statement.
:EOD	V/E	Use restricted to jobs only.
EOF	V/E	Not supported
EOJ	V/E	Unchanged
ERRDUMP	XL	Dumps process or system error stack to specified depth.
EXIT	XL	Terminates the command interpreter.
FCOPY	V/E	Unchanged
FILE	V/E	Unchanged
FOREIGN	V/E	Not supported

Command	Mode	MPE XL Status
FORTGO	V/E	Unchanged
FORTPREP	V/E	Unchanged
FORTRAN	V/E	Unchanged
FREERIN	V/E	Unchanged
FTN	V/E	Unchanged. XL equivalent is FTNXL.
FTNGO	V/E	Unchanged. XL equivalent is FTNXLGO.
FTNPREP	V/E	Unchanged. XL equivalent is FTNXLLK.
FTNXL	XL	Compiles a FORTRAN 77/XL program.
FTNXLGO	XL	Compiles, links, and executes a FORTRAN 77/XL program.
FTNXLLK	XL	Compiles and links a FORTRAN 77/XL program.
FULLBACKUP	V/E	Function now in the STORE command.
GETLOG	V/E	Does not support SDISC or CTAPE.
GETRIN	V/E	Unchanged
GIVE	V/E	Not supported
HEADOFF	V/E	Unchanged
HEADON	V/E	Unchanged
HELLO	V/E	New *CIPARM* and *CIINFO* parameters.
		New *termname* parameter.
HELP	V/E	Provides help in user commands and program files.
IF	V/E	Enhanced evaluation of expressions controls job/file execution with a conditional structure.
INPUT	XL	Permits interactive assignment to variables.
JOB	V/E	Unchanged
JOBFENCE	V/E	Unchanged
JOBPRI	V/E	Unchanged
JOBSECURITY	V/E	Unchanged
LDISMOUNT	V/E	Unchanged. XL equivalent is VSRELEASESYS.
LIMIT	V/E	Unchanged
LINK	XL	Merges relocatable object files to create an executable program file.

Command	Mode	MPE XL Status
LISTACCT	V/E	New *PASS* parameter and new display format.
LISTEQ	V/E	Unchanged
LISTF	V/E	New display options 3, 4, and -3. Display for option -1 modified.
		New display option 6.
LISTFTEMP	V/E	New display options 3 and -3. Display for option -1 modified.
LISTGROUP	V/E	New *PASS* parameter and new display format.
LISTLOG	V/E	Unchanged
LISTREDO	XL	Displays the contents of the command line history stack.
LISTUSER	V/E	New PASS parameter and new display format.
LISTVS	V/E	Function now in VOLUTIL.
LMOUNT	V/E	Unchanged. XL equivalent is VSRESERVESYS.
LOG	V/E	Does not support SDISC or CTAPE.
=LOGOFF	V/E	New #S*nnn*/#J*nnn* parameters to keep one job/session logged on.
=LOGON	V/E	Unchanged
MOUNT	V/E	Unchanged. XL equivalent is VSRESERVE.
NEWACCT	V/E	New *ONVS* parameter. Modified *volumesetname* parameter.
NEWGROUP	V/E	New *HOMEVS* parameter. Modified *volumesetname* parameter.
NEWUSER	V/E	New *acctname* parameter.
NEWVSET	V/E	Function now in VOLUTIL.

Command	Mode	MPE XL Status
OCTCOMP	XL	Translates MPE V-compatible code to MPE XL instructions.
OPENQ	V/E	Unchanged
OPTION	XL	Modifies environment of user-defined commands and command files.
OUTFENCE	V/E	Unchanged
PARTBACKUP	V/E	Function now in the **STORE** command.
PASCAL	V/E	Unchanged. XL equivalent is **PASXL**.
PASCALGO	V/E	Unchanged. XL equivalent is **PASXLGO**.
PASCALPREP	V/E	Unchanged. XL equivalent is **PASXLLK**.
PASXL	XL	Compiles a Pascal/XL program.
PASXLGO	XL	Compiles, links, and executes a Pascal/XL program.
PASXLLK	XL	Compiles and links a Pascal/XL program.
PAUSE	XL	Suspends current activity for specified number of seconds.
PREP	V/E	Unchanged
PREPRUN	V/E	Unchanged
PRINT	XL	Prints the contents of a file to the standard list device or to a specified file.
PTAPE	V/E	Not supported
PURGE	V/E	Unchanged
PURGEACCT	V/E	Modified *volumesetname* parameter.
PURGEGROUP	V/E	New *acctname* parameter. Modified *volumesetname* parameter.
PURGEUSER	V/E	New *acctname* parameter.
PURGEVSET	V/E	Function now in VOLUTIL.
RECALL (=RECALL)	V/E	Unchanged
REDO	V/E	New *cmdid* and *editstring* parameters.
REFUSE	V/E	Unchanged
RELEASE	V/E	Unchanged
RELLOG	V/E	Unchanged

Command	Mode	MPE XL Status
RENAME	V/E	Unchanged
REPLY (=REPLY)	V/E	Unchanged
REPORT	V/E	Modified *volumesetname* parameter.
RESET	V/E	Unchanged
RESETACCT	V/E	Unchanged
RESETDUMP	V/E	Function modified to disarm system debugger.
RESTORE	V/E	New *VOLSET*, *FCRANGE*, *LISTDIR*, *VOL*, *VOLCLASS*, and *DIRECTORY* parameters. Modified *maxfiles* and *filesetlist* parameters.
RESUME	V/E	Unchanged
RESUMEJOB	V/E	Unchanged
RESUMELOG	V/E	Unchanged
RESUMESPOOL	V/E	Unchanged
RETURN	XL	Returns execution from the current UDC or command file to the calling environment.
RPG	V/E	Unchanged. XL equivalent is RPGXL.
RPGGO	V/E	Unchanged. XL equivalent is RPGXLGO.
RPGPREP	V/E	Unchanged. XL equivalent is RPGXLLK.
RPGXL	XL	Compiles an RPG/XL program.
RPGXLGO	XL	Compiles, links, and executes an RPG/XL program.
RPGXLLK	XL	Compiles and links an RPG/XL program.
RUN	V/E	New *nmstacksize*, *nmheapsize*, *library*, *unsatproc*, and *PRI* parameters. Modified *NOPRIV*, *LMAP*, *stacksize*, *maxstack*, and *dlsize* parameters.
SAVE	V/E	Unchanged
SECURE	V/E	Unchanged
SHOWJOB	V/E	Unchanged
SET	V/E	New *ECHO*, *MSG*, and *SPEED* parameters.
SETCATALOG	V/E	New *RESET*, *APPEND*, and *DELETE* parameters.
SETDUMP	V/E	New *commands* parameter.

Command	Mode	MPE XL Status
SETJCW	V/E	Unchanged
SETMSG	V/E	Unchanged
SETVAR	XL	Assigns a value to an MPE XL variable.
SHOWALLOCATE	V/E	Not supported
SHOWALLOW	V/E	Unchanged
SHOWCACHE		Not supported
SHOWCATALOG	V/E	Modified *listfile* parameter.
SHOWCOM	V/E	Not supported
SHOWDEV	V/E	Unchanged
SHOWIN	V/E	Unchanged
SHOWJCW	V/E	Unchanged
SHOWJOB	V/E	Unchanged
SHOWLOG	V/E	Unchanged
SHOWLOGSTATUS	V/E	Unchanged
SHOWME	V/E	Unchanged.
		New banner display and USER VERSION item.
SHOWOUT	V/E	Unchanged
SHOWQ	V/E	Unchanged.
		New *ACTIVE* and *STATUS* parameters.
SHOWTIME	V/E	Unchanged
SHOWVAR	XL	Displays specified variable names and their values.
=SHUTDOWN	V/E	Unchanged
SHUTQ	V/E	Unchanged
SPEED	V/E	New *newspeed* parameter. Modified *newinspeed* and *newoutspeed* parameters.
SPL	V/E	Unchanged
SPLGO	V/E	Unchanged
SPLPREP	V/E	Unchanged

Command	Mode	MPE XL Status
STARTCACHE	V/E	Not supported
STARTSESS	V/E	New *ciinfo* and *ciparm* parameters.
STARTSPOOL	V/E	Unchanged
STOPCACHE	V/E	Not supported
STOPSPOOL	V/E	Unchanged
STORE	V/E	New *ONVS*, *STORESET*, *DIRECTORY*, *INTER*, *TRANSPORT*, and *FCRANGE* parameters. Modified *HOW*, *filesetlist*, and *storefile* parameters.
		New *MAXTAPEBUF* parameter.
STREAM	V/E	Unchanged
STREAMS	V/E	Unchanged
SUSPENDSPOOL	V/E	Unchanged
SWITCHLOG	V/E	Unchanged
SYSDUMP	V/E	Replaced with SYSGEN.
SYSGEN	XL	Starts configuration dialog and/or installation tape creation. This replaces SYSDUMP.
TAKE	V/E	Not supported
TELL	V/E	Unchanged
TELLOP	V/E	Unchanged
TUNE	V/E	Modified to ignore *minclockcycle*, which is now a default value.
UP	V/E	Unchanged
VINIT	V/E	Function moved to VOLUTIL.
VMOUNT	V/E	Unchanged
VSCLOSE	XL	Instructs the system to close a volume set.
VSOPEN	XL	Reopens a volume set close with VSCLOSE.

VSRELEASE	XL	Releases a volume set that was reserved with VSRESERVE.
VSRELEASESYS	XL	Releases a volume set system-wide.
VSRESERVE	XL	Requests operator to put volume set online and reserves the volume set for the user.
VSRESERVESYS	XL	Reserves a volume set system-wide.
VSTORE	XL	Verifies if data on backup tape is valid and reports errors incurred by STORE when writing the tape.
VSUSER	V/E	Unchanged
WARN	V/E	Unchanged
WELCOME	V/E	Unchanged
WHILE	XL	Controls job, UDC, or command file execution sequence.
XEQ	XL	Executes a program or command file and prevents MPE XL from executing a built-in command or UDC file with the same name.

REDO (changed)

MPE V/E allows you to call back your last command with the REDO command and re-execute or edit it. MPE XL has expanded the function of this command and created the REDO facility.

REDO now keeps a history of commands in a stack. You set the size of the stack with the *hpredosize* variable; default size is 20, maximum size is 1,000. Use the LISTREDO command to send the commands from the stack to the standard list device. You choose how or whether you want the displayed commands to be numbered.

Use the DO and REDO commands to re-execute a previous command on the stack. Using REDO allows you to interactively edit the command whose number you specify. The basic editing directives are the same in MPE V/E and MPE

XL, although MPE XL has some new ones, like the ability to append to the end of the line or to change one string for another.

COMMAND intrinsic (CM) (changed)

You use the `COMMAND` intrinsic in CM to execute a command from an application. If an MPE V/E command is passed that MPE XL does not support, MPE XL returns an error message (`CIERROR 9013`) stating that the command is not supported. `COMMAND` allows process creation commands, such as `RUN` or `FCOPY`, if the caller has PH capability, or if the program being run was prepared or linked with PH capability.

The *parmnum* parameter has the following significance:

- If it returns a positive value, that number is the file system error number which prevented the command from executing.
- If it returns a negative value, that number (as a positive value) is the column number where the error occurred.

MYCOMMAND (CM) and SEARCH (CM) (changed)

Because of the architectural difference between MPE V/E (16-bit) and MPE XL (32-bit) systems, the NM version of the `MYCOMMAND` intrinsic and the `SEARCH` intrinsic are incompatible with the CM versions. The *defn* parameter in both intrinsics is impacted by pointer differences between the two versions of MPE.

Additional Information

For further information, refer to the *Getting Started as an MPE XL Programmer* (32650-90008) and *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

6

Getting System Information

This section discusses modifications to some of the commands, intrinsics, and other tools you can use on MPE XL to retrieve system information.

Overview of Differences: MPE V/E and MPE XL

You can use commands and intrinsics to get system information. You can find out about system configuration, file structure, and whether a process is executing properly.

The MPE V/E programmer will not find major changes in information retrieval processes in MPE XL.

The `FREE5` utility has been replaced by the `DISCFREE` utility.

MPE V/E is based on a 16-bit word; MPE XL is based on a 32-bit word. Dumps and other readouts that were octal in MPE V/E are likely to be hexadecimal in MPE XL. Data requirements for command and intrinsic parameters may have change because MPE XL uses a 32-bit virtual addresses for files.

Some NM intrinsics have a *status* parameter, a 32-bit signed integer passed by reference, which replaces condition codes for error checking For more information about the *status* parameter, see Chapter 2, Preparing a Program for Execution.

New

MPE XL features that are not available on MPE V/E include:

- DISCFREE Utility

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- System Logging Facility
- The following intrinsics:

```
FFILEINFO
FGETINFO
FLABELINFO
```

- The following commands:

```
LISTACCT                    LISTF
LISTFTEMP                   LISTGROUP
LISTUSER                    REPORT
SHOWCATALOG
```

Not Used

Existing MPE V/E features that are not supported on MPE XL include:

- FREE5 Utility

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

```
CALENDAR                    CLOCK
DATELINE                    FCHECK
FERRMSG                     FINDJCW
FMTCALENDAR                 FMTCLOCK
FMTDATE                     FRELATE
GETJCW                      JOBINFO
PRINTFILEINFO               PROCINFO
PROCTIME                    PUTJCW
SETJCW                      TIMER
WHO
```

- The following commands:

```
LISTEQ                      RESUMELOG
SETJCW                      SHOWALLOW
SHOWDEV                     SHOWIN
SHOWJCW                     SHOWJOB
SHOWLOG                     SHOWME
SHOWOUT                     SHOWTIME
SWITCHLOG
```

DISCFREE Utility (NM) (new)

The MPE V/E program `FREE5` displays information about a system's disc free space in a histogram format . This has been replaced on MPE XL by the utility `DISCFREE`, which expands upon the functions of `FREE5`. Information is now shown either as a histogram or in a new format. The new condensed format displays the amount of free space, transient and permanent space, and the volume's total space capacity.

For more information about `DISCFREE`, refer to the description in *System Utilities Reference Manual* (32650-90081).

FFILEINFO (changed)

You can retrieve items of information by specifying up to five *itemnum/item* pairs in the parameter of FFILEINFO. A number of the item values are changed in MPE XL:

No.	Item Value
6	For standard disc files, this returns the logical device number where the device's label resides. This may not be the same logical device that contains the file's data.
13	Indicates buffered physical count of data blocks transferred. If file was opened using other than buffered access (*nobuf*), this figure will not be meaningful.
44	The number returned here indicates the number of disc extents currently allocated to the file. Extent allocation is in the order in which they are accessed, not in physical order, as in MPE V/E.
50	Returns the logical device number (ldev) of the device associated with the file.

Note In the preceding discussion, the term **standard disc file** refers to a file that is *not* a KSAM, RIO, circular, or message file. For further discussion of file types, see *Accessing Files Programmer's Guide* (32650-90017).

New Item Values

Because of the change to 32-bit addressing, it is now possible to get values for record size (Item 4), block size (Item 14), and extent size (Item 15, described below) that are greater than those allowed in MPE V/E. If the MPE V/E limit is exceeded for any of those three values, a zero (0) is returned in the appropriate item. Three new items have been added that will return the larger values available for standard disc files: Items 67, 68, and 69. (Note that Item 69 exists for compatibility reasons only. See the description of Items 15 and 16 below.)

The following new values are returned only for standard disc files:

Item 62 File lockword

Item 63 Unique File Identity (UFID) (NM only)

Item 64 File virtual address

Item 66 Global Unique File Descriptor (GUFD) pointer (NM only)

Item 67 MPE XL record size (32-bit unsigned integer)

Item 68 MPE XL block size (32-bit unsigned integer)

Item 69 MPE XL extent size (32-bit unsigned integer)

Item 74 File label virtual address (64-bit address) (NM only)

Item 75 Hardware path (32-byte character array) (NM only)

Item 76 Volume Restriction (34-byte character array) (NM only)

Item Values No Longer Valid

Because of the changed system architecture in MPE XL, some values returned by `FFILEINFO` are no longer meaningful for standard disc files. The items have been retained for compatibility reasons, and always return a zero:

Item 19 File label disc sector address

Item 40 Disc device status

In the MPE XL file system, disc file extents may be fixed or variable in length with no practical limit on their numbers. Extent values are determined by run-time access heuristics and available space. If a user specifies extent size = 0, or maximum number of extents = 0 at file creation, then size and number of extents are determined by the system. In that case the following item values are calculated using MPE defaults, and do not reflect actual values:

Item 15 Extent size

Item 16 Maximum # of extents

Hardcoded Item Values

Certain item values have been hardcoded because MPE XL does not distinguish between the devices currently supported. These items and their values are:

For standard disc files only:

Item 5	Device type/subtype (returns type=3, subtype=8)
Item 41	Device type (returns type=3)
Item 42	Device subtype (returns subtype=8)

For all disc files:

Item 7	Hardware device address (returns drt=8, unit=0)
Item 47	DRT number (returns drt=8)
Item 48	Device unit number (returns unit=0)

FGETINFO (changed)

The `FGETINFO` intrinsic now calls the `FFILEINFO` intrinsic to retrieve item values. `FGETINFO` has been retained in MPE XL for compatibility reasons only. For improved system performance, it is recommended that programmers use direct calls to `FFILEINFO` instead of using `FGETINFO`.

A number of the parameter values returned by `FGETINFO` are different for MPE XL:

lrecsize	Same as `FFILEINFO`, Item 4
devtype	Same as `FFILEINFO`, Item 5
hdaddr	Same as `FFILEINFO`, Item 7
physcount	Same as `FFILEINFO`, Item 13
blksize	Same as `FFILEINFO`, Item 14
extsize	Same as `FFILEINFO`, Item 15

numextents Same as `FFILEINFO`, Item 16

FLABELINFO (changed)

In MPE V/E, the `FFLABELINFO` intrinsic returns file label information about permanent files only. On MPE XL (CM and NM), the `FFLABELINFO` intrinsic returns file label information about both permanent and temmporary files.

Because of the change to 32-bit addressing, it is now possible to get values for record size (Item 14), block size (Item 15), and extent size (Item 18) that are greater than allowed in MPE V/E. If the MPE V/E limit is exceeded for any of those values, a zero (0) is returned in the appropriate item. Three new items have been added to allow for the larger values possible in MPE XL: Items 30, 31, and 32. Item 32 exists for compatibility reasons only; see the description of `FFILEINFO` Items 15 and 16.

The following new values are returned only for disc files that were created in a Native Mode environment:

Item 27	Unique File Identity (UFID) (NM only)
Item 28	File limit in bytes
Item 29	Start of file offset
Item 30	MPE XL record size (32-bit unsigned integer)
Item 31	MPE XL block size (32-bit unsigned integer)
Item 32	MPE XL extent size (32-bit unsigned integer)
Item 33	Lockword (8-byte) (only if creator, AM, or SM)
Item 34	Volume restriction (34-byte character array)

LISTACCT, LISTGROUP, and LISTUSER commands (changed)

These three commands have been modified slightly. These modifications are:

- A new parameter, *pass*, has been added. This parameter permits a user with the appropriate capability to view the password for the account, group, or user listed.
- The MPE V/E version of these commands gave output as an octal dump. The information displayed by these commands now appears in ASCII format, similar to the output from the MPE V/E LISTDIR5 utility. Figure 6-1 is an example of MPE XL `LISTGROUP` output:

```
LISTGROUP XL.SYS
********************
GROUP:  XL.SYS

DISC SPACE:  124824(SECTORS)     PASSWORD:
CPU TIME   : 0(SECONDS)          SECURITY--READ    : ANY
CONNECT TIME: 0(MINUTES)                   WRITE   : GU
DISC LIMIT: UNLIMITED                      APPEND  : GU
CPU LIMIT : UNLIMITED                      LOCK    : ANY
CONNECT LIMIT: 0(MINUTES)                  EXECUTE : ANY
PRIV VOL : n/a                             SAVE    : GU
FILE UFID:$0D05F001 $740998E1 $00001E20 $00000008 $0000000C
MOUNT REF CNT : n/a
HOME VOL SET : XL_SYSTEM_VOLUME_SET
CAP: BA,IA,PM,MR,DS,PH
```

Figure 6-1. Example of LISTGROUP Output

LISTF Command (changed)

File information displayed by the `LISTF` command has been modified in MPE XL. The output of `LISTF`, −1 and `LISTF`, 2 is modified. Four new listlevels provide file information comparable to MPE V/E LISTDIR5 utility displays. (LISTDIR5 is not available on MPE XL.) Changes to this command are the following:

- The −1 *listlevel* option, which showed an octal dump of the file label, now shows a hexadecimal dump.
- In MPE XL, if you enter the 2 *listlevel* option for a loaded program file, the file shows as busy (* next to file name). This does not happen on MPE V. On MPE V/E, the load bit is set within the program file, and a `LISTF` display appears normal for this program file. On MPE XL, the program file is actually opened through `FOPEN` and will appear as busy when a `LISTF` is performed while the program is loaded/ALLOCATED.
- The four new levels that have been added to the *listlevel* parameter are:

  3 Displays −1 listlevel output in ASCII format (similar to the output from the `LISTF` command in the MPE V/E LISTDIR5 utility). Access restrictions in effect for a file are not shown; use level 4 (described below) to see them.

  −3 Displays the same information as listlevel 3 and includes the file creator and lockword. The lockword and creator are displayed if the user is the creator of the file, the account manager for the file's account (a user with AM capability), or the system manager (a user with SM capability).

  4 Displays the file access restrictions in effect for the specified file (similar to the `LISTSEC` command in the MPE V/E LISTDIR5 utility). This includes account, group, and file-level security, and the logon security for the user.

  6 Displays the fully qualified file name.

Examples follow for sample outputs.

```
:LISTF JOB, 3

********************
FILE: JOB.PUB.SYS

FILE CODE : 1030      FOPTIONS: BINARY,FIXED,NOCCTL,STD
BLK FACTOR: 1         CREATOR : **
REC SIZE: 256(BYTES)  LOCKWORD: **
BLK SIZE: 256(BYTES)  SECURITY--READ    : ANY
EXT SIZE: 0(SECT)               WRITE   : ANY
NUM REC: 7816                   APPEND  : ANY
NUM SEC: 0                      LOCK    : ANY
NUM EXT: 4                      EXECUTE : ANY
MAX REC: 31250                **SECURITY IS ON
                      FLAGS   : n/a
NUM LABELS: 0         CREATED : MON, JUN  9, 1986, 9:47 AM
MAX LABELS: 0         MODIFIED: MON, JUN  9, 1986, 9:48 AM
DISC DEV #: 0         ACCESSED: WED, JUN 11, 1986, 2:38 PM
CLASS     : DISC      LABEL ADDR: **
SEC OFFSET: 0
```

Figure 6-2. Example of Output from LISTF, 3 level

```
:LISTF JOB, -3

********************
FILE: JOB.PUB.SYS

FILE CODE : 1030      FOPTIONS: BINARY,FIXED,NOCCTL,STD
BLK FACTOR: 1         CREATOR : MANAGER
REC SIZE: 256(BYTES)  LOCKWORD: PRIVATE
BLK SIZE: 256(BYTES)  SECURITY--READ    : ANY
EXT SIZE: 0(SECT)               WRITE   : ANY
NUM REC: 7816                   APPEND  : ANY
NUM SEC: 0                      LOCK    : ANY
NUM EXT: 4                      EXECUTE : ANY
MAX REC: 31250                **SECURITY IS ON
                      FLAGS   : n/a
NUM LABELS: 0         CREATED : MON, JUN  9, 1986, 9:47 AM
MAX LABELS: 0         MODIFIED: MON, JUN  9, 1986, 9:48 AM
DISC DEV #: 0         ACCESSED: WED, JUN 11, 1986, 2:38 PM
CLASS     : DISC      LABEL ADDR: $00000010 $00004414
SEC OFFSET: 0
```

Figure 6-3. Example of Output From LISTF, -3 level

```
:LISTF JOB, 4

**************************************
FILE: JOB.PUB.SYS

ACCOUNT ------  READ : ANY
               WRITE : AC
              APPEND : AC
                LOCK : ANY
             EXECUTE : ANY

GROUP --------  READ : ANY
               WRITE : GU
              APPEND : GU
                LOCK : ANY
             EXECUTE : ANY
                SAVE : GU

FILE ---------  READ : ANY                FCODE: 1030
               WRITE : ANY
              APPEND : ANY
                LOCK : ANY              **SECURITY IS ON
             EXECUTE : ANY

FOR MANAGER.SYS: READ, WRITE, LOCK, APPEND, EXECUTE,
```

Figure 6-4. Example of Output from LISTF, 4 level

LISTFTEMP Command (changed)

This command has been modified similarly to `LISTF`. There are two new levels for the *listlevel* parameter, 3 and −3, that operate exactly the same as the new levels with the same numbers in `LISTF`. There is no Level 4, however, because that information is not relevant for temporary files.

REPORT Command (changed)

The *volset* parameter of this command has been modified to accept MPE XL volume set names. In MPE V/E, volume set names were composed of *vsidname.group.account*. MPE XL volume set names consist simply of 1 to 32 characters, beginning with an alphabetic character. The remaining characters may be alphabetic, numeric, underscores, or periods.

You may still use the MPE V/E form of the name, but you must fully qualify it. You cannot specify *vsidname* alone and expect the *group* and *account* to default.

SHOWCATALOG Command (changed)

This command has been modified slightly. The MPE V/E version of the command has an optional parameter, *listfile*, that allows you to send the output to a specified file name on device class LP (line printer). The MPE XL version also permits you to use a file equation to direct the catalog listing to another disk or tape file. The default (as in MPE V/E) is that the listing is sent to the `$STDLIST` device.

In the following example, output is sent to a disc file called "udcfile".

```
:
:FILE udcfile;dev=disc
:SHOWCATALOG *udcfile
UDC CATALOG LIST SENT TO LISTFILE.
:
```

Example of SHOWCATALOG

FREE5 Utility (not used)

FREE5, an MPE V/E utility program, displays information in a histogram format about a system's disc free space. This has been replaced by an expanded MPE XL utility, DISCFREE discussed earlier in this chapter.

Additional Information

For more detailed information, refer to *Accessing Files Programmer's Guide* (32650-90017) and *Getting System Information Programmer's Guide* (32650-90018).

7

Managing Processes

This chapter describes the differences between MPE XL and MPE V/E in process management, including the following:

- New intrinsics and commands for variables.
- Stack management.
- Interrupt handling.
- *plabel* incompatabilities between CM and NM.

Overview of MPE XL and MPE V/E Differences

Processes are the basic executable entities in MPE. You can use various intrinsics and commands to create, activate, suspend, interrogate, and delete processes, and to enable communication between them. Process management intrinsics in MPE XL have been modified to deal effectively with both the CM and the NM programming environments. The Process Handling (PH) and Programmatic Sessions (PS) capability-class attributes function the same way in MPE XL as in MPE V/E.

Two architectural differences between MPE V/E and MPE XL are in the system control of stacks and the incompatibility between NM and CM *plabels*. Internal differences in file structure do not impact the operation of IPC (interprocess communication). The function of JCWs (Job Control Words) in MPE V/E is expanded on MPE XL by the addition of variables, which are managed with several new intrinsics and commands.

New

MPE XL features that are not available on MPE V/E include:

- The following intrinsics:

```
HPCIGETVAR                    HPCIPUTVAR
HPCIDELETEVAR                 HPFOPEN
```

- The following commands:

```
SETVAR
SHOWVAR
DELETEVAR
```

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- Data Stack
- The following intrinsics:

```
FCONTROL                      CREATE
CREATEPROCESS                 STARTSESS
```

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

```
ABORTSESS           ACTIVATE            CAUSEBREAK
FATHER              FCLOSE              FINDJCW
FINSTATE            FINTEXIT            FOPEN
FREAD               FWRITE              GETINFO
GETJCW              GETPRIORITY         GETPROCID
GETPROCINFO         PAUSE               PROCINFO
PROCTIME            PUTJCW              QUIT
QUITPROG            RECEIVEMAIL         SENDMAIL
SETJCW              SUSPEND             TERMINATE
```

- The following commands:

```
SETJCW
SHOWJCW
```

HPCIGETVAR, HPCIPUTVAR, and HPCIDELETEVAR Intrinsics (new)

The `HPCIGETVAR`, `HPCIPUTVAR`, and `HPCIDELETEVAR` intrinsics are new for MPE XL. They are used to retrieve, set, and delete variables in the job or session variable table. Interprocess communication between processes in the same job or session can be performed using several types of variables, and you can use these intrinsics to manipulate variables for interprocess communication. See Chapter 5, Using the Command Interpreter, in this manual, and the *MPE XL Intrinsics Reference Manual* (32650-90028) for more information about these intrinsics and their use.

HPFOPEN (new)

The MPE XL intrinsic `HPFOPEN` is analogous to the MPE XL intrinsic `FOPEN`. `HPFOPEN` is discussed generally in Chapter 4, Accessing Files, of this manual. `FOPEN` is still valid, but `HPFOPEN` is recommended because of its many additional features. `HPFOPEN` is available in NM only.

Some features of `HPFOPEN` that apply specifically to message files have changed. The options available are modified somewhat, and some of the option numbers have changed, especially those relating to access type, format, and copy mode. Consult the *MPE XL Intrinsics Reference Manual* (32650-90028) for full details.

SETVAR, SHOWVAR, and DELETEVAR Commands (new)

The `DELETEVAR`, `SETVAR`, and `SHOWVAR` commands are used to delete, to add or alter, and to show variables from the job or session Variable Table.

MPE V/E has only one type of variables for interprocess communication: JCWs (Job Control Words). On MPE XL, you can use variables as well for interprocess communication between processes in the same job or session.

These new commands are also described in Chapter 13 of this volume, and in the *MPE XL Commands Reference Manual* (32650-90003) .

Data Stack (changed)

The demand-paged virtual memory scheme and the greatly expanded addressing range of the Hewlett-Packard Precision Architecture have eliminated the need for programmatic manipulation of the NM stack. In MPE XL, the operating system itself controls the NM stack and maintains efficient program operation.

When programming in the high-level languages available in the 900 Series HP 3000 NM programming environment, all manipulations of the NM stack are accomplished by the operating system. Your NM program cannot manipulate the NM stack using optional parameters found in process management intrinsics, except when you are creating a new process. When you use the `CREATEPROCESS` intrinsic to create a new process, you are allowed to specify the maximum allowable size of the NM stack and the NM heap. Additionally, you can manipulate the CM stack if your NM program specifies a CM program when it creates a new process. When a process management intrinsic is called by a program executing in Compatibility Mode (CM), intrinsic parameters continue to manipulate the Compatibility Mode stack as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007). An exception to this rule is when your CM program creates a process that executes in NM; in this case, stack manipulation parameters described in the *MPE V/E Intrinsics Reference*

Manual are ignored, and two new optional parameters are available that allow you to specify the maximum allowable size of the NM stack and the NM heap.

CREATE (changed)

The `CREATE` intrinsic parameter functions depend on whether it is calling an NM program or by a CM program.

When the *formaldesignator* parameter specifies a CM program file, all `CREATE` parameters function as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007). All stack manipulation parameters affect the CM stack only. In addition, the NM stack is created using MPE XL NM default values.

When the *formaldesignator* parameter specifies an NM program file the *loadflags*, *stacksize*, *dlsize*, and *maxdata* parameters have modified interpretations. In addition, the CM stack is created using MPE V/E maximum default values.

The *loadflags* parameter LIBSEARCH (library search) bits reflect changes to libraries in NM. Optional sequences are available to search *XL=* parameter search string, system, group, and public libraries. The *loadflags* NOCB bits are impacted by dual mode programming. These bits control where MPE XL establishes control blocks of device files and files of type Message, RIO, and Circular. Because some operating system services are performed in CM, device files and files of these types will, by default, have control blocks established in the PCBX area of the CM stack. If you are programming in mixed mode (both NM and CM) you may anticipate the need for a larger DL to Z area in the CM stack of the process you are creating. You can use the NOCB option to indicate that control blocks of the indicated files are to be established in an extra data segment.

The *stacksize*, *dlsize*, and *maxdata* parameters are ignored, due to changes in implementation of the NM stack. The CM stack is created using MPE V/E maximum default values. The NM stack is created using MPE XL NM default values.

CREATEPROCESS (changed)

The CREATEPROCESS intrinsic is affected by the dual programming modes in MPE XL in two ways.

- It can be called from either a CM or an NM program.
- The process it creates can be either a CM or an NM process.

The CREATEPROCESS intrinsic is effected by the difference between CM and NM in word size, stack and heap management. The CM version of CREATEPROCESS, has the same *item/itemnum* options as MPE V/E, although some have changed because of the management of stacks and libraries. The NM version has more options. Table 7-1, following, summarizes the changes.

Calling from CM or NM

The following are the parameters of the CREATEPROCESS intrinsic that have been effected by word size.

createstatus
When you call CREATEPROCESS from CM, pass a 16-bit signed integer value to *createstatus* (same as MPE V/E).

When you call CREATEPROCESS from NM, pass a 32-bit signed integer value by reference to *createstatus*.

itemnums
When you call CREATEPROCESS from CM, pass an array of 16-bit signed integers by reference to *itemnums* (same as MPE V/E).

When you call CREATEPROCESS from NM, pass an array of 32-bit signed integers by reference to *itemnums*.

items
When you call CREATEPROCESS from CM, pass an array of 16-bit signed integers by reference to *item* (same as MPE V/E).

When you call CREATEPROCESS from NM, pass an array of 32-bit signed integers by reference to *item*. This modification occurs because the array may contain explicit pointer values 32-bits long in NM.

Creating a CM or NM Process

The *formaldesignator* parameter specifies the process you want to create. The following things apply whether you call `CREATEPROCESS` from an NM program or a CM program.

Creating an NM Process

When the *formaldesignator* parameter specifies a CM program file, all but two `CREATEPROCESS` intrinsic parameters and *itemnums/item* pairs function as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007).

The two new items let you to specify the maximum allowable size of both the NM stack (Item# 26) and the NM heap (Item# 27). All references to the stack in the MPE V/E manual pertain to the CM stack only. Refer to Table 7-1 for a description of both the new items.

Note When you are calling the NM version of `CREATEPROCESS` to create a CM process, you must be certain to pass the values contained in items 2 through 7 in the following manner:

Bits (0:16) Set to zero.

Bits (16:16) Passes the specified value.

Creating a CM Process

When *formaldesignator* specifies a NM program file, the *itemnums/item* pairs listed in Table 7-1 are applicable. In addition, the CM stack is created using MPE V/E maximum default values.

Table 7-1. CREATEPROCESS *itemnums,item* Descriptions

Item#	Description of Info Passed in Item
0	(Unchanged from MPE V/E.)
1	(Unchanged from MPE V/E.)
2	(Unchanged from MPE V/E.)
3	(Changed.) Refer to the description of changes to the *loadflags* parameter of the `CREATE` intrinsic, above.
4	(Changed.) This item is ignored due to changes in the implementation of the NM stack. The CM stack is created using MPE V/E maximum default values.
5	(Changed.) This item is ignored due to changes in the implementation of the NM stack. The CM stack is created using MPE V/E maximum default values.
6	(Changed.) This item is ignored due to changes in the implementation of the NM stack. The CM stack is created using MPE V/E maximum default values.
7	(Unchanged from MPE V/E.)
8	(Unchanged from MPE V/E.)
9	(Unchanged from MPE V/E.)
10	(Unchanged from MPE V/E.)
13-18	(New.) Reserved for MPE XL.

Table 7-1.
CREATEPROCESS *itemnums,item* **Descriptions (continued)**

Item#	Description of Info Passed in Item
19	(New.) *XL=* option; the address (type-coerced to an integer) of a character array containing a list of executable library files (XLs) that the NM Loader searches to satisfy unresolved external references found in *formaldesignator*. The file names (following MPE XL file naming conventions) must be separated by commas. If you specify this item, the LIBSEARCH option available in ITEM #3 is ignored.
20-22	(New.) Reserved for MPE XL.
23	(New.) The address (type-coerced to an integer) of a character array containing the name of a procedure to which unsatisfied references are linked. (The array element following the name must contain an ASCII carriage return character or a blank.)
24	(New.) The length of the `LIBLIST` array referenced by item number 19.
25	(New.) Reserved for MPE XL.
26	(New.) The address (type-coerced to an integer) of a 32-bit signed integer variable containing the maximum size, in bytes, of the NM stack.
27	(New.) The address (type-coerced to an integer) of a 32-bit signed integer variable containing the maximum size, in bytes, of the NM heap.

FCONTROL (changed)

File system interprocess communication often makes use of software interrupts. This feature allows execution of a process to be suspended while control is passed to a special interrupt handling procedure. Several file system intrinsics are used specifically for interrupt handling.

To arm or disarm software interrupts for a particular file, you use `FCONTROL` with a *controlcode* of `48`. You pass the *plabel* (external label) of your interrupt handler in the *param* parameter of this intrinsic.

Your use of this intrinsic may be affected by architectural differences between MPE V/E and MPE XL-based systems. In MPE V/E and MPE XL CM, *plabel*s are 16 bits long, in MPE XL NM they are 32 bits long. Therefore, to simplify any ambiguity, the call to `FCONTROL` (to arm your handler) should be initiated from the same mode as your handler. For example, if you have a CM interrupt handler to be armed for a particular message file, the call to `FCONTROL` to arm the handler should also be initiated in CM. Similarly, a NM interrupt handler should be armed by calling `FCONTROL` from NM. A failure to follow this rule will result in the incorrect mode setting for the *plabel*, resulting in unpredictable results when the interrupt handler is invoked (because the mode of the *plabel* will be incorrect). The best way to avoid *plabel* problems is to be sure that you arm and call your interrupt handling procedure in the same mode. If for some reason this is impossible, you can use a switch stub to call the procedure; refer to the *Switch Programming Guide* (32650-90014) for further details on this subject.

STARTSESS Command (changed)

There are new and enhanced optional parameters available to the MPE XL `HELLO` command that can be passed through the required *logonstr* parameter of `STARTSESS`. For details on these optional parameters, refer to the *MPE XL Commands Reference Manual* (32650-90003).

The required *logonstr* parameter can not specify the following `termtypes` because they are no longer supported on MPE XL due to modifications to the attached peripheral environment:

```
4, 6, 12, 13, 14, 15, 16, 19, 20
```

Additional Information

For more information, refer to the *Process Management Programmer's Guide* (32650-90023), the *Interprocess Communication Programmer's Guide* (32650-90019), and the *MPE XL Intrinsics Reference Manual* (32650-90028).

8

Managing Resources

This chapter describes differences between MPE XL and MPE V/E in the implementation of the following resource management tasks:

- Managing shared resources with RINs, so that a process sharing a resource with other processes can be guaranteed exclusive use of that resource. RINs can be programmatically acquired, locked, and unlocked using appropriate system intrinsics.
- Dynamic loading of library procedures with system intrinsics provided to enable you to dynamically load a procedure located in an executable or segmented library.

Overview of Differences: MPE V/E and MPE XL

Managing Shared Resources with RINs is the same in both MPE V/E and MPE XL based computer systems. Intrinsics related to dynamic loading of library procedures have undergone some modification in MPE XL Native Mode. In addition, new Native Mode intrinsics have been introduced to assist you with the dynamic loading of Native Mode executable library procedures.

New

MPE XL features that are not available on MPE V/E include:

- The following intrinsics:

HPFIRSTLIBRARY (NM)	HPGETPROCPLABEL(NM)
HPMYPROGRAM (NM)	HPMYFILE (NM)

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- The following intrinsics:

LOADPROC (NM)
UNLOADPROC (NM)

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

FREELOCRIN	GETLOCRIN
LOADPROC (CM)	LOCKGLORIN
LOCKLOCRIN	LOCRINOWNER
UNLOADPROC (CM)	UNLOCKGLORIN
UNLOCKLOCRIN	

- The following commands:

FREERIN
GETRIN

HPFIRSTLIBRARY (NM) (new)

The `HPFIRSTLIBRARY` intrinsic returns the fully qualified file name of the first NM executable library (XL) in the binding sequence of the calling process. You can pass this name to the `HPGETPROCPLABEL` intrinsic in the *firstfile* parameter. `HPGETPROCPLABEL` searches the files in the binding sequence for a procedure, beginning with the first XL. The first XL is the second file in the binding sequence (located immediately after the program file).

The only required parameter is the character array, *formaldesignator*, which returns the fully qualified file name of the first XL in the binding sequence of the calling process.

The *formaldesignator* parameter must be at least 28 bytes in length in order to contain the longest possible MPE XL file name, with delimiters. The lockword is not returned. The first and last characters of the returned value are blanks that act as delimiters.

The *status* parameter, a 32-bit signed integer passed by reference is optional, but recommended. It indicates status in two fields. One reports the presence of errors and warnings, and the other indicates their source. If an error or warning condition is encountered, and you did not specify the *status* parameter, `HPFIRSTLIBRARY` causes the calling process to abort. Using the *status* parameter in MPE XL is analogous to using the condition codes in MPE V/E.

Another optional parameter returns the length of the file name returned in the *formaldesignator* parameter.

HPGETPROCPLABEL (NM) (new)

The HPGETPROCPLABEL intrinsic locates a procedure found in an NM executable library file (XL) and returns its procedure label (plabel). In addition, if the procedure is not yet loaded for the process, HPGETPROCPLABEL dynamically loads the procedure.

You can then use the NM plabel to dynamically call the specified procedure, provided the programming language contains features for making dynamic procedure calls.

A plabel returned by HPGETPROCPLABEL is valid only for the duration of the calling process.

Two parameters are required: the process name, a character array, and the plabel, a 32-bit unsigned integer passed by reference.

The *status* parameter, a 32-bit signed integer passed by reference, is optional but recommended. It indicates status in two fields. One reports the presence of errors and warnings, and the other indicates their source. If an error or warning condition is encountered, and you did not specify the *status* parameter, HPGETPROCPLABEL causes the calling process to abort. Using the *status* parameter in MPE XL is analogous to using the condition codes and the PRINTFILEINFO intrinsic in MPE V/E.

You can use an optional parameter to passes the name of the program file or XL at which to begin searching. Another optional parameter indicates whether you want the process name parameter to be case-sensitive.

HPMYPROGRAM (NM) (new)

The HPMYPROGRAM intrinsic returns the fully qualified file name of the program being executed by this process. You can pass this file name to the HPGETPROCPLABEL intrinsic in the *firstfile* parameter. HPGETPROCPLABEL searches the files in the binding sequence for a procedure, beginning with the program file. The program file is the first file in the binding sequence of the calling process.

The only required parameter is the character array that returns the file name. The *formaldesignator* parameter must be at least 28 bytes in length in order to contain the longest possible MPE XL file name, with delimiters. The lockword is not returned. The first and last characters of the returned value are blanks that act as delimiters.

The *status* parameter, a 32-bit signed integer passed by reference is optional, but recommended. It indicates status in two fields. One reports the presence of errors and warnings, and the other indicates their source. If an error or warning condition is encountered, and you did not specify the *status* parameter, HPMYPROGRAM causes the calling process to abort. Using the *status* parameter in MPE XL is analogous to using the condition codes in MPE V/E.

An optional parameter returns the length of the file name returned in the *formaldesignator* parameter (including the two blanks that act as delimiters), or a zero to indicate that no file name is returned.

HPMYFILE (NM) (new)

The HPMYFILE intrinsic returns the fully qualified file name of the Native Mode program or executable library (XL) that called HPMYFILE. You can pass this file name to the HPGETPROCPLABEL intrinsic in the *firstfile* parameter. HPGETPROCPLABEL searches the files in the binding sequence of its calling process for a procedure, beginning with the file returned by HPMYFILE.

The only required parameter is the *formaldesignator*, a character array that returns the fully qualified file name of the Native Mode program or XL that

called HPMYFILE. The *formaldesignator* parameter must be at least 28 bytes in length in order to contain the longest possible MPE XL file name, with delimiters. The lockword is not returned. The first and last characters of the returned value are blanks that act as delimiters.

The *status* parameter, a 32-bit signed integer passed by reference is optional, but recommended. It indicates status in two fields. One reports the presence of errors and warnings, and the other indicates their source. If an error or warning condition is encountered, and you did not specify the *status* parameter, HPMYFILE causes the calling process to abort. Using the *status* parameter in MPE XL is analogous to using the condition codes in MPE V/E.

An optional parameter returns the length of the file name returned in the *formaldesignator* parameter (including the two blanks that act as delimiters), or a zero to indicate that no file name is returned.

LOADPROC (NM) (changed)

The LOADPROC intrinsic is used by a program executing in NM to dynamically load a procedure located in a CM segmented library. LOADPROC returns the procedure's CM plabel.

The CM plabel can then be used by the SWITCHTOCM intrinsic for calling and executing the CM library procedure from the NM program. For more information on cross-mode procedure calling, refer to *Switch Programming Guide* (32650-90014).

Note A new intrinsic, HPGETPROCPLABEL, is used by an NM program to dynamically load a procedure located in a NM executable library (XL). HPGETPROCPLABEL is not available to CM programs.

The LOADPROC intrinsic is used by a program executing in CM to dynamically load a procedure located in a CM segmented library. The CM version of the LOADPROC intrinsic performs as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007).

UNLOADPROC (NM) (changed)

The `UNLOADPROC` intrinsic is used by a program executing in NM to dynamically unload a CM segmented library (SL) procedure that was previously loaded with the `LOADPROC` intrinsic.

The `UNLOADPROC` intrinsic is used by a program executing in CM to dynamically unload a procedure located in a CM SL. The CM version of the `UNLOADPROC` intrinsic performs as described in the &90007;.

Additional Information

For more information, refer to *Resource Management Programmer's Guide* (32650-90024) and the *MPE XL Intrinsics Reference Manual* (32650-90028). For details concerning the use of the NM versions of `LOADPROC` and `UNLOADPROC` refer to *Switch Programming Guide* (32650-90014).

9

Managing Message Catalogs

This chapter briefly describes the message facilities of the MPE XL operating system: the Application Message Facility, the System Message Facility, and the Help Facility. It also summarizes similarities and differences between the MPE XL and MPE V/E implementations of these message facilities.

Overview of Differences: MPE V/E and MPE XL

The message facilities of the MPE V/E operating system are similar to those of MPE XL.

New

MPE XL features that are not available on MPE V/E include:

- `SYSCAT.PUB.SYS` intrinsics:

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- `GENMESSAGE` intrinsic (NM)

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

CATCLOSE	CATOPEN
CATREAD	NLAPPEND

SYSCAT.PUB.SYS (new)

SYSCAT.PUB.SYS contains the Native Mode system error message under MPE XL; CATALOG.PUB.SYS contains CM error messages. You may access these messages, but they are primarily for system use. Access CATALOG.PUB.SYS with the GENMESSAGE intrinsic and SYSCAT.PUB.SYS with the catalog intrinsics CATOPEN, CATREAD, and CATCLOSE. CATALOG.PUB.SYS is formatted with the MAKECAT utility; SYSCAT.PUB.SYS is formatted with the GENCAT utility.

GENMESSAGE (NM) (changed)

In MPE XL NM, the GENMESSAGE intrinsic parameters *param1* through *param5* may now pass 32-bit pointer values instead of the 16-bit pointer values used in MPE V/E systems. MPE V/E and MPE XL CM use a 16-bit word for data alignment; MPE XL NM uses a 32-bit word. As a result, these parameters may require a 32-bit value to match the Pascal integer type to be passed by value if so specified in *parmask*.

Only the five parameters have changed, and only in MPE XL NM. All the GENMESSAGE parameters in MPE XL CM, and all the other parameters in MPE XL NM, remain the same as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007).

For further information, refer to Chapter 10, Converting Data Types, and to *Message Catalogs Programmer's Guide* (32650-90021).

Additional Information

For additional discussion of these facilities, refer to the *Native Language Programmer's Guide* (32650-90022) and *Message Catalogs Programmer's Guide* (32650-90021).

10

Converting Data Types

This chapter describes the differences between MPE V/E and MPE XL data storage techniques and the implications for the programmer. The MPE V/E format of floating-point real numbers and the word size is not changed in MPE XL Compatibility Mode (CM), but is different in MPE XL Native Mode (NM) and some files will need to be converted; a sample program for conversion is included.

Overview of Differences: MPE V/E and MPE XL

Alignment for MPE V/E and MPE XL CM data structures and data variables is based on a 16-bit word. MPE XL NM aligns based on a 32-bit word. Take care that data passed between the two formats aligns properly.

MPE V/E and MPE XL CM floating-point formats for real numbers is different from MPE XL NM. A new intrinsic, `HPFCONVERT`, converts between the two floating-point formats. Since an executable module can use only one format, conversion will be necessary in mixed mode programs, or with data used across modes.

Use the same tools and commands in MPE XL that you use in MPE V/E to convert data from one type to another, such as from ASCII to binary, within a format.

New

MPE XL features that are not available on MPE V/E include:

- Data alignment (word size) (NM)
- IEEE real number floating-point format (NM)
- The `HPFCONVERT` intrinsic

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

`ASCII`	`BINARY`
`DASCII`	`DBINARY`
`CTRANSLATE`	`NLTRANSLATE`

Data Alignment (Word Size) (NM) (new)

MPE V/E and MPE XL have different word sizes. MPE V/E and MPE XL CM have a 16-bit word size; MPE XL has a 32-bit word size. Many data structures that are aligned on 16-bit boundaries in MPE V/E are aligned on 32-bit boundaries in MPE XL NM. In MPE XL NM, 32-bit data types are aligned on 32-bit boundaries, by default, to improve performance.

Converting Data between CM and NM

One way to convert data to use across formats is by choosing the appropriate compiler option. MPE XL compilers offer two directives: HP3000_16 specifies MPE V/E or MPE XL CM alignment, and HP3000_32 specifies MPE XL NM alignment.

For example, if you are compiling an application that will be run in Native Mode, but will use MPE V/E compatible data files, you would select the HP3000_16 compiler directive. This NM option causes the compiler to:

- Align data in records on 16-bit boundaries (as in MPE V/E and MPE XL CM), instead of on 32-bit boundaries (as in MPE XL NM).
- Select HP3000 format for real numbers, (as in MPE V/E and MPE XL CM), instead of IEEE floating point representation, (as in MPE XL NM.)

Using the HP3000_16 compiler directive will maintain data alignment and format compatibility with MPE V/E, but will also impact the use of NM data in the program.

For an application to use both native aligned data files and MPE V/E aligned data files, you could specify in the program record definitions to force MPE XL or MPE V/E aligned records on a structure-by-structure basis.

For example, when you need to maintain CM data alignment and format, but also need to create NM data structures, the individual structures that must be in NM format should be explicitly defined as HP3000_32 while operating under the HP3000_16 directive.

Another alternative is to create a program that reads data in one mode and writes it in another.

Data format conversion takes CPU time. To maximize performance, try to convert your files to the NM format only once.

The considerations for converting files from MPE V/E to MPE XL NM vary depending on the language. Table 10-1 and Table 10-2 show data type correspondences for MPE XL NM intrinsics and supported languages.

Table 10-1.
NM Data Types: Primitive, Generic, HP Business BASIC/XL, HP C/XL

Primitive Type	Intrinsics	HP Business BASIC/XL	HP C/XL
Character	**C**	$ dimension as 1 character	CHAR or UNSIGNED CHAR
Integer: 16-bit unsigned	**U16**	N/A	UNSIGNED SHORT INT
Integer: 32-bit unsigned	**U32**	N/A	UNSIGNED INT or UNSIGNED LONG INT
Integer: 64-bit unsigned	**U64**	N/A	N/A
Integer: 16-bit signed	**I16**	SHORTINT or subrange [−32768..32767]	SHORT INT
Integer: 32-bit signed	**I32**	INTEGER	INT or ENUM
Integer: 64-bit signed	**I64**	N/A	N/A
Real 32-bit (Single-Precision)	**R32**	SHORTREAL	FLOAT
Real 64-bit (Double-Precision)	**R64**	REAL	DOUBLE
Decimal: Packed or Unpacked	N/A	N/A	N/A
Decimal: floating-point	N/A	Short Decimal or Decimal	N/A

Table 10-2.
NM Data Types: Primitive, HP COBOL II/XL, HP FORTRAN 77/XL, and HP Pascal/XL

Primitive Type	HP COBOLII/XL & HP RPG/XL	HP FORTRAN 77/XL	HP Pascal/XL
Character	USAGE DISPLAY or group item	CHARACTER	CHARACTER
Integer: 16-bit unsigned	PIC 9 to PIC 9(4) COMP	LOGICAL OR LOGICAL*2	0..65535 or any 16-bit SUBRANGE
Integer: 32-bit unsigned	PIC 9(5) to PIC 9(9) COMP	LOGICAL OR LOGICAL*4	Any 32-bit subrange
Integer: 64-bit unsigned	PIC 9(10) to PIC 9(18) COMP	N/A	N/A
Integer: 16-bit signed	PIC S(9) to PIC S9(4) COMP	INTEGER or INTEGER *2	SHORTINT or any 16-bit subrange
Integer: 32-bit signed	PIC S9(5) to PIC S9(9) COMP	INTEGER or INTEGER *4	INTEGER or any 32-bit subrange
Integer: 64-bit signed	PIC S9(10) to PIC S9(18) COMP	N/A	N/A
Real 32-bit (Single-Precision)	N/A	REAL or REAL*4	REAL
Real 64-bit (Double-Precision)	N/A	DOUBLE PRECISION or REAL*8	LONGREAL
Decimal: Packed	USAGE COMP-3	N/A	N/A
Decimal: Unpacked		N/A	N/A

Note It is a recommended programming practice to pass character data between languages by using COBOL II alphanumeric data, HP FORTRAN 77 character variables (using $ALIAS with a specified language option), or Pascal packed arrays of char.

Converting COBOL files

Characteristics of COBOL native alignment are:

- 32-bit aligned on 01 level items, 77 level items, and SYNC items (if SYNC32 is specified).
- All other data and structures are byte-aligned.

COBOL/V and COBOL/XL have incompatibilities due to data alignment: indexed data items, and synchronized data items.

COBOL II alphanumeric data is represented as a byte array. A byte pointer to the first byte is passed by reference.

For further information, refer to *HP COBOL II/XL Migration Guide* (31500-90004).

Converting FORTRAN Files

Native alignment of some data types in HP FORTRAN 77/XL is different from that of HP FORTRAN 77/V, as shown in Table 10-3. This results in COMMON block layout and data layout differences in files.

Table 10-3. FORTRAN 77 Native Alignment

DATA TYPE	MPE V/E Alignment	MPE XL Alignment
CHARACTER	8-BIT	8-BIT
INTEGER*2 LOGICAL*2	16-BIT	16-BIT
INTEGER*4 LOGICAL*4 REAL*4	16-BIT	32-BIT
REAL*8 DOUBLE PRECISION COMPLEX*8 COMPLEX*16 COMMON	16-BIT	64-BIT

Figure 10-1 shows an example of how HP FORTRAN 77/XL aligns a COMMON block in CM and Native Mode (default).

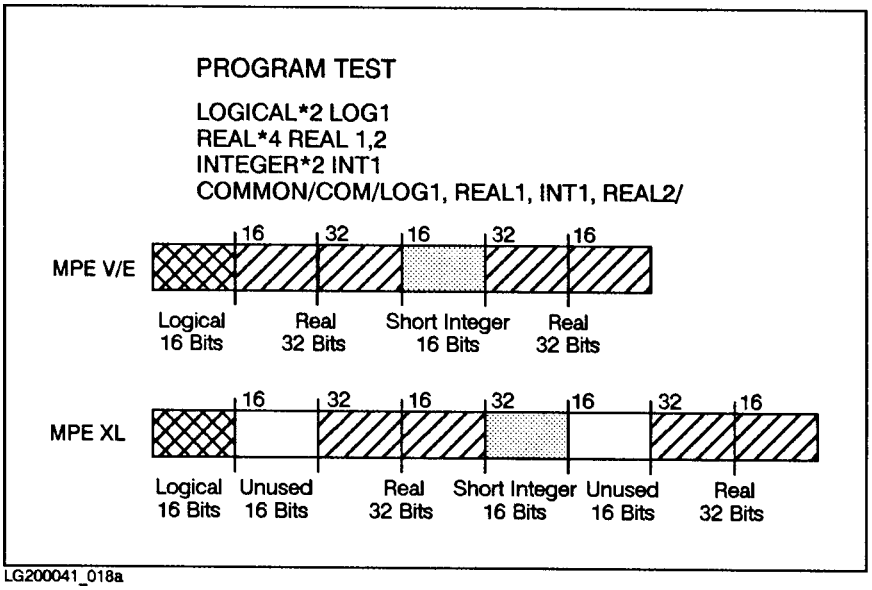

Figure 10-1. HP FORTRAN 77 COMMON Block Data Alignment Example

To align data for maximum performance, the compiler leaves a word, or a word portion, blank or unused. HP FORTRAN 77/V uses the unused 16-bit sections, and the COMMON block is more compact.

An HP FORTRAN 77 character variable is represented as a byte array. A byte pointer to the byte array and a word indicating the length of the character variable are passed by reference. You must use the ALIAS directive with a language option to make the compiler pass only a pointer to the character array.

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

1. Real 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 `HPFCONVERT` to convert real numbers from MPE V/E to MPE XL floating-point format.
4. Write the data out to a new file.

Note A program or subroutine that has $HP3000_16 ON cannot call the `HPFCONVERT` intrinsic.

For further information, refer to *HP FORTRAN 77/XL Migration Guide* (31501-90004).

Converting Pascal Files

Table 10-4 shows the alignment of simple data types on HP Pascal/V and HP Pascal/XL.

Table 10-4. Pascal Native Alignment

Variable Types	HP Pascal/V		HP Pascal/XL	
	Allocation	Alignment	Allocation	Alignment
BOOLEAN	1 byte	Byte	1 byte	Byte
CHAR	1 byte	Byte	1 byte	Byte
ENUMERATION	**1-256 elements:**		**1-256 elements:**	
ENUMERATION	1 byte	Byte	1 byte	Byte
ENUMERATION	**Any other:**		**257-64K elements**	
	2 bytes	2 byte	2 bytes	2 byte
			Over 64K elements:	
			4 bytes	4 byte
INTEGER	4 bytes	2 byte	4 bytes	4 byte
POINTER	2 bytes	2 byte	4 bytes	4 byte
REAL	4 bytes	2 byte	4 bytes	4 byte
LONGREAL	8 bytes	2 byte	8 bytes	8 byte
0..255	1 byte	Byte	1 byte	Byte
0..65535	4 bytes	2 byte	2 bytes	2 byte
-32768..32767	2 bytes	2 byte	4 bytes	4 byte
OTHERWISE	4 bytes	4 byte	4 bytes	4 byte

HP Pascal/V and HP Pascal/XL have 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.

A Pascal packed array of char (PAC) is represented as a byte array. A pointer to the first byte of the array is passed by reference. A Pascal string is represented as a byte array prefixed with a word containing the current length and suffixed with a housekeeping byte. A pointer to the word containing the current length is passed by reference.

For detailed information on independent variable allocation and alignment of data in records and other structures, refer to *HP Pascal/XL Migration Guide* (31502-90004).

Sample Conversion Program

The following example shows a conversion program that uses the `HP3000_16` compiler directive and several data types.

Sample Program (part 1 of 2)

```
$HP3000_16$
Program Convertfile(file1,file2);

CONST
  HP3000_32bit=1;
  IEEE_32bit=3;
  RoundToZero=1;

TYPE
  ARR1=ARRAY[1..10] of -32768..32767;
  CMrec=PACKED RECORD
                f1:char;
                f2:boolean;
                f3:String[40];
                f4:ARR1;
                f5:REAL;
                END;

  NMARR1=$HP3000_32$
        ARRAY[1..10] of -32768..32767;
  NMrec=$HP3000_32$
        PACKED RECORD
                f1:char;
                f2:boolean;
                f3:String[40];
                f4:NMARR1;
                f5:REAL;
                END;

VAR
  file1:FILE OF CMrec;
  file2:FILE OF NMrec;
  V1:CMrec;
  V2:NMrec;
  INX:1..10;
  status:INTEGER;
  except:-32768..32767;
```

Sample Program (part 2 of 2)

```
PROCEDURE HPFPCONVERT;  INTRINSIC;

BEGIN (*Program Convertfile*)
   RESET(file1);
   REWRITE(file2);
   WHILE NOT EOF(file1) DO
      BEGIN (*Read and Write*)
         READ(file1,V1);
         WITH V1 DO
            BEGIN (*Assign the component*)
               V2.f1:=f1;
               V2.f2:=f2;
               V2.f3:=f3;
               FOR INX:=1 to 10 DO
                  V2.f4[INX]:=f4[INX];

               HPFPCONVERT(f5,V2.f5,HP3000_32bit,IEEE_bit,status,except,
                           RoundToZero);
            END; (*Assign the component*)
         WRITE(file1,V2);
     END; (*Read and Write*)
END. (*Program Convertfile*)
```

IEEE Real Number Format (NM) (new)

MPE V/E uses HP3000 format for storing floating-point real numbers. The MPE XL operating system uses two formats: HP 3000 and IEEE, the standard set by the the American National Standards Institute and Institute of Electrical and Electronics Engineers (ANSI/IEEE Std 754-1985). In MPE XL, HP3000 is the default storage format for CM, and IEEE is the default for NM. Table 10-5 shows a summary of their characteristics.

Table 10-5. IEEE and HP 3000 Format Comparison

	IEEE	HP 3000
Single precision:		
Accuracy (in decimal digits)	7.2	6.9
Range	-3.4E38 to -1.4E-45, 0, +1.4E-45 to +3.4E38	-1.2E77 to -8.6E-78, 0, +8.6E-78 to +1.2E77
Double precision:		
Accuracy (in decimal digits)	15.9	16.5
Range	-1.8E308 to -4.9E-324, 0, +4.9E-324 to +1.8E308	-1.2E77 to -8.6E-78, 0, +8.6E-78 to +1.2E77

Note Values in this table are rounded.

Real Number Bit Format

When stored in memory, real numbers are aligned on word boundaries. MPE V/E and MPE XL CM use a 16-bit word format; MPE XL NM uses a 32-bit word format.

In MPE V/E and in MPE XL, real numbers are represented in memory by 32 bits (single-precision) or 64 bits (double-precision) format.

Represent a floating-point zero by setting all the bits to zero (0). Represent other numbers as binary numbers with three fields:

- Sign. The sign field is the first bit of the first word. A value of 0 in this position indicates the number is positive; a value of 1, that it is negative.
- Exponent. The exponent field starts on bit 1. Notice that this field is 9 bits long on MPE V/E and MPE XL CM, but on MPE XL NM it is 8 bits long for single-precision numbers and 11 bits long for double-precision numbers.
- Mantissa. The mantissa field begins after the exponent field and goes to the end of the word. Its length varies, depending on format and precision. Data is stored as a binary number of the form 1.*xxx*, where the 1 and binary point of the mantissa are not actually stored, but assumed to be present.

Figure 10-2 and Figure 10-3 show a comparison of the MPE V/E and MPE XL internal representation of single-precision and double-precision real numbers.

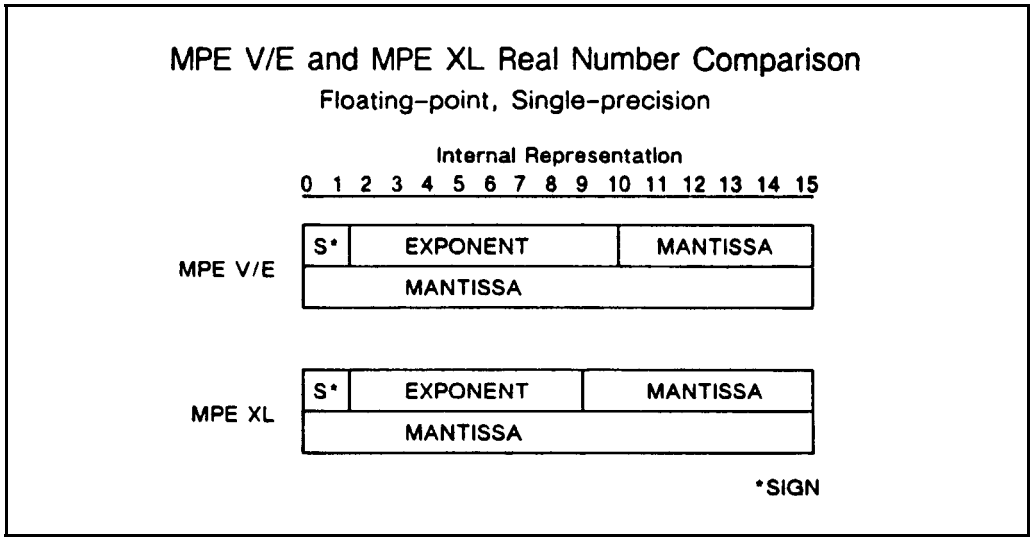

Figure 10-2. MPE V/E and MPE XL Single-Precision Real Number Comparison

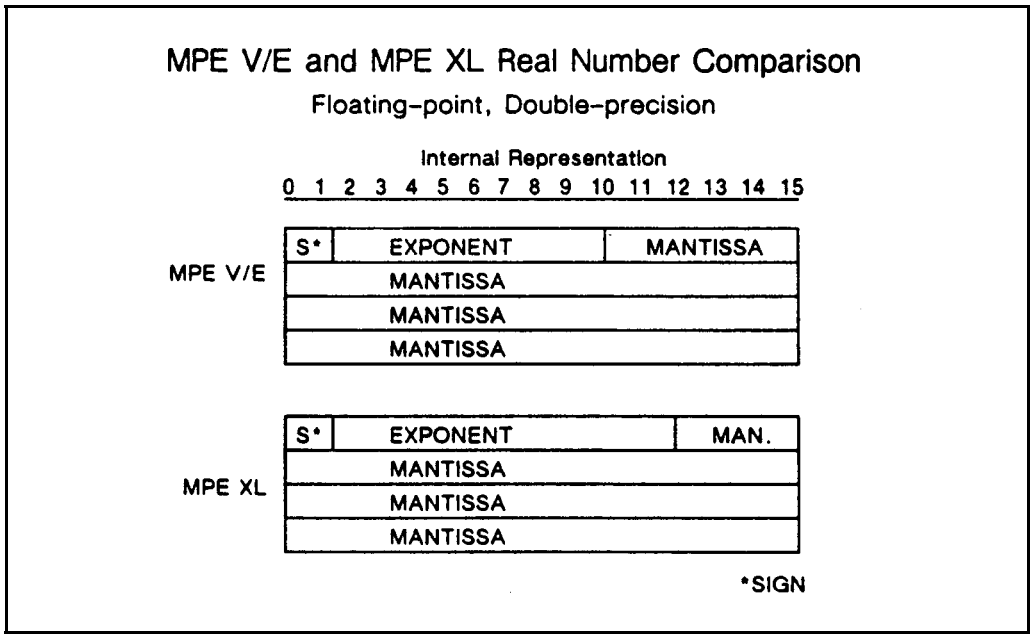

Figure 10-3. MPE V/E and MPE XL Double-Precision Real Number Comparison

Converting Numbers Between Formats

You can use the MPE XL intrinsic `HPFCONVERT` in a program to convert individual numbers between formats as necessary.

You can also force a floating-point number format with the compiler options available in MPE XL. Choosing the HP3000_16 compiler option will force the 16-bit word data alignment and the HP3000 format for real number storage compatable with MPE V/E and MPE XL CM. Specifying the HP3000_32 compiler option will align data with 32-bit words and store real numbers in IEEE format, which are the defaults for MPE XL NM.

As Table 10-5 shows, IEEE and HP3000 formats have different accuracies and ranges for single-precision and double-precision real numbers.

Conversion Exception Conditions

Table 10-6 shows the valid floating-point conversion procedures and indicates any exception to accurate conversion that can occur. These exception conditions are described in the subsections below.

Table 10-6. Floating-point Conversion Procedures

Source format	Destination format	Exception Conditions
CM32	CM32	-
CM32	CM64	-
CM32	NM32	OU
CM32	NM64	-
CM64	CM32	O
CM64	CM64	-
CM64	NM32	OUX
CM64	NM64	X
NM32	CM32	IO
NM32	CM64	IO
NM32	NM32	-
NM32	NM64	I
NM64	CM32	IOU
NM64	CM64	IOU
NM64	NM32	IOUX
NM64	NM64	-

Where:

CM32 is HP 3000 32-bit real	**I** is Invalid
CM64 is HP 3000 64-bit real	**O** is Overflow
NM32 is IEEE 32-bit real	**U** is Underflow
NM64 is IEEE 64-bit real	**X** is Inexact

Converting CM to NM

When converting CM to NM, the following exceptions could occur:

Inexact

Inexact occurs when the rounded result is not exact or an underflow or overflow occurs. In this case, the result is restricted due to limitations in exponent range and precision.

Invalid Operation (NaN)

Not a Number (NaN), is a symbolic entity encoded in IEEE floating-point format. Signaling NaNs are the values assigned to uninitialized variables. Quiet NaNs are propagated by invalid or unavailable data and/or results and provide diagnostic information.

Any operation that involves a signaling NaN or invalid operation, returns a quiet NaN as the result when no trap occurs and a floating-point result is to be delivered. An operation using one or two quiet NaNs as input signals no exception; however, if a floating-point result is to be delivered, a quiet NaN is returned that is the same as one of the input NaNs.

Note NaNs are identified by the following: the binary coded exponent field contains all 1s and the fraction field is not equal to zero. (This differs from the IEEE representation for infinity where the binary exponent field is all 1s, but the fraction field is all 0s.)

Overflow

Overflow is signaled when the destination format's largest finite number is exceeded in magnitude by the rounded floating-point result that would occur if the exponent range were unbounded. The result is determined by the rounding mode and the sign of the intermediate result as follows:

- Round to nearest carries all overflows to infinity with the sign of the intermediate result.
- Round to zero carries all overflows to the format's largest finite number with the sign of the intermediate result.

- Round toward negative infinity carries the positive overflows to the format's largest finite number and carries negative overflows to negative infinity.
- Round toward positive infinity carries the negative overflows to the format's most negative finite number and carries positive overflows to positive infinity.

Underflow

Underflow is signaled on one of two correlated events; the first is the creation of a tiny nonzero result between the smallest nonzero numbers, which may cause an overflow if used in division. The second is an extreme loss of accuracy during the approximation of tiny numbers by denormalized numbers. Tininess is detected after rounding. Loss of accuracy is detected as an inexact result.

Converting NM to CM

Converting from NM to CM could cause the following exceptions to occur:

Invalid Operation (NaN)

Invalid operation is signaled when the source is a signaling or a quiet NaN. The result is the destination format's largest finite number with the sign of the source.

Overflow

Overflow always occurs when an attempt is made to convert infinities (which are represented only in IEEE) to CM formats. When an overflow occurs, the returned result is the destination format's largest finite number with the sign of the source.

Underflow

Underflow is signaled when the magnitude of the source number (treated as if the exponent range and precision are unbounded) is less than half the magnitude of the destination format's smallest nonzero number. The result is zero.

HPFPCONVERT (new)

The `HPFPCONVERT` intrinsic converts data between MPE XL and MPE V/E binary floating-point formats. It accepts a source binary floating-point number and converts it to the equivalent destination binary floating-point number in the other MPE format. You must specify the format of the source and destination numbers. You have the option of specifying the rounding mode.

The source number must be a floating-point number, not a constant. The conversion is performed by regarding the source number as infinitely precise and with unbounded range, and then rounding it to fit the designated destination format. Rounding is performed according to the formal rules for the rounding mode specified. Rounding methods and exception signaling are determined solely from the destination format and are independent of the source format.

Conversion is performed as if all arithmetic traps are disabled. No trapping to user-supplied or system-supplied arithmetic trap routines is done.

The *status* parameter, a 32-bit signed integer passed by reference is optional, but recommended. It indicates status in two fields. One reports the presence of errors and warnings, and the other indicates their source. If an error or warning condition is encountered, and you did not specify the *status* parameter, `HPFPCONVERT`causes the calling process to abort. Using the *status* parameter in MPE XL is analogous to using condition codes in MPE V/E.

The optional parameter *exceptions*, a 16-bit signed integer passed by reference, returns any exception conditions that occurred during conversion: none, inexact, underflow and overflow, and invalid operation (NaN).

Note `HPFCONVERT` is available only on MPE XL-based systems.

Additional Information

For further details, refer to *Data Types Conversion Programmer's Guide* (32650-90015).

11

Sorting and Merging

This chapter describes the differences between MPE V/E and MPE XL in sorting and merging operations. The MPE XL Sort/Merge facility has changed, but the commands are similar. Changes in parameters are noted.

Overview of MPE XL and MPE V/E Differences

MPE XL contains the new SORT-MERGE/XL facility with HPSORT-HPMERGE intrinsics. Using the new facility is similar to using SORT-MERGE/V.

All SORT-MERGE intrinsics can be used in NM and CM, but for more efficiency in Native Mode, use the new HPSORT-HPMERGE intrinsics. The new HPSORT-HPMERGE intrinsics are used in the same way as the SORT-MERGE intrinsics, but some parameters have changed.

New

MPE XL features that are not available on MPE V/E include:

- The following intrinsics:

HPMERGEEND (NM)	HPMERGEERRORMESS (NM)
HPMERGEINIT (NM)	HPMERGEOUTPUT (NM)
HPMERGESTAT (NM)	HPMERGETITILE (NM)
HPSORTEND (NM)	HPSORTERRORMESS (NM)
HPSORTINIT	HPSORTINPUT
HPSORTOUTPUT	HPSORTSTAT
HPSORTTITLE	

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- The following intrinsics:

  MERGEINIT (NM)
  SORTINIT (NM)

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

MERGEEND	MERGEERRORMESS	MERGEINIT (CM)
MERGEOUTPUT	MERGESTAT	MERGETITLE
SORTEND	SORTERRORMESS	SORTINIT (CM)
SORTINPUT	SORTOUTPUT	SORTSTAT
SORTITLE		

New Intrinsic Overview

HPSORT-HPMERGE intrinsics are used in the same way that SORT-MERGE intrinsics are: they correspond one-to-one in their placement in programs. In NM, using HPSORT-HPMERGE intrinsics is more efficient. Although the two sets of intrinsics coordinate closely, they differ in certain areas: parameter data types, error checking, and miscellaneous parameter changes.

Data Types

MPE V/E architecture is based on a 16-bit word, and the MPE XL emulates this environment in CM. MPE XL architecture is based on a 32-bit word, and the HPSORT-HPMERGE intrinsic parameters adhere to the 32-bit standards. The 16-bit parameters you use in SORT-MERGE calls are 32-bit parameters when they are used in HPSORT-HPMERGE calls.

Error Checking

HPSORT-HPMERGE intrinsics have an optional *status* parameter, a 32-bit signed integer passed by reference. The *status* parameter is optional, but recommended. If an error or warning condition is encountered, and you did not specify the *status* parameter, your process will abort.

Using the *status* parameter is analogous to using condition codes and the `PRINTFILEINFO` intrinsic. The parameter has two 16-bit fields. One reports errors and warnings, and the other indicates the subsystem where they occurred. The *status* parameter codes are interpreted in the appendixes of *SORT-MERGE/XL Programmer's Guide* (32650-90080), and in the *MPE XL Intrinsics Reference Manual* (32650-90028).

Parameter Changes

This is an overview of the parameter differences between SORT-MERGE intrinsics and HPSORT-HPMERGE counterparts.

Every HPSORT-HPMERGE intrinsic call has *status* as its first optional parameter so that information will not be repeated for each intrinsic. The data type difference mentioned above is also not repeated for each intrinsic. For details about the HPSORT-HPMERGE intrinsics, refer to *MPE XL Intrinsics Reference Manual* (32650-90028).

HPMERGEEND and HPSORTEND (NM) (new)

Use the new *statistics* parameter in the `HPMERGEINIT`, `HPMERGEEND`, and `HPMERGESTAT` or the `HPSORTINIT`, `HPSORTEND`, and `HPSORTSTAT` intrinsics to obtain and print statistical information.

Note The *statistics* parameter does not return the number of compares for `HPSORT-MERGE` intrinsics.

HPMERGEERRORMESS and HPSORTERRORMESS (NM) (new)

In MPE V/E, you use *errorcode* for checking error conditions and returning intrinsic status. In MPE XL NM, use the *status* parameter instead.

HPMERGEINIT and HPSORTINIT (NM) (new)

The *altseq* parameter, a character array for the HPSORT-HPMERGE intrinsics, is different. The character values are the character representations of the integer array used for in the SORT-MERGE intrinsic counterparts.

You can indicate the new IEEE format for floating-point real numbers in the *keys* parameter.

The *statistics* parameter is different. *statistics* does not return the number of comparisons.

The *failure* and *errorparam* parameters are not used in the HPSORT-HPMERGE intrinsics.

The MPE V/E *spaceallocation* is now called *memsize* in MPE XL NM.

HPMERGEOUTPUT, HPSORTINPUT, and HPSORTOUTPUT (NM) (new)

Change *record* to *buffer*: The parameter name has changed, but the parameter has the same meaning.

HPMERGESTAT and HPSORTSTAT (NM) (new)

The *statistics* parameter is now required in MPE XL NM.

HPMERGETITLE and HPSORTTITILE (NM) (new)

Differences to the `HPMERGETITLE` and `HPSORTTITILE` intrinsics include only the error checking and data type differences as mentioned above for all HPSORT-HPMERGE intrinsics.

MERGEINIT (NM) (changed)

Some parameters have changed for `MERGEINIT` in native mode. Reserved parameters are: *preprocessor*, *postprocessor*, *keycompare*, *errorproc*, and *memsize*. You can not specify these parameters but you must maintain the parameter positions.

SORTINIT (NM) (changed)

Some parameters have changed for `SORTINIT` in native mode. Reserved parameters are: *keycompare*, *errorproc*, and *memsize*. You can not specify these parameters, but you must maintain the parameter positions.

Additional Information

For information on using the SORT-MERGE intrinsics, refer to *Sort-Merge Reference Guide* (32214-90001). For information about the HPSORT-HPMERGE intrinsics, refer to the *MPE XL Intrinsics Reference Manual* (32650-90028) and the *SORT-MERGE/XL Programmer's Guide* (32650-90080).

12

Handling Traps

This chapter describes the differences between MPE XL and MPE V/E in the implementation of traps and trap handling.

Overview of MPE XL and MPE V/E Differences

MPE XL provides system intrinsics to deal with the following types of trap conditions:

- Arithmetic traps
- CONTROL-Y traps
- Software library traps
- Software system traps

When a trap condition is detected during process execution, control is transferred to the MPE XL trap handling subsystem. Normally, a system-provided trap handler aborts the process and outputs an appropriate error message. Some system intrinsics allow you to specify a user-created trap handler that replaces the system trap handler.

New arithmetic trap conditions are available in the Native Mode (NM) programming environment of MPE XL, requiring parameter changes in the NM versions of arithmetic trap handling intrinsics. A new NM intrinsic, `HPENBLTRAP`, lets you selectively enable or disable arithmetic traps. NM trap handling intrinsics that pass and return plabels have been modified to handle 32-bit NM plabels.

New

MPE XL features that are not available on MPE V/E include:

- Native Mode arithmetic traps
- NM and CM Traps Differences
- `HPENBLTRAP` intrinsic (NM)

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- NM and CM trap differences
- The following intrinsics:

`ARITRAP` (NM)	`XARITRAP` (NM)
`XCONTRAP` (NM)	`XLIBTRAP` (NM)
`XSYSTRAP` (NM)	

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

`ARITRAP` (CM)	`RESETCONTROL`
`XARITRAP` (CM)	`XCONTRAP` (CM)
`XLIBTRAP` (CM)	`XSYSTRAP` (CM)

Native Mode Arithmetic Traps (new)

The following modifications have been made to arithmetic trap handling in the MPE XL NM Programming environment:

- Two trap conditions still available to MPE V/E and MPE XL CM applications are not available in NM.
- Twelve new arithmetic trap conditions are available to NM applications.
- NM compiler directives are available to disable arithmetic traps.

NM versions of the `ARITRAP` and `XARITRAP` intrinsics have been modified to handle the new NM arithmetic trap handling environment. In addition, a new NM intrinsic, `HPENBLTRAP`, lets you selectively enable or disable arithmetic traps, providing you with more flexibility than the `ARITRAP` intrinsic.

Note

The following apply to various arithmetic trap conditions available to NM applications (accessed through the `HPENBLTRAP`, `ARITRAP`, and `XARITRAP` intrinsics):

- The results of disabling arithmetic traps on MPE XL are not guaranteed to be identical to those on MPE V/E.
- NM supports two floating point formats: IEEE and 3000 Mode. Both execute in NM, but 3000 Mode performs HP 3000-type manipulations. Since it is possible to use both formats during program execution, there are separate bits in the mask for enabling/disabling traps of these formats.
- By default, all arithmetic traps except IEEE floating point exceptions are enabled, and the system trap handler is armed. Many floating point operations result in an inexact result. Consequently, most compiler libraries doing floating point operations will result in an inexact trap if the IEEE Inexact Result trap is enabled. Therefore, you should enable the IEEE Inexact Result trap only if absolutely necessary.
- Some of the new trap conditions are not strictly arithmetic traps (for example, range errors, NIL pointers, and paragraph stack overflow). However, they and many arithmetic traps

are caught by reserved instructions that raise the conditional traps. For this reason, they are treated as arithmetic traps.

- Some of the instructions that raise conditional traps are reserved to indicate some of the arithmetic trap conditions. A nonreserved instruction is one not generated by a compiler. If a nonreserved instruction causes a conditional trap, this is reported as an Unimplemented Condition Trap.

Arithmetic Traps Not Available in Native Mode

The following arithmetic trap conditions are *not* handled by the NM arithmetic trap-handling intrinsics, `HPENBLTRAP`, `ARITRAP`, and `XARITRAP`:

- Invalid Source Word Count
- Invalid Decimal Operand Length

These trap conditions continue to be handled by the CM versions of the arithmetic trap handling intrinsics.

New Native Mode Arithmetic Traps

The new arithmetic trap conditions listed below are available to applications executing in the NM programming environment of MPE XL.

- IEEE floating point divide by zero.
- IEEE floating point, inexact result.
- IEEE floating point underflow.
- IEEE floating point overflow.
- IEEE floating point, invalid operation.
- range errors.
- software-detected NIL pointer reference.
- result of software-detected pointer arithmetic misaligned, or error in conversion from long pointer to short pointer.
- unimplemented condition traps.

- paragraph stack overflow.
- 3000 mode packed decimal error.

Using Compiler Directives to Disable Traps

Another way to disable arithmetic traps on MPE XL is to use NM compiler directives, for example `$ovflcheck off$` in HP Pascal/XL. When compiler directives are used, the compiler generates arithmetic instructions that do not trap on overflow. When compiler directives are not used, the trap actually takes place, but the MPE XL trap subsystem recovers from the trap and takes the action required to continue execution.

Arming Traps in a Mixed Mode Programming Environment (new)

One major effect of the architectural differences between MPE V/E-based and MPE XL-based systems concerns the nature of recovery actions in the two MPE XL programming environments, CM and NM. Results of trapping operations on the two systems are different when you recover and continue. CM trap handlers cannot catch NM traps and, likewise, NM trap handlers cannot catch CM traps. Consequently, if your user-written code executes in both NM and CM, then in order to catch all arithmetic, library, and system traps, you must have both a CM and an NM trap handler and use the Switch subsystem to arm/enable other-mode trap handlers.

The exception to the situation described above is the CONTROL-Y (subsystem break) trap. There is only one subsystem break handler at a time for a process, be it CM or NM. If a CM CONTROL-Y trap handler is armed with `XCONTRAP` and you hit CTRL Y while the program is executing in NM, the operating system automatically switches on your behalf to invoke the trap handling procedure. The converse is also true.

Refer to the *Switch Programming Guide* (32650-90014) for more information about the Switch subsystem and mixed mode programming.

HPENBLTRAP (NM) (new)

The MPE XL NM intrinsic `HPENBLTRAP` corresponds to the MPE V/E and CM `ARITRAP` intrinsic. Both selectively enable or disable arithmetic traps, but `HPENBLTRAP` provides you with more flexibility.

Arming and Enabling Traps

There is a difference between arming and enabling traps. Enabling a trap means that the occurrence of a trap condition is not ignored. Arming a trap is required so that, on a trap condition, a user-written routine is invoked and can take appropriate recovery actions.

The following list summarizes what can occur when an arithmetic trap condition arises:

- If a trap is both enabled and armed, the user-written trap handler is invoked whenever a trap condition occurs.
- If a trap is enabled but not armed, one of two situations applies:
  - If you have executed an HP Pascal/XL TRY statement, control is passed to the RECOVER block by doing an ESCAPE.
  - If you have not executed a HP Pascal XL TRY statement, an error message is output and the process aborts.
- If a trap is disabled, irrespective of whether it is armed or not, the trap is ignored, and execution of the process continues uninterrupted.

Real Number and Arithmetic Traps

NM supports two floating point formats: IEEE and HP3000 Mode. Both execute in NM, but 3000 Mode performs HP 3000 type manipulations. Since it is possible to use both formats during program execution, there are separate bits in the mask for enabling/disabling traps of these formats.

Note By default, all traps except IEEE floating point exceptions are enabled, and the system trap handler is armed. Many floating point operations result in an inexact result. Consequently, most compiler libraries doing floating-point operations will result

in an inexact trap if the IEEE inexact result trap is enabled. Therefore, you should enable the IEEE inexact result trap [bit (17:1)] only if absolutely necessary.

HPENBLTRAP Parameters

Two HPENBLTRAP parameters are required: *oldmask* and *mask*.

The *oldmask* parameter accepts a 32-bit signed integer passed by reference and returns the value of the previous *mask* to your program.

The *mask* parameter, a 32-bit signed integer passed by value, indicates the current mask. You indicate, bit by bit, which traps you want enabled or disabled according to the correspondence code listed in the *MPE XL Intrinsics Reference Manual* (32650-90028).

For example, if bit 30 is set to 0, the Integer Divide by Zero trap is off; and if bit 17 is set to 1, the IEEE floating point inexact result trap is on.

Some of the specified error conditions are not strictly arithmetic traps (for example, range errors, NIL pointers, and paragraph stack overflow). However, they and many arithmetic traps are caught by reserved instructions that raise the conditional traps. For this reason, all are enabled/disabled by `HPENBLTRAP`.

Some of the instructions that raise conditional traps are reserved to indicate some of the above trap conditions. A nonreserved instruction is one not generated by a compiler. If a nonreserved instruction causes a conditional trap, this is reported as an Unimplemented Condition Trap.

HPENBLTRAP Condition Codes

The following Condition Codes apply to `HPENBLTRAP`:

CCE	Request granted. All traps were originally disabled.
CCG	Request granted. At least one trap was originally enabled.
CCL	Not returned by this intrinsic.

CM and NM Trap Handler Differences (changed)

A user-created CM trap handler differs from a NM trap handler in its calling sequence and the method by which the trap handler obtains error information:

- In CM, typically, error information is obtained by the trap handler from the process stack.
- In NM, a pointer to a record containing error information is passed to the trap handler.

ARITRAP (NM) (changed)

The *trapstate* parameter of the NM version of the `ARITRAP` intrinsic is a required 32-bit signed integer passed by value. The parameter size has been expanded to handle the additional arithmetic traps available to NM applications. The *trapstate* parameter represents a value enabling or disabling arithmetic traps. (An exception is the IEEE Inexact Result trap, which can only be disabled.)

The CM version of `ARITRAP` performs as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007). The NM version performs as described in the *MPE XL Intrinsics Reference Manual* (32650-90028)

XARITRAP (NM) (changed)

The following are the parameters that have been modified in the NM version of the `XARITRAP` intrinsic. The parameter sizes have been expanded to handle the additional arithmetic traps available to NM applications.

The *mask* and *oldmask* parameters of the NM version `XARITRAP` must be 32-bit signed integer values. The *mask* and *oldmask* parameters represent values determining which enabled trap conditions invoke the user-written software trap handler specified in the *plabel* or *oldplabel* parameters.

These parameters represent arithmetic traps identically to the *mask* and *oldmask* parameters of the `HPENBLTRAP` intrinsic.

In MPE V/E and in MPE XL CM, plabels must be 16-bit entities. In NM, memory addresses represented as plabels must be 32-bit entities. In NM, the *plabel* and *oldplabel* parameters of `XARITRAP` must be 32-bit signed integer values passed by reference.

The CM version of `XARITRAP` performs as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007). The NM version performs as described in the *MPE XL Intrinsics Reference Manual* (32650-90028)

Note CM arithmetic trap handlers cannot catch traps encountered in NM code and, likewise, NM arithmetic trap handlers cannot catch traps encountered in CM code. For details on arming traps when your application contains both CM and NM code, refer to "Arming Traps in a Mixed Mode Programming Environment (new)" earlier in this chapter.

XCONTRAP (NM) (changed)

The following are the parameters that have been modified in the NM version of the `XCONTRAP` intrinsic.

The *plabel* and *oldplabel* parameters of the NM version of `XCONTRAP` must be 32-bit signed integer values passed by reference. In NM, memory addresses represented as plabels must be 32-bit entities (in MPE V/E and in MPE XL CM, plabels must be 16-bit entities).

The CM version of `XCONTRAP` performs as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007). The NM version performs as described in the *MPE XL Intrinsics Reference Manual* (32650-90028).

XLIBTRAP (NM) (changed)

The following are the parameters that have been modified in the NM version of the `XLIBTRAP` intrinsic.

The *plabel* and *oldplabel* parameters of the NM version of `XLIBTRAP` must be 32-bit signed integer values passed by reference. In NM, memory addresses represented as plabels must be 32-bit entities (in MPE V/E and in MPE XL CM, plabels must be 16-bit entities).

The CM version of `XLIBTRAP` performs as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007). The NM version performs as described in the *MPE XL Intrinsics Reference Manual* (32650-90028).

Note CM arithmetic trap handlers cannot catch traps encountered in NM code and, likewise, NM arithmetic trap handlers cannot catch traps encountered in CM code. For details on arming traps when your application contains both CM and NM code, refer to "Arming Traps in a Mixed Mode Programming Environment (new)" earlier in this chapter.

XSYSTRAP (NM) (changed)

The following are the parameters that have been modified in the NM version of the `XSYSTRAP` intrinsic.

The *plabel* and *oldplabel* parameters of the NM version of `XSYSTRAP` must be 32-bit signed integer values passed by reference. In NM, memory addresses represented as plabels must be 32-bit entities (in MPE V/E and in MPE XL CM, plabels must be 16-bit entities).

The CM version of `XSYSTRAP` performs as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007). The NM version performs as described in the *MPE XL Intrinsics Reference Manual* (32650-90028).

Note CM arithmetic trap handlers cannot catch traps encountered in NM code and, likewise, NM arithmetic trap handlers cannot catch traps encountered in CM code. For details on arming traps when your application contains both CM and NM code, refer to "Arming Traps in a Mixed Mode Programming Environment (new)" earlier in this chapter.

Additional Information

Refer to the *MPE V/E Intrinsics Reference Manual* (32033-90007), the *MPE XL Intrinsics Reference Manual* (32650-90028) and the *Trap Handling Programmer's Guide* (32650-90026) for details on the NM intrinsics and features mentioned in this chapter.

13

Debugging Applications

This chapter briefly presents the new MPE XL System Debugger and related intrinsics. This is an introduction, and not thorough enough to prepare you to use the debugger. Using the debugger, both in CM and in NM, is significantly different than using the MPE V/E System Debugger. Before you try to use it, you should become familiar with the MPE XL manual, *System Debug Reference Manual* (32650-90013).

Overview of Differences: MPE V/E and MPE XL

MPE XL System Debug provides both privileged and non-privileged users with an interactive debugging facility invoked through an integrated set of commands, NM intrinsics, and CM intrinsics.

Compared to MPE V/E Debug, the MPE XL Debugger has been extensively modified and expanded. Even in CM, MPE XL Debug is *not* the same as MPE V/E Debug. Many commands and intrinsics have been extended and modified, CM as well as NM. A number of new features have been added—for instance, window displays can be used for both CM and NM debugging. The prompts you see on the screen, and the responses you make to them, have been changed.

If you are using MPE XL Debug with a CM program, *do not* follow the instructions in the MPE V/E Debug documentation. You should become familiar with *System Debug Reference Manual* (32650-90013) before trying to use the debugger in either mode.

New

MPE XL features that are not available on MPE V/E include:

- MPE XL System Debug
- The following intrinsics:

```
HPDEBUG (NM)
HPRESETDUMP (NM)
HPSETDUMP (NM)
```

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- The following intrinsics:

```
SETDUMP
STACKDUMP
```

- The following commands:

```
DEBUG
ERRDUMP
SETDUMP
```

Not Used

Existing MPE V/E features that are not supported on MPE XL include:

- The MPE V/E System Debugger
- The following intrinsic:

```
STACKDUMP'
```

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

```
DEBUG
RESETDUMP
```

- The following command:

```
RESETDUMP
```

MPE XL System Debug (new)

MPE XL Debug's functions have been greatly expanded by comparison with MPE V/E Debug. Many new features have been added. MPE XL Debug allows you to:

- Calculate the value of expressions to determine the correct values for variables at a given point in the program. Values can be custom formatted in several bases.
- Use new full screen displays (windows) to inspect registers, program code, the current stack frame, and the top of stack. You can dynamically monitor changing values by aiming groups of custom user windows at important data blocks and watching the changes as they occur.
- Create and reference user-defined variables.
- Define powerful parameterized macros. Macros can be invoked as new commands to perform useful sequences of commands, or as functions within expressions that return single values.
- Define aliases for command and macro names.
- Display online help for all commands, predefined functions, and environment variables.

- Execute commands from a file, record all user input to a logfile, and record all debug output to a listfile.
- 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 to be automatically executed when the breakpoint is hit.
- Single step 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, and Hewlett-Packard Precision Architecture virtual or real memory addresses.
- Display a symbolic procedure stack trace, optionally displaying interleaved NM and CM calls. It is also possible to temporarily set the current debug environment back to the environment that existed at any marker on the stack.

HPDEBUG (NM) (new)

The `HPDEBUG` intrinsic is the counterpart to the MPE V/E `DEBUG` intrinsic. It calls the MPE XL system debugger. You can use an optional parameter to pass debug commands you want to be automatically executed when the debugger is entered.

When the `HPDEBUG` calls the debugger, the debugger pushes your commands onto its command stack and executes them. You pass a character array of up to 1024 characters in the *cmdstr* parameter. The first character in the buffer is recognized as the delimiter. The last character in the command string must be immediately followed by that same delimiter.

For processes in jobs, process execution is resumed when the command string is exhausted. Processes run from a job will not be allowed to stop in the system debugger. If the command string does cause control to return to the calling procedure, any remaining commands are left pending on the debugger command stack to be executed the next time the debugger is called.

For processes in sessions, control remains in the debugger unless you pass a `CONTINUE` command. If no command in the command string causes control to be returned to the calling procedure, the user will be left in the debugger as long as the process is being run from a session environment.

You may have output sent to any writable ASCII file you specify. By default, output is sent to the terminal `LDEV` for sessions and `$STDLIST` for jobs.

The *status* parameter, a 32-bit signed integer passed by reference is optional, but recommended. It indicates status in two fields. One reports the presence of errors and warnings, and the other indicates their source. If an error or warning condition is encountered, and you did not specify the *status* parameter, `HPDEBUG` causes the calling process to abort. Using the *status* parameter in MPE XL is analogous to using condition codes in MPE V/E. You can translate returned values with the *MPE XL Intrinsics Reference Manual* (32650-90028).

HPRESETDUMP (NM) (new)

The `HPRESETDUMP` intrinsic disarms the system debugger call from a process abort. Only the current process is affected.

The *status* parameter, a 32-bit signed integer passed by reference is optional, but recommended. It indicates status in two fields. One reports the presence of errors and warnings, and the other indicates their source. If an error or warning condition is encountered, and you did not specify the *status* parameter, `HPDEBUG` causes the calling process to abort. Using the *status* parameter in MPE XL is analogous to using condition codes in MPE V/E. You can translate returned values with the *MPE XL Intrinsics Reference Manual* (32650-90028).

HPSETDUMP (NM) (new)

The HPSETDUMP intrinsic arms the system debugger call from a process abort. If a process aborts, HPSETDUMP enables automatic execution of a set of system debugger commands. The process can be the current process or any child of the current process created after the intrinsic call. That is, the intrinsic affects all new child processes and all generations thereafter.

Before a process aborts, the debugger is called to execute the commands you entered in the *cmdstr* parameter. The commands are contained in a character array of up to 255 characters, left-justified, with a delimiting character as the first and last character of the command string. Commands that attempts to obtain user input cause an error when executed by the debugger.

If the process that aborts is being run from a job, the process will terminate after executing the command string.

If the process is being run from a session the specified command string is executed first. Next, the debugger stops to accept interactive commands with I/O performed at the user terminal if the following conditions are met:

- The abort did not occur while in system code.
- The process entered the abort code via a native mode interrupt (typically caused by arithmetic and code-related traps).

Once the debugger accepts interactive input, the user is free to enter any DEBUG command. The user may choose to resume the process or have it terminate with the CONTINUE command.

If the cause of the abort is a stack overflow, a stack trace is printed and the process is terminated immediately thereafter. The command string is not executed.

The *status* parameter, a 32-bit signed integer passed by reference is optional, but recommended. It indicates status in two fields. One reports the presence of errors and warnings, and the other indicates the subsystem where they occurred. If an error or warning condition is encountered, and you did not specify the *status* parameter, HPSETDUMP causes the calling process to abort. Using the *status* parameter in MPE XL is analogous to using condition codes in MPE V/E.

SETDUMP, STACKDUMP (changed)

The SETDUMP and STACKDUMP intrinsics are available in a modified form in both CM and NM. The optional parameters listed below are ignored by MPE XL when they are specified in the indicated intrinsic call:

flags—SETDUMP and STACKDUMP

idnumber—STACKDUMP

selec—STACKDUMP

Note that the *idnumber* parameter, while ignored as input, is still used, when appropriate, to return a file system error number.

Note While SETDUMP and STACKDUMP are available in NM, enhanced functionality is provided to NM programs through the HPDEBUG, HPRESETDUMP, and HPSETDUMP intrinsics.

Note that there is no HPSTACKDUMP intrinsic. The user can produce a custom stackdump by using the intrinsic and entering *"TRACE"* as one of the command parameters.

The secondary entrypoint STACKDUMP', used to send the stack trace to an already-opened file, is not available in the NM version of STACKDUMP (it remains available in CM only). An equivalent feature is available in NM through the HPSETDUMP intrinsic.

DEBUG Command (changed)

The `DEBUG` command enters the system debugger directly from the session CI.

For MPE XL, there is a new optional parameter, *commands*, that defines a string of system debugger commands to be executed at the time of invocation. This string may be up to 255 characters long.

The command string can control return to the CI. If the command string contains a command to return to the CI, any further commands in the string are not executed. Subsequent commands are left pending on the debugger's command stack. If the string does not return to the CI, the user is left in the debugger.

For descriptions of the commands that can be used with the debugger, refer to the *System Debug Reference Manual* (32650-90013). Most debugger commands are valid in CM as well as NM (see the command descriptions for this information).

ERRDUMP Command (changed)

This command dumps the process or system error stack for a specified number of entries. The error stack normally contains error numbers corresponding to non-zero values of HPE_STATUS; these are the errors that the operating system encountered while processing the last request.

The error numbers in the error stack are translated and the corresponding text displayed (from SYSCAT.PUB.SYS) on the `STDLIST` device. This information is useful in application or system debugging.

SETDUMP Command (changed)

This command arms the system debugger call for a process abort. For MPE XL, a new optional parameter, `DEBUG="`*commands*`"`, defines a string of optional commands that will be executed when the process aborts. These are the same debugger commands that can be used with the `DEBUG` command (refer to the specific command descriptions for details).

Some parameters of the MPE V/E `SETDUMP` command are retained for compatibility reasons. They are ignored, however, by MPE XL. These parameters are:

- DB = Dump DL to Qinitial
- ST = Dump Qinitial to S

STACKDUMP' (NM) (not used)

The secondary entrypoint `STACKDUMP'`, used to send the stack trace to an already-opened file, is not available in the NM version of `STACKDUMP` (it remains available in CM only). An equivalent feature is available in NM through the `HPSETDUMP` intrinsic.

MPE V/E System Debugger

The MPE V/E System Debugger is not supported on MPE XL in CM or NM. Using the MPE XL System Debugger is *not* the same as using the MPE V/E debugger.

Additional Information

For more information, refer to the *System Debug Reference Manual* (32650-90013). The MPE V/E programmer should consult this manual before attempting to use the system debugger in CM or in NM, as the commands and instructions are *not* the same as the MPE V/E debugger.

14

Using Extra Data Segments

Introduction

This chapter describes the differences between MPE XL and MPE V/E in the implementation of the Data Segment Management (DS) capability and the use of extra data segments. Split Stack Mode execution in the native mode programming environment is also discussed.

Overview of MPE XL and MPE V/E Differences

Data segment management is identical in MPE V/E and MPE XL Compatibility Mode (CM). Data Segment Management has been modified in MPE XL Native Mode (NM).

Split Stack Mode Execution is identical in MPE V/E and MPE XL CM. Split Stack Mode execution has been eliminated from NM.

Changed

Existing MPE V/E features that have been modified on MPE XL include:

- Extra Data Segments (NM)

Not Used

Existing MPE V/E features that are not supported on MPE XL include:

- Split Stack Mode Execution (NM)
- `SWITCHDB` intrinsic (NM)

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

ALTDSEG	DMOVIN
DMOVOUT	FREEDSEG
GETDSEG	SWITCHDB (CM)

Extra Data Segments (NM) (new)

The demand-paged virtual memory scheme and the greatly expanded addressing range of the 900 Series HP 3000 computer systems have eliminated the need for extra data segments.

In NM, much larger data areas can be declared in your program than are possible in MPE V/E or in CM. Also, your program can dynamically allocate more data areas than is possible in MPE V/E or in CM. For example, use of mapped files (discussed in Chapter 4, Accessing Files), or use of the NEW procedure in Pascal XL, greatly expands the data area that the NM program can directly access.

Data segment management (DS) intrinsics are not recommended for use in the NM programming environment. These intrinsics are provided in NM so that you can recompile an MPE V/E program using NM compilers without making source code changes. Use of DS intrinsics in NM will degrade the performance of your NM program.

Data Segment Management intrinsics continue to function in the CM environment as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007).

MPE XL Alternatives

The following are examples of situations where extra data segments are used in MPE V/E and suggested alternative approaches available in MPE XL.

Auxiliary Storage for Large Arrays

In MPE V/E, extra data segments are used to hold large arrays or tables that do not fit on the MPE V/E data stack, because of the limited size (only 64K bytes) of the stack.

In MPE XL, the expanded addressing available allows NM programs to place these arrays or tables in the data area simply by declaring them as program variables.

Interprocess Communication

In MPE V/E, extra data segments are used to allow messages to be passed between processes running in the same job/session.

In MPE XL, there are several alternative implementations available. The following, JCWs or CI variables and message files, are the two most useful.

Job Control Words and/or CI Variables. Short messages can be passed between processes in the same job/session using Job Control Words (JCWs) and/or Command Interpreter variables. JCWs are somewhat changed from MPE V/E and CI variables are a new feature on MPE XL. Values are written and read using the following intrinsics:

Table 14-1. Interprocess Communication Intrinsics

	Write	Read
CI Variables	`HPCIPUTVAR`	`HPCIGETVAR`
JCWs	`PUTJCW`	`GETJCW`

Refer to the discussion on using JCWs in *Interprocess Communication Programmer's Guide* (32650-90019). Refer to the discussion on the use of variables and JCWs programmatically in *Command Interpreter Access and Variables Programmer's Guide* (32650-90011).

Message Files. Message files are a special type of file with features that make them ideal for managing messages between processes. In addition, message files allow processes in different jobs/sessions to communicate with one another.

Refer to the discussion of using message files in *Interprocess Communication Programmer's Guide* (32650-90019).

Data Sharing Between Processes

In MPE V/E, extra data segments are sometimes used when two processes are sharing data (rather than passing it to one another). For example, a process builds a table of tax information that another process uses to calculate tax withholding.

In MPE XL, shared files are strongly recommended for such a situation. Refer to the discussion of sharing files in *Accessing Files Programmer's Guide* (32650-90017).

Startup Information for a Child Process

In MPE V/E an extra data segment is sometimes used when you need to pass information from a parent process to a child process at process creation time.

In MPE XL, you can pass information from a parent process to a new child process using optional parameters found in the `:RUN` command and in the `CREATEPROCESS` and `CREATE` intrinsics. This information can be obtained by the new process with the `GETINFO` intrinsic.

Refer to the discussion of passing information to a process in *Process Management Programmer's Guide* (32650-90023).

Split Stack Mode Execution (NM) (not used)

The demand-paged virtual memory scheme and the greatly expanded addressing range of the 900 Series HP 3000 computer systems have eliminated the need for Split Stack Mode Execution. In the MPE XL NM programming environment, you can declare in your program much larger data areas than are possible in MPE V/E or in the MPE XL CM programming environment. Also, your program can dynamically allocate more data areas than is possible in MPE V/E or in CM. For example, use of mapped files or use of the `NEW` procedure in Pascal XL greatly expands the data area that the NM program can directly access.

To ensure compatibility between MPE V/E and CM, Split Stack Mode Execution continues to function in the CM programming environment as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007).

SWITCHDB (NM) (not used)

The `SWITCHDB` intrinsic is not available to programs executing in NM.

To ensure compatibility between MPE V/E and MPE XL CM, `SWITCHDB` is available to programs executing in CM. The CM version of `SWITCHDB` is described in the *MPE V/E Intrinsics Reference Manual* (32033-90007).

Additional Information

Refer to the *MPE Segmenter* (30000-90011) *MPE V/E Intrinsics Reference Manual* (32033-90007) and the for details about Compatibility Mode intrinsics discussed in this chapter. Refer to the *MPE XL Intrinsics Reference Manual* (32650-90028) for details about the Native Mode intrinsics.

15

Changing Stack Sizes

Introduction

This chapter describes the differences between MPE XL and MPE V/E in the implementation of two stack management intrinsics, `DLSIZE` and `ZSIZE`.

Overview of MPE XL and MPE V/E Differences

When programming in the high level languages available in the Native Mode (NM) environment, all expansions and contractions of the NM stack are accomplished by the operating system. The NM versions of the `DLSIZE` and `ZSIZE` intrinsics do not affect the NM stack. Instead, they are implemented to enable your NM program to affect the Compatibility Mode (CM) stack.

The changes occur mostly because of Hewlett-Packard Precision Architecture (HP-PA) on the 900 Series HP 3000. Instead of segmentation, the MPE XL operating system has demand-paged virtual memory. The commands and intrinsics relating to segmentation are maintained for backward compatibility with MPE V/E, however.

Changed

Existing MPE V/E features that have been modified on MPE XL include:

MPE V/E features that are changed on MPE XL:

- Compatibility Mode Stack Allocation
- The following intrinsics:

`DLSIZE` (NM)
`ZSIZE` (NM)

Unchanged

Existing MPE V/E features that are the same on MPE XL include:

- The following intrinsics:

  `DLSIZE` (CM)
  `ZSIZE` (CM)

Compatibility Mode Stack Allocation (new)

In CM, MPE XL emulates the MPE V/E stack, but the operation is somewhat different. Because of HP-PA, MPE XL has implemented a different stack allocation scheme in CM than currently exists with the MPE V/E stack. The difference is primarily how MPE XL implements the CM Z-register.

MPE V/E Stack Allocation

In MPE V/E, it is important to keep the user's stack area as small as possible to allow for more efficient memory management. This is accomplished through the use (by both the system and the user) of the Z-register (or `Z`) and the *MAXDATA* parameter found in the `CREATE` and `CREATEPROCESS` intrinsics, and in the `RUN` command.

The Z-register (or `Z`) is initialized at process creation time to a value based upon both the user's DB area size and the value passed in either the *STACK=* parameter of the `RUN` command or, the *stacksize* parameter of the `CREATE` and `CREATEPROCESS` intrinsics.

During the life of the process, whenever the S-register (or `S`) exceeds `Z`, a stack overflow occurs, and the system increases `Z` by increments of 512 16-bit words until `Z` once again exceeds `S`. These intermediate stack overflows are handled by the system until both `S` and `Z` exceed `MAXDATA`, at which time the process is

aborted. This mechanism allows the user and the system to maintain as small a stack segment as possible for the particular application.

MPE XL CM Stack Allocation

Because of the demand paged virtual memory scheme, MPE XL does not need to keep the CM stack as small as possible. As a result, there is no "real" Z-register on MPE XL. `Z` is a system-managed value used to emulate the MPE V/E Z-register. To maintain compatibility with MPE V/E, the initial value of the CM stack's `Z` is calculated in the same way that `Z` is calculated in MPE V/E.

The major difference between MPE XL CM and MPE V/E stack allocation is that in MPE XL CM, there are no intermediate stack overflows that cause `Z` to be incremented by the system. The only time a stack overflow is detected, is when `S` exceeds `MAXDATA`, at which time the process is aborted. This implementation allows for less overhead in the management of the CM stack.

However, in order to emulate the relationship between `S` and `Z` that exists on MPE V/E, the algorithm described below is used whenever an application pushes the Z-register on to the user stack:

1. MPE XL checks to see if `S` (itself an emulated register) is greater than `Z`.
2. If `S` is greater than `Z`, MPE XL creates a temporary copy of `Z` (`Z_PRIME`), and adds increments of 512 16-bit words to `Z_PRIME` until the value is greater than `S` (`Z_PRIME` is never allowed to exceed `MAXDATA`).
3. `Z_PRIME` is pushed on to the user stack (not `Z`).

From the user's perspective, `Z` always appears larger than `S` because it is `Z_PRIME` that is pushed on to the user's stack (in fact, `Z` remains unchanged).

In MPE XL, when `S` exceeds `Z`, the system does not increase `Z`. `Z` is modified only when:

- The `ZSIZE` intrinsic is called.
- The `DLSIZE` intrinsic is called.
- The PCBX area of the CM stack is expanded.

In the above three operations, Z is expanded in 512 16-bit word increments until Z exceeds the current value of the emulated S-register (although Z is never allowed to exceed MAXDATA).

Impact on Migrated Applications

In MPE V/E, it is common for an application to cause the Z-register to increase through several intermediate stack overflows. In many cases, the value of S is then decreased, leaving a large amount of space between S and Z available for expansion. Thus, when the application checks to see if there is room on the stack for a data structure, there is usually plenty of space. However, on MPE XL, the emulated Z-register is not increased in the same way it is on MPE V/E. The amount of space reported between S and Z is usually of a much smaller value.

If you want to emulate in CM those increases to the Z-register that occur in MPE V/E whenever a intermediate stack overflow is detected, you must call the ZSIZE intrinsic from your application to set the value of Z to the desired size.

DLSIZE (NM) (new)

The NM version of the DLSIZE intrinsic enables your NM program to cause the area between DL and DB in the CM stack to be expanded or contracted within the CM stack segment.

DLSIZE does not affect the NM stack because the demand paged virtual memory scheme and the greatly expanded range of the Hewlett-Packard Precision Architecture have eliminated the need for programmatic expansion and contraction of the NM stack. DLSIZE is provided in NM so that you can recompile an MPE V/E program using NM compilers without making source code changes.

The CM version of DLSIZE functions as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007).

ZSIZE (NM) (new)

The NM version of the `ZSIZE` intrinsic enables your NM program to alter the size of the current DB to Z area of the CM stack by adjusting the register offset of the Z address from the DB address (DB to Z).

`ZSIZE` does not affect the NM stack because the demand paged virtual memory scheme and the greatly expanded range of the HP Precision Architecture have eliminated the need for programmatic expansion and contraction of the Native Mode (NM) stack. The `ZSIZE` intrinsic is provided in NM so that you can recompile an MPE V/E program using NM compilers without making source code changes.

The CM version of `ZSIZE` functions as described in the *MPE V/E Intrinsics Reference Manual* (32033-90007).

Additional Information

Refer to the *MPE XL Intrinsics Reference Manual* (32650-90028) for details about Native Mode intrinsics discussed in this chapter.