IBM Parallel Environment for AIX

Operation and Use, Volume 1
Using the Parallel Operating Environment

Version 2 Release 4

SC28-1979-02

IBM Parallel Environment for AIX

Operation and Use, Volume 1
Using the Parallel Operating Environment

Version 2 Release 4

SC28-1979-02

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vii.

| **Third Edition (October 1998)**

| This edition applies to Version 2, Release 4 , Modification 0 of the IBM Parallel Environment for AIX (5765-543), and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. A form for your comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department 55JA, Mail Station P384

522 South Road

Poughkeepsie, NY 12601-5400

United States of America

FAX (United States and Canada: 1+914+432-9405 FAX (Other Countries) Your International Access Code)+1+914+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL

Internet e-mail: mhvrcf@vnet.ibm.com

World Wide Web: http://www.rs6000.ibm.com (select Parallel Computing)

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

500 Columbus Avenue

Thornwood, NY 10594

USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Mail Station P300

522 South Road

Poughkeepsie, NY 12601-5400

USA

Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

About This Book

This book describes the IBM Parallel Environment (PE) for AIX program product and its Parallel Operating Environment (POE). It shows how to use POE's facilities to compile, execute, and analyze parallel programs.

This book concentrates on the actual commands and how to use them, as opposed to the writing of parallel programs. For this reason, you should use this book in conjunction with *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*, (GC23-3894). New users should refer to *IBM Parallel Environment for AIX: Hitchhiker's Guide*, (GC23-3895), for basic and introductory information on PE.

| This book assumes that AIX Version 4.3.2 or later , X-Windows**, and the PE software are already installed. It also assumes that you have been authorized to run the Parallel Operating Environment (POE). The PE software is designed to run on an IBM RS/6000 SP, an RS/6000 network cluster, or on a mixed system where additional RS/6000 processors supplement an SP system. For complete information on installing the PE software and setting up users, see *IBM Parallel Environment for AIX: Installation*, (GC28-1981). Also, see the appropriate AIX 4.3.2 or later documentation listed in “Related Publications” on page xii.

Who Should Use This Book

This book is designed primarily for end users and application developers. It is also intended for those who run parallel programs, and some of the information covered should interest system administrators. Readers should have knowledge of the AIX operating system and the X-Window system. Where necessary, this book provides some background information relating to these areas. More commonly, this book refers you to the appropriate documentation.

How This Book is Organized

Overview of Contents

This book contains the following information:

- Chapter 1, “Introduction” on page 1 is a quick overview of the PE program product. It describes the various PE components, and how you might use each in developing a parallel application program.
- Chapter 2, “Executing Parallel Programs” on page 9 describes how to compile and execute parallel programs using the Parallel Operating Environment (POE).
- Chapter 3, “Managing POE Jobs” on page 55 includes information on allocating nodes with the Resource Manager, IBM LoadLeveler support, and the environment variables to use when running your applications.
- Chapter 4, “Monitoring Program Execution and System Activity” on page 75 describes the Program Marker Array which allows you to monitor program execution online, and the System Status Array which allows you to monitor system activity online.

- Appendix A, “Parallel Environment Commands” on page 83 contains the manual pages for the PE commands discussed throughout this book.
- Appendix B, “POE Environment Variables and Command-Line Flags” on page 139 describes the environment variables you can set to influence the execution of parallel programs and the operation of PE tools. This appendix also describes the command-line flags associated with each of the environment variables. When invoking a parallel program, you can use these flags to override the value of an environment variable.

Typographic Conventions

This book uses the following typographic conventions:

Type Style	Used For
bold	**Bold** words or characters represent system elements that you must use literally, such as command names, flag names, and path names. **Bold** words also indicate the first use of a term included in the glossary.
italic	*Italic* words or characters represent variable values that you must supply. *Italics* are also used for book titles and for general emphasis in text.
`Constant width`	Examples and information that the system displays appear in `constant width` typeface.

In addition to the highlighting conventions, this manual uses the following conventions when describing how to perform tasks. User actions appear in uppercase boldface type. For example, if the action is to enter the **pedb** command, this manual presents the instruction as:

ENTER **pedb**

The symbol “●” indicates the system response to an action. So the system's response to entering the **pedb** command would read:

● The **pedb** main window opens.

Related Publications

IBM Parallel Environment for AIX Publications

- *IBM Parallel Environment for AIX: General Information*, GC23-3906
- *IBM Parallel Environment for AIX: Hitchhiker's Guide*, GC23-3895
- *IBM Parallel Environment for AIX: Installation*, GC28-1981
- *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*, SC28-1980
- *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*, GC23-3894
- *IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference*, GC23-3893
- *IBM Parallel Environment for AIX: Messages*, GC28-1982

- *IBM Parallel Environment for AIX: Licensed Program Specifications*, GC23-3896

As an alternative to ordering the individual books, you can use SBOF-8588 to order the entire IBM Parallel Environment for AIX library.

Related IBM Publications

- *IBM AIX Version 4 Getting Started*, SC23-2527
- *IBM AIX General Concepts and Procedures for RS/6000* GC23-2202
- *IBM AIX Version 4 Files Reference*, SC23-2512
- *IBM AIX Version 4 System Management Guide: Communications and Networks*, SC23-2526
- *IBM AIX Version 4.1 Installation Guide* SC23-2550
- *IBM AIX Version 4.2 Installation Guide* SC23-1924
- *IBM AIX Version 4 Commands Reference*, SBOF-1851 (all volumes)
- *IBM AIX Versions 3.2 and 4 Performance Tuning Guide* SC23-2365
- *IBM AIX Version 4 Messages Guide and Reference* SC23-2641
- *IBM AIX Version 4.1 Network Installation Management Guide and Reference*, SC23-2627
- *IBM AIX Version 4.2 Network Installation Management Guide and Reference*, SC23-1926
- *IBM AIX Version 4 System Management Guide: Operating System and Devices*, SC23-2525
- *IBM AIX Version 4 General Programming Concepts: Writing and Debugging Programs*, SC23-2533
- *IBM AIX Version 4 Communications Programming Concepts* SC23-2610
- *Diskless Workstation Management Guide*, SC23-2433
- *C++ for AIX/6000: Language Reference*, SC09-1606
- *C++ for AIX/6000: Standard Class Library Reference*, SC09-1604
- *C++ for AIX/6000: User's Guide*, SC09-1605
- *IBM Performance Toolbox 1.2 and 2 for AIX: Guide and Reference*, SC23-2625

Related Non-IBM Publications

- Almasi, G., Gottlieb, A., *Highly Parallel Computing* Benjamin-Cummings Publishing Company, Inc., 1989.
- Gropp, W., Lusk, E., Skjellum, A., *Using MPI*, The MIT Press, 1994.
- Message Passing Interface Forum, MPI: *A Message-Passing Interface Standard* Version 1.1, University of Tennessee, Knoxville, Tennessee, June 6,
| 1995. This standard can be found at the following URL:
| **http://www.mpi_forum.org/**
- Foster, I., *Designing and Building Parallel Programs* Addison-Wesley, 1995.
- Pfister, Gregory, F., *In Search of Clusters* Prentice Hall, 1995.

Setting POE Environment Variables

Throughout this book, a number of POE environment variables are discussed. These environment variables can be set to influence the operation of certain tools and the execution of parallel programs. The POE environment variables also have associated command-line flags that can be used when invoking a parallel program or tool. The POE command-line flags temporarily override their associated environment variable. The following table shows how to set a POE environment variable or command-line flag depending on the shell you are using. The remainder of this book assumes use of the Korn Shell. For a complete listing of the POE environment variables and command-line flags, refer to Appendix B, “POE Environment Variables and Command-Line Flags” on page 139.

	To set an environment variable:	**To override an environment variable when invoking a parallel program or PE tool:**
In Korn Shell:	**ENTER** **export** *VARIABLE***=***value*	**ENTER** *command program -flag value*
In C Shell	**ENTER** **setenv** *VARIABLEvalue*	**ENTER** *command program -flag value*

National Language Support

For National Language Support (NLS), all PE components and tools display messages located in externalized message catalogs. English versions of the message catalogs are shipped with the IBM Parallel Environment for AIX program product, but your site may be using its own translated message catalogs. The AIX environment variable **NLSPATH** is used by the various PE components to find the appropriate message catalog. **NLSPATH** specifies a list of directories to search for message catalogs. The directories are searched, in the order listed, to locate the message catalog. In resolving the path to the message catalog, **NLSPATH** is affected by the values of the environment variables **LC_MESSAGES** and **LANG**. If you get an error saying that a message catalog is not found, and want the default message catalog:

ENTER **export NLSPATH=/usr/lib/nls/msg/%L/%N**

export LANG=C

The PE message catalogs are in English, and are located in the following directories:

/usr/lib/nls/msg/C
/usr/lib/nls/msg/En_us
/usr/lib/nls/msg/en_US

If your site is using its own translations of the message catalogs, consult your system administrator for the appropriate value of **NLSPATH** or **LANG**. For additional information on NLS and message catalogs, see *IBM Parallel Environment for AIX: Messages* and *AIX for RS/6000: General Programming Concepts*

Accessing Online Information

In order to use the PE man pages or access the PE online (HTML) publications, the **ppe.pedocs** file set must first be installed. To view the PE online publications, you also need access to an HTML document browser such as Netscape. An index to the HTML files that are provided with the **ppe.pedocs** file set is installed in the **/usr/lpp/ppe.pedocs/html** directory.

Online Information Resources

If you have a question about the SP, PSSP, or a related product, the following online information resources make it easy to find the information:

- Access the new SP Resource Center by issuing the command: **/usr/lpp/ssp/bin/resource_center**. Note that the **ssp.resctr** fileset must be installed before you can do this.

  If you have the Resource Center on CD ROM, see the readme.txt file for information on how to run it.

- Access the RS/6000 Web Site at: **http://www.rs6000.ibm.com**.

Getting the Books Online

All of the PE books are available in Portable Document Format (PDF). They are included on the product media (tape or CD ROM), and are part of the **ppe.pedocs** file set. If you have a question about the location of the PE softcopy books, see your System Administrator.

To view the PE PDF publications, you need access to the Adobe Acrobat Reader 3.0.1. The Acrobat Reader is shipped with the AIX Version 4.3 Bonus Pack and is also freely available for downloading from the Adobe web site at URL **http://www.adobe.com**.

As stated above, you can also view or download the PE books from the IBM RS/6000 web site at **http://www.rs6000.ibm.com**. At the time this manual was published, the full path was **http://www.rs6000.ibm.com/resource/aix_resource/sp_books.** However, note that the structure of the RS/6000 web site can change over time.

Chapter 1. Introduction

The IBM Parallel Environment for AIX program product (PE) is an environment designed for the development and execution of parallel Fortran, C, or C++ programs. PE consists of components and tools for developing, executing, debugging, profiling, and tuning parallel programs.

The PE is a distributed memory message passing system. It runs on the RS/6000 platform using the AIX operating system (Version 4.2.1). Specifically, you can use the PE to execute parallel programs on:

- any configuration of the IBM RS/6000 SP as described in the *IBM Parallel System Support Programs for AIX: Installation and Migration Guide* Essentially, an SP system is a collection of RS/6000 processors grouped into a number of *frames*. Each frame of an SP system can contain from two to 16 RS/6000 processors.
- a networked cluster of RS/6000 processors, including a single processor or a single workstation.
- a *mixed system*. In a mixed system, additional RS/6000 processors supplement the processors of an SP system.

The RS/6000 processors of your system are called *processor nodes*. A parallel program executes as a number of individual, but related, *parallel tasks* on a number of your system's processor nodes. The group of parallel tasks is called a *partition*. The processor nodes are connected on the same LAN, so the parallel tasks of your partition can communicate to exchange data or synchronize execution. If you are using an SP system:

- Your system may have an optional high performance switch for communication. The switch increases the speed of communication between nodes. It supports a high volume of message passing with increased bandwidth and low latency.
- Your system administrator can divide its nodes into separate pools. An SP system pool is a subset of processor nodes and is given an identifying pool
| number. A LoadLeveler system pool is a subset of processor nodes and is
| given an identifying pool name or number.

PE supports the two basic parallel programming models – SPMD and MPMD. In the *SPMD (Single Program Multiple Data) model*, the programs running the parallel tasks of your partition are identical. The tasks, however, work on different sets of data. In the *MPMD (Multiple Program Multiple Data) model*, each node may be running a different program. A typical example of this is the master/worker MPMD program. In a master/worker program, one task – the master – coordinates the execution of all the others – the workers.

Note: While the remainder of this introduction describes each of the PE components and tools in relation to a specific phase of an application's life cycle, this does not imply that they are limited to one phase. They are ordered this way for descriptive purposes only; you will find many of the tools useful across an application's entire life cycle.

The application developer begins by creating a parallel program's source code. The application developer might create this program from scratch or could modify an existing serial program. In either case, the developer places calls to **Message**

Passing Interface (MPI) or **Low-level Application Programming Interface (LAPI)** routines so that it can run as a number of parallel tasks. This is known as *parallelizing* the application. The MPI is similar to the Message Passing Library (MPL) from an earlier version of Parallel Environment. MPI provides message passing capabilities for the current version of PE. There are two libraries for MPI:

- Signal handling - which uses UNIX signals and signal handlers
- Threaded - which uses and supports POSIX user threads.

All tasks of a program must use either signal handling or threaded calls but not a combination of each.

MPL programs are still supported for non-threaded applications.

Note: Throughout this book, when referring to anything non-specific for MPI and MPL, the term *message passing* will be used. For example:

```
message passing program

message passing routine

message passing call
```

The message passing calls enable the parallel tasks of your partition to communicate data and coordinate their execution. The message passing routines in turn call communication subsystem library routines which handle communication among the processor nodes. There are two separate implementations of the communication subsystem library – the Internet Protocol (IP) Communication Subsystem and the User Space (US) Communication Subsystem. While the message passing application interface remains the same, the communication subsystem libraries use different protocols for communication among processor nodes. The IP communication subsystem uses Internet Protocol, while the US communication subsystem is designed for the SP system's high performance switch feature. The communication subsystem library implementations are dynamically linked when you invoke the program. For more information on the message passing subroutine calls, refer to *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference*, and *IBM Parallel Environment for AIX: Hitchhiker's Guide*

In addition to message passing communication, the Parallel Environment supports a separate communication protocol known as the **Low-level Application Programming Interface (LAPI)**. LAPI differs from MPI in that it is based on an “active message style” mechanism that provides a one-sided communications model. That is, one process initiates an operation and the completion of that operation does not require any other process to take a complimentary action.

LAPI only runs with the US Communication Subsystem. For this reason, it is designed to run on the SP system's high performance communication adapter only. The RS/6000 workstation cluster does not support LAPI.

Although LAPI is used for data communication in conjunction with PE, it is actually part of the communication subsystem for IBM's Parallel System Support Programs (PSSP). For more information on LAPI, see *IBM Parallel System Support Programs for AIX: Administration Guide*, and *IBM Parallel System Support Programs for AIX: Command and Technical Reference*

After writing the parallel program, the application developer then begins a cycle of modification and testing. The application developer now compiles and runs his program from his **home node** using the **Parallel Operating Environment (POE)**. The home node is any workstation on the LAN. POE is an execution environment designed to hide, or at least smooth, the differences between serial and parallel execution.

To assist with node allocation for job management, the role of IBM LoadLeveler has been expanded to work with POE for interactive jobs. LoadLeveler will now provide resource management function both on and off the SP system. You can run parallel programs on a cluster of processors running LoadLeveler, or on a mixed system of LoadLeveler processors that supplement an SP system. LoadLeveler not only provides SP node allocation for jobs using the US communication subsystem, but also provides management for non-SP nodes, or for SP nodes being used for jobs other than user space. LoadLeveler will still be used by POE for batch jobs as well. See the IBM LoadLeveler documentation for more information on this job management system.

In general, with POE, you invoke a parallel program from your home node and run its parallel tasks on a number of **remote nodes**. When you invoke a program on your home node, POE starts your **Partition Manager** which allocates the nodes of your partition and initializes the local environment. Depending on your hardware and configuration, the Partition Manager uses a **host list file**, LoadLeverler, or the **SP system Resource Manager** to allocate nodes. A host list file contains an explicit list of node requests, while LoadLeveler or the Resource Manager allocate nodes from one or more system pools implicitly based on their availability. On an SP system using the Resource Manager, you can also use a host list file to determine how an allocated node's resources – its SP switch adapter and CPU – are used. Your program task can either:

- share or not share the node's SP switch adapter
- share or not share the node's CPU.

With regard to the expanded LoadLeveler function, POE now provides an option to enable you to specify whether your program will use MPI, LAPI, or both. Using this option, POE ensures that each API initializes properly and informs LoadLeveler which APIs are used so each node is set up completely.

For Single Program Multiple Data (SPMD) applications the Partition Manager executes the same program on all nodes. For Multiple Program Multiple Data (MPMD) applications, the Partition Manager prompts you for the name of the program to load on each node. The Partition Manager also connects standard I/O to each remote node so the parallel tasks can communicate with the home node. Although you are running tasks on remote nodes, POE allows you to continue using the standard UNIX** and AIX execution techniques with which you are already familiar. For example, you can redirect input and output, pipe the output of programs, or use shell tools. The POE includes:

- A number of **parallel compiler scripts**. These are shell scripts that call the C, C++, or Fortran compilers while also linking in an interface library to enable communication between your home node and the parallel tasks running on the remote nodes. You dynamically link in a communication subsystem implementation when you invoke the executable.

- A number of **POE Environment Variables** you can use to set up your execution environment. These are AIX environment variables you can set to influence the operation of POE. These environment variables control such things as how processor nodes are allocated, what programming model you are using, and how standard I/O between the home node and the parallel tasks should be handled. Most of the POE environment variables also have associated command-line flags that enable you to temporarily override the environment variable value when invoking POE and your parallel program.

- Two X-Windows** analysis tools:
  - The **Program Marker Array**. This is a programmable array of small boxes, or *lights*, which are associated with parallel tasks. Under program control, these lights can change color to provide you with immediate visual feedback as your program executes. See Figure 1 on page 75 for a complete description of this tool.
  - The **System Status Array**. This tool lets you quickly survey the utilization of your processor nodes. It is useful when listing nodes in a host list file for explicit node allocation, and is discussed in "Using the System Status Array" on page 78.

The following tools are discussed in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference* and allow you to debug, visualize, and tune parallel programs.

There are two **parallel debugging facilities**. The first – **pdbx** – is a line-oriented debugger based on the **dbx** debugger. The other – **pedb** – is a Motif**-based debugger.

Once the parallel program is debugged, you now want to tune the program for optimal performance. To do this, you turn to the PE parallel profiling capability and Visualization Tool to analyze the program.

The **parallel profiling capability** enables you to use the PE Xprofiler graphical user interface, as well as the AIX commands **prof** and **gprof** on parallel programs. Xprofiler is a tool that helps you analyze your parallel application's performance quickly and easily. It uses procedure profiling information to construct a graphical display of the functions within your application. Xprofiler provides quick access to the profiled data, which lets you identify the functions that are the most CPU-intensive. The graphical user interface also lets you manipulate the display in order to focus on the application's critical areas.

The **Visualization Tool (VT)** contains a set of displays which allow you to visualize performance characteristics of your program and system. Each display presents specific, often complex, information in an easily-interpretable form such as a bar chart or a strip graph. You can use VT's displays for *trace visualization* and online *performance monitoring*.

- In *trace visualization*, you play back statistical and event records – or *trace records* – generated during your program's execution. You can use VT to visualize information about the program as well as its use of the underlying system. This visualized information can help you tune the program to optimize its use of the underlying system.
- In *performance monitoring*, you use VT as an online monitor to study the operational status and activity of each of the processor nodes in your SP system or RS/6000 network cluster. This mode of VT is similar to the System Status Array in that it only displays system statistics and not communication

information. Like the System Status Array, it is useful when listing nodes in a host list file for explicit node allocation.

Note: Once the parallel program is tuned to your satisfaction, you might prefer to execute it using a job management system such as IBM LoadLeveler*. If you do use a job management system, consult its documentation for information on its use.

What's New in PE 2.4?

AIX 4.3 Support

With PE 2.4, POE supports user programs developed with AIX 4.3. It also supports programs developed with AIX 4.2, intended for execution on AIX 4.3.

Parallel Checkpoint/Restart

This release of PE provides a mechanism for temporarily saving the state of a parallel program at a specific point (*checkpointing*), and then later **restarting** it from the saved state. When a program is checkpointed, the checkpointing function captures the state of the application as well as all data, and saves it in a file. When the program is restarted, the restart function retrieves the application information from the file it saved, and the program then starts running again from the place at which it was saved.

Enhanced Job Management Function

In earlier releases of PE, POE relied on the SP Resource Manager for performing job management functions. These functions included keeping track of which nodes were available or allocated and loading the switch tables for programs performing User Space communications. LoadLeveler, which had only been used for batch job submissions in the past, is now replacing the Resource Manager as the job management system for PE. One notable effect of this change is that LoadLeveler now allows you to run up to four User Space tasks per node.

MPI I/O

With PE 2.4, the MPI library now includes support for a subset of MPI I/O, described by Chapter 9 of the MPI-2 document; MPI-2: Extensions to the Message-Passing Interface, Version 2.0. MPI-I/O provides a common programming interface, improving the portability of code that involves parallel I/O.

1024 Task Support

With regard to MPI/LAPI jobs, this release of PE supports a maximum of 2048 tasks for IP, and 1024 tasks for US, as opposed to the previous release, which supported a maximum of 512 tasks.

Enhanced Compiler Support

In this release, POE is adding support for the following compilers:

- Fortan Version 5
- C
- C++

- xlhpf

Message Queue Facility

The **pedb** debugger now includes a message queue facility. Part of the **pedb** debugger interface, the message queue viewing feature can help you debug Message Passing Interface (MPI) applications by showing internal message request queue information. With this feature, you can view:

- A summary of the number of active messages for each task in the application. You can select criteria for the summary information based on message type and source, destination, and tag filters.
- Message queue information for a specific task.
- Detailed information about a specific message.

Xprofiler Enhancements

This release includes a variety of enhancements to Xprofiler, including:

- *Save Configuration* and *Load Configuration* options for saving the names of functions, currently in the display, and reloading them later in order to reconstruct the function call tree.
- An *Undo* option that lets you undo operations that involve adding or removing nodes or arcs from the function call tree.

PE 2.4 Migration Information

This section is intended for customers migrating from earlier releases of PE to PE 2.4. It contains specific information on some differences between earlier releases that you need to consider prior to installing or using PE 2.4. To find out which release of PE you currently have installed, use **lslpp**.

AIX Compatibility

PE 2.4 commands and applications are compatible with AIX Version 4.3.2 or later only, not with earlier versions of AIX.

Existing Applications

Applications from previous versions of Parallel Environment are binary compatible with PE 2.4, with the following exceptions:

- User applications created using PE 2.4 are not binary compatible with Version 1.
- In order to run under PE 2.4, you must recompile existing applications that were developed under PE Version 1.
- In order to run under PE 2.4, you must recompile any statically bound applications that were created with PE Version 2 Release 1.

Existing Host List Files

Host list files from previous releases that contained multiple pool or usage specifications will be affected as follows when using LoadLeveler:

- Usage specification in a host list file will be ignored.
- You can request how nodes are used with the **MP_CPU_USE** and/or **MP_ADAPTER_USE** environment variables, or their associated command line flags
- LoadLeveler does not allow dissimilar pool entries.

LAPI Applications

LAPI programs must set the **MP_MSGAPI** environment variable.

MPI and MPL Applications

- The MPI function became available in Version 2.
- The MPL message passing applications are source compatible between PE Version 1 Release 2 and PE 2.4, but must be recompiled.

Coexistence

All tasks within a partition or cluster must be running the same version of PE. You cannot mix versions of PE.

Therefore, for all processors within a workstation cluster, the same release level of the PE software is required.

When you use partitioning, you may have different levels of PE software installed on different partitions; however, *within a partition*, all the nodes must be at the same level of PE software.

Note: See *IBM Parallel Environment for AIX: Installation* for more information about software compatibility within a workstation cluster or partition, and for administrative and usage information about running different versions of POE in a partitioned environment.

Use of /usr/lib in LIBPATH

Users who previously set **LIBPATH** to include **/usr/lib** should no longer do so. Setting **LIBPATH** to include **/usr/lib** would cause the POE application not to include all of the POE libraries at execution time.

/usr/lib is included in the loader section search path of all POE applications at compile time, so there is no need to include it in **LIBPATH**.

Compiler -ip and -us options

The **-ip** and **-us** flags for PE Version 1 **mpcc**, **mpCC**, and **mpxlf** compiler scripts are no longer used or supported. All application programs are dynamically linked using these scripts.

Instructions are provided on how to create statically executable versions of your applications in *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*. User-written scripts that utilize these options need to be rewritten.

VT trace files are incompatible

VT trace files generated using Version 1 or Version 2, Release 2 will not be compatible with Version 2.4, and vice versa. Trace files must be regenerated.

However, refer to *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference* for information about the VT trace file format, if you want to write your own conversion program.

SP_Name Environment Variable

Previous versions of POE allowed jobs using the SP Resource Manager to be submitted from a non-SP node by setting the **SP_NAME** environment variable. For POE Version 2 Release 2 or later, you must also install the **ssp.clients** fileset. Refer to *IBM Parallel Environment for AIX: Installation* for more information.

POSIX Threads Support

PE 2.4 supports IEEE POSIX 1003.1-1996 of POSIX threads (sometimes known as Draft 10), that is xpg5 compliant as a default when compiling parallel applications. Existing applications from previous releases of PE were built with an earlier version of POSIX threads (Draft 7).

Existing threaded applications are supported in binary compatibility mode, without needing to recompile. However, these will run with the older objects from the previous version's threads library.

All new applications are compiled with the new draft of POSIX threads as the default. However, the POE threaded compiler scripts (**mpcc_r**, **mpCC_r**, **mpxlf_r**, **mpxlf90_r**) also provide an optional flag (**-d7**) to allow applications to be compiled with the older version of the threads library. See the appropriate compiler command description for further details.

32/64 Bit Applications

POE compiles and runs all applications as 32 bit applications. 64 bit applications are not yet supported.

Chapter 2. Executing Parallel Programs

This chapter describes the Parallel Operating Environment (POE). POE is a simple and friendly environment designed to ease the transition from serial to parallel application development and execution. POE lets you develop and run parallel programs using many of the same methods and mechanisms as you would for serial jobs. POE allows you to continue to use the standard UNIX** and AIX application development and execution techniques with which you are already familiar. For example, you can redirect input and output, pipe the output of programs into **more** or **grep**, write shell scripts to invoke parallel programs, and use shell tools such as history. You do all these in just the same way you would for serial programs. So while the concepts and approach to writing parallel programs must necessarily be different, POE makes your working environment as familiar as possible.

This chapter describes the steps involved in compiling and executing your parallel C, C++, or Fortran programs using either an IBM RS/6000 SP, an RS/6000 network cluster, or a mixed system.

Executing Parallel Programs Using POE

This section discusses how to compile and execute your parallel C, C++, or Fortran programs. It leaves out the first step in any application's life cycle which is actually writing the program. For information on writing parallel programs, refer to *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference IBM Parallel Environment for AIX: Hitchhiker's Guide*, and *IBM Parallel System Support Programs for AIX: Command and Technical Reference*.

Note: If you are using POE for the first time, check that you have authorized access. See *IBM Parallel Environment for AIX: Installation* for information on setting up users.

In order to execute an MPI, MPL, or LAPI parallel program, you need to:

1. Compile and link the program using shell scripts or makefiles which call the C, C++, or Fortran compilers while linking in the Partition Manager interface and message passing subroutines.
2. Copy your executable to the individual nodes in your partition if it is not accessible to the remote nodes.
3. Set up your execution environment. This includes setting the number of tasks, and determining the method of node allocation.
4. Optionally start either of the POE X-Windows analysis tools – the Program Marker Array and the System Status Array – you want to use.
5. Load and execute the parallel program on the processor nodes of your partition. You can:
   - load a copy of the same executable on all nodes of your partition. This is the normal procedure for SPMD programs.
   - individually load the nodes of your partition with separate executables. This is the normal procedure for MPMD programs.

- load and execute a series of SPMD or MPMD programs, in job step fashion, on all nodes of your partition.

Step 1: Compile the Program

As with a serial application, you must compile a parallel C, C++, or Fortran program before you can run it. Instead of using the **cc**, **xlC**, or **xlf** commands, however, you use the commands **mpcc**, **mpCC**, or **mpxlf**. The **mpcc**, **mpCC**, and **mpxlf** commands not only compile your program, but also link in the Partition Manager and message passing interface libraries. When you later invoke the program, the subroutines in these libraries enable the home node Partition Manager to communicate with the parallel tasks, and the tasks with each other. To compile threaded C, C++, or Fortran programs, use the **mpcc_r**, **mpCC_r**, or **mpxlf_r** commands. These commands can also be used to compile non-threaded programs with the threaded libraries.

| To compile programs with the checkpoint/restart capability, use the **mpcc_chkpt**,
| **mpCC_chkpt**, or **mpxlf_chkpt** commands. See *IBM Parallel Environment for AIX:
| Hitchhiker's Guide* for an overview of checkpointing and restarting POE programs.
| For specific details, see the section later in this chapter, “Checkpointing and
| Restarting Programs” on page 51.

These compiler commands are actually shell scripts which call the appropriate compiler. You can use any of the **cc**, **xlC**, or **xlf** flags on these commands.

The following table shows what to enter to compile a program depending on the language in which it is written. For more information on these commands, see Appendix A, “Parallel Environment Commands” on page 83.

To:	Enter:
Compile a C program. A communication subsystem library implementation will be dynamically linked when the executable is invoked.	**mpcc** *program.c* **-o** *program*
Compile a C++ program. A communication subsystem library implementation will be dynamically linked when the executable is invoked.	**mpCC** *program.C* **-o** *program*
Compile a Fortran program. A communication subsystem library implementation will be dynamically linked when the executable is invoked.	**mpxlf** *program.f* **-o** *program*
Compile a C program which uses threaded MPI. A communication subsystem library implementation will be dynamically linked when the executable is invoked.	**mpcc_r** *program.c* **-o** *program*
Compile a C++ program which uses threaded MPI. A communication subsystem library implementation will be dynamically linked when the executable is invoked.	**mpCC_r** *program.C* **-o** *program*
Compile a Fortran program which uses threaded MPI. A communication subsystem library implementation will be dynamically linked when the executable is invoked.	**mpxlf_r** *program.f* **-o** *program*

Notes:

1. Be sure to specify the **-g** flag when compiling a program for use with one of the parallel debuggers or with VT. The **-g** flag is a standard compiler flag that produces an object file with symbol table references. This object file is needed by the debuggers and by VT's Source Code view. For more information on the **-g** option, refer to its use on the **cc** command as described in *IBM AIX Version 4 Commands Reference*.
2. Application programs compiled for use with POE are limited to eight (8) data segments. The **-bmaxdata** option cannot specify any more than 0x80000000.

Creating a Static Executable

Note: We discourage you from creating statically bound executables with POE. If service is ever applied that affects any of the Parallel Environment libraries, you will need to recompile your application to create a new executable that will work with the new libraries. This could lead to a lot of work and may expose you to potential problems, which would be avoided if dynamic libraries are used.

In general, to create a static executable, do the following:

1. Create an object file of your program using **cc**, **xlf**, or **xlC**. For your threaded program, use **cc_r** or **xlC_r**. Include the path **/usr/lpp/ppe.poe/include** to the message passing include files in your compilation statement. For example:

   ```
   cc -c myprog.c -I/usr/lpp/ppe.poe/include
   ```

2. Using **ld**, create the executable.

The following table shows how you create a C, C++, or Fortran static executable for IP or US.

Note: When you see **ld**, **l** represents a lower case **L**. When you see **-bI**, **I** represents an upper case **i**.

To:	For IP, Enter:	For US (SP only), Enter:
Create a C or C++ static executable	**ld -o** *myprog myprog.o* **/lib/crt0.o -binitfini:poe_remote_main -bnso -lmpci -lmpi -lvtd -lc -lppe -bI :/lib/syscalls.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/ip**	**ld -o** *myprog myprog.o* **/lib/crt0.o -binitfini:poe_remote_main -bnso -bI:/usr/lpp/ssp/css/libus /fs_ext.exp -lmpci -lmpi -lvtd -lc -lppe -bI:/lib/syscalls.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/us**
Create a Fortran static executable	**ld -o** *myprog myprog.o* **/lib/crt0.o -binitfini:poe_remote_main -bnso -lmpci -lmpi -lvtd -lxlf90 -lxlf -lc -lppe -bI:/lib/syscalls.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/ip**	**ld -o** *myprog myprog.o* **/lib/crt0.o -binitfini:poe_remote_main -bnso -bI:/usr/lpp/ssp/css/libus /fs_ext.exp -lmpci -lmpi -lvtd -lxlf90 -lxlf -lc -lppe -bI:/lib/syscalls.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/us**
Create a C or C++ static executable which uses threaded MPI	**ld -o** *myprog myprog.o* **/lib/crt0_r.o -binitfini:poe_remote_main -bnso -lmpi_r -lvtd_r -lc_r -lppe_r -lpthreads -lmpci_r -lc /usr/lib/libc.a -bI:/lib/threads.exp -bI:/lib/syscalls.exp -L/usr/lib/threads -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/ip**	**ld -o** *myprog myprog.o* **/lib/crt0_r.o -binitfini:poe_remote_main -bnso -bI:/usr/lpp/ssp/css/libus/fs_ext.exp -lmpi_r -lvtd_r -lc_r -lppe_r -lpthreads -lmpci_r -lc /usr/lib/libc.a -bI:/lib/threads.exp -bI:/lib/syscalls.exp -L/usr/lib/threads -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/us**
Create a Fortran static executable which uses threaded MPI	**ld -o** *myprog myprog.o* **/lib/crt0_r.o -binitfini:poe_remote_main -bnso -lmpci_r -lmpi_r -lvtd_r -lxlf90_r -lc_r -lppe_r -lc -lpthreads /usr/lib/libc.a -bI:/lib/syscalls.exp -bI:/lib/threads.exp -L/usr/lib/threads -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/ip**	**ld -o** *myprog myprog.o* **/lib/crt0_r.o -binitfini:poe_remote_main -bnso -bI:/usr/lpp/ssp/css/libus/fs_ext.exp -lmpci_r -lmpi_r -lvtd_r -lxlf90_r -lc_r -lppe_r -lc -lpthreads /usr/lib/libc.a -bI:/lib/syscalls.exp -bI:/lib/threads.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/us**

Notes:

1. Users of PE 2.1 and 2.2 who have made references to any **crt0** in **/usr/lpp/ppe.poe/lib** (for example, users who create statically bound executables) and who wish to recompile using PE 2.4 should do the following:
   a. References to any **crt0** in **/usr/lpp/ppe.poe/lib** should be changed to the desired **crt0** in **/lib** or **/usr/lpp/xlC/lib**.
   b. The **-binitfini:poe_remote_main** binder option should be added to the compile or **ld** statement.
2. POE compile scripts utilize the **-binitfini** binder option. As a result, POE programs have a priority default of zero. If other user applications are using the **initfini** binder option, they should only specify a priority in the range of 1 to 2,147,483,647.
3. If you try to create a US static executable on the SP control workstation, and the **ld** command fails because it cannot find the **mpci** library file, it is possible that a link needs to be set by your system administrator. Refer to *IBM Parallel Environment for AIX: Installation* for instructions on installing PE on the SP control workstation.
4. On a cluster, you can create an IP static executable only. The US libraries are only shipped with an SP system.
5. When creating a Fortran static executable, include the **xlf90** and **xlf** libraries in the **ld** command after the **-lvtd** statement.
6. To use threads and Fortran, you should have Fortran Release 4.1.0.1 or later .
7. To create a static executable of a program which uses LAPI subroutines, see “Understanding and Using the Communications Low-Level Application Programming Interface (LAPI),” in *IBM Parallel System Support Programs for AIX: Administration Guide*

Step 2: Copy Files to Individual Nodes

Note: You only need to perform this step if your executable, your data files, and (if you plan to use **pdbx**) your source code files are not in a commonly accessed, or *shared*, file system. If running on an SP system, you can also skip this step if the needed files are part of a file collection which is distributed automatically. For information on using file collections, see *IBM Parallel System Support Programs for AIX: Administration Guide* For more information on the parallel debuggers, see *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*.

If the program you are running is in a shared file system, the Partition Manager loads a copy of your executable in each processor node in your partition when you invoke a program. If your executable is in a private file system, however, you must copy it to the nodes in your partition. If you plan to use the parallel debugger **pdbx**, you must copy your source files to all nodes as well. You can easily copy files to nodes using the **mprcp** command. All you do is pass **mprcp** the name of the host list file you are using to define your partition and the absolute path name of the file.

For example, to send a copy of *program* to all the processor nodes listed in *host.list* in your current directory:

ENTER **mprcp** *host.list $PWD/program*

- The **mprcp** command copies *program* to each of the nodes listed in *host.list* using the **rcp** command. If a program of the same name already exists, the **mprcp** command will overwrite it.

For more information on the **rcp** command, refer to *IBM AIX Version 4 Commands Reference* . For more information on the **mprcp** command, see Appendix A, “Parallel Environment Commands” on page 83.

You can also copy your executable to each node with the **mcp** command. There is an advantage in using **mcp** over **mprcp** in that **mcp** copies large programs faster. **mcp** uses the message passing facilities of the Parallel Environment to copy a file from a file system on the home node to a remote node file system. For example, assume that your executable program is on a mounted file system (*/u/edgar/somedir/myexecutable*), and you want to make a private copy in */tmp* on each node in *host.list*.

ENTER **mcp** */u/edgar/somedir/myexecutable /tmp/myexecutable* **-procs** *n*

Note: If you load your executable from a mounted file system, you may experience an initial delay while the program is being initialized on all nodes. You may experience this delay even after the program begins executing, because individual pages of the program are brought in on demand. This is particularly apparent during initialization of the message passing interface; since individual nodes are synchronized, there are simultaneous demands on the network file transfer system. You can minimize this delay by copying the executable to a local file system on each node, using the **mcp** message passing file copy program.

Step 3: Set Up the Execution Environment

This step contains the following sections:

Before invoking your program, you need to set up your execution environment. There are a number of POE environment variables discussed throughout this book and summarized in Appendix B, "POE Environment Variables and Command-Line Flags" on page 139. Any of these environment variables can be set at this time to later influence the execution of parallel programs. This step covers those environment variables most important for successful invocation of a parallel program. When you invoke a parallel program, your home node Partition Manager checks these environment variables to determine:

- the number of tasks in your program as specified by the **MP_PROCS** environment variable.
- how to allocate processor nodes for these tasks. There are two basic methods of node allocation – specific and non-specific.

  For *specific node allocation*, the Partition Manager reads an explicit list of nodes contained in a *host list file* you create. If you are using an RS/6000 network cluster, or if you are using a mixed system and want to include nodes not on the SP system, you must use this method of node allocation.

  For *non-specific node allocation*, you give the Partition Manager the name or number of a LoadLeveler pool, or the number of an SP system pool. A pool name or number may also be provided in a host list file when using LoadLeveler, or a list of SP system pools may be provided if using the Resource Manager. The Partition Manager then connects to LoadLeveler or the SP system Resource Manager, which allocates nodes from the specified pool(s) for you.

Note: If you are using an SP system, and plan to use its high performance switch adapter for communication, note that each node has been configured by your system administrator for communication using the IP communication subsystem, the US communication subsystem, or both. Any node you request through specific or non-specific node allocation must be configured for the appropriate communication subsystem library implementation. Check with your system administrator to learn which nodes were initialized for the US communication subsystem, which were initialized for the IP communication subsystem, or which nodes allow either.

There are five separate environment variables that, collectively, determine how nodes are allocated by the Partition Manager. While these are the only ones you must set to allocate nodes, keep in mind that there are many other environment variables you can set. These are summarized in Appendix B, “POE Environment Variables and Command-Line Flags” on page 139, and control such things as standard I/O handling and VT trace file generation. The environment variables for node allocation are:

MP_HOSTFILE which specifies the name of a host list file to use for node allocation. If set to an empty string (*“ ”*) or to the word “*NULL*”, this environment variable specifies that no host list file should be used. If **MP_HOSTFILE** is not set, POE looks for a file *host.list* in the current directory. You need to create a host list file if you want specific node allocation, or if you want non-specific node allocation from a number of SP system pools.

MP_RESD which specifies whether or not the Partition Manager should connect to a job management system (LoadLeveler or the Resource Manager) to allocate nodes.

Notes:

1. **MP_RESD** only specifies whether or not to use a job management system.
2. When the Resource Manager is used, the actual system you are using must be identified by the environment variable **SP_NAME**, of the control workstation on the SP system.
3. When running POE from a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the external node (see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation* for more information).
4. When running POE from a workstation that is external to the SP system, and using the Resource Manager, the *ssp.clients* fileset must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information).

MP_EUILIB which specifies the communication subsystem library implementation to use – either the IP communication subsystem implementation or the User Space (US) communication subsystem implementation. The IP communication subsystem library uses Internet Protocol for communication among processor nodes, while the US communication subsystem library lets you drive an SP system's high performance switch directly from your parallel tasks, without going through the kernel or operating system. For US communication on an SP system, you must have the high performance switch feature.

MP_EUIDEVICE which specifies the adapter set you want to use for IP communication among processor nodes. The Partition Manager only checks this if you are using the IP communication subsystem implementation with LoadLeveler or the SP system Resource Manager. It does not check this if you are using an RS/6000 network cluster. If **MP_RESD**=*no*, the value of **MP_EUIDEVICE** is ignored.

MP_RMPOOL	which specifies the name or number of a LoadLeveler pool, or number of an SP system pool. The Partition Manager only checks this if you are using LoadLeveler or the SP system Resource Manager for non-specific node allocation without a host list file. You can use the **llstatus** command to return information about LoadLeveler pools. To use **llstatus** on a workstation that is external to the LoadLeveler system, the *LoadL.so* fileset must be installed on the external node. For more information, see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation*. You can use the **jm_status** command to return information about SP system pools. To use **jm_status** on a workstation that is external to the SP system, the *ssp.clients* fileset must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information). For information on this command, and on SP system pools in general, refer to *IBM Parallel System Support Programs for AIX: Command and Technical Reference*

The remainder of this step consists of sub-steps describing how to set each of these environment variables, and how to create a host list file. Depending on the hardware and message passing library you are using, and the method of node allocation you want, some of the sub-steps that follow may not apply to you. For this reason, pay close attention to the task variant tables at the beginning of many of the sub-steps. They will tell you whether of not you need to perform the sub-step.

For further clarification, the following tables summarize the procedure for determining how nodes are allocated. The tables describe the possible methods of node allocation available to you, what each environment variable must be set to, and whether or not you need to create a host list file. To make the procedure of setting up the execution environment easier and less prone to error, you may eventually wish to create a shell script which automates some of the environment variable settings. To allocate nodes of an SP system, see Table 1. If you are using an RS/6000 network cluster, or if you are using a mixed system and want to allocate nodes not on the SP system, see Table 2 on page 18.

Table 1. Execution Environment Setup Summary (for an SP system)

	If you want to use the US communication subsystem library for communication among parallel tasks and...	**If you want to use the IP communication subsystem library for communication among parallel tasks and...**

	you want non-specific node allocation from a single pool:	**you want specific node allocation or if you want non-specific node allocation from more than one pool:**	**you want non-specific node allocation from a single pool:**	**you want specific node allocation or non-specific node allocation (using the Resource Manager) from more than one pool:**
A host list file...	not required.	required.	not required.	required.
MP_HOSTFILE	should be set to an empty string ("") or the word "*NULL*"	should be set to the name of your host list file. If not set, the host list file is assumed to be *host.list* in the current directory.	should be set to an empty string ("") or the word "*NULL*"	should be set to the name of your host list file. If not set, the host list file is assumed to be *host.list* in the current directory.
MP_RESD	should be set to *yes*. If set to an empty string (""), or if not set, the Partition Manager assumes **MP_RESD** is *yes*.	should be set to *yes*. If set to an empty string (""), or if not set, the Partition Manager assumes **MP_RESD** is *yes*.	should be set to *yes*. If set to an empty string (""), or if not set, the Partition Manager assumes **MP_RESD** is *yes*.	should be set to *yes*. If set to an empty string (""), the Partition Manager assumes **MP_RESD** is *no*.
MP_EUILIB	*us*	*us*	*ip*	*ip*
MP_EUIDEVICE	*css0* (the high performance switch). However, the actual value is ignored when **MP_EUILIB** is set to *us*.	*css0* (the high performance switch). However, the actual value is ignored when **MP_EUILIB** is set to *us*.	should specify the adapter type. A valid, case-sensitive value is *css0* (the high performance switch). Note that the **MP_EUIDEVICE** value is only used when the value of **MP_EUILIB** is *ip*.	should specify the adapter type. A valid, case-sensitive value is *css0* (the high performance switch). Note that the **MP_EUIDEVICE** value is only used when the value of **MP_EUILIB** is *ip*.
MP_RMPOOL	should be set to the name or number of a LoadLeveler pool, or the number of an SP system pool. It must be used if you are not using a host list file.	is ignored if you are using a host list file.	should be set to the name or number or a LoadLeveler pool, or the number of an SP system pool. It must be used if you are not using a host list file.	is ignored if you are using a host list file.

Table 2. Execution Environment Setup Summary (for RS/6000 Network Cluster or Mixed System)		
	If you are using an RS/6000 network cluster:	**If you are using a mixed system:**
A host list file...	is used.	is used.
MP_HOSTFILE	should be set to the name of a host list file. If not defined, the host list file is assumed to be *host.list* in the current directory.	should be set to the name of a host list file. If not defined, the host list file is assumed to be *host.list* in the current directory.
MP_RESD	should be set to *no*.	should be set to *yes*. If set to an empty string (*""*), the Partition Manager assumes **MP_RESD** is *no*.
MP_EUILIB	should be set to *ip*.	should be set to *ip*.
MP_EUIDEVICE	is not checked.	should specify the adapter type. Valid, case-sensitive, values are *en0* (Ethernet), *tr0* (token ring), *fi0* (FDDI), and *css0* (the high performance switch).
MP_RMPOOL	is not used because you are using a host list file.	is not used because you are using a host list file.

The following table shows how nodes will be allocated depending on the value of the environment variables discussed in this step. It is provided here for additional illustration. Refer to it in situations when the environment variables are set in patterns other than those suggested in Table 1 on page 16 and Table 2.

Table 3 (Page 1 of 2). Node Allocation Summary					
If	**Then**				
The value of MP_EUILIB is:	**The value of MP_RESD is:**	**Your Host List file contains a list of:**	**The allocation mode will be:**	**The communication subsystem library implementation used will be:**	**The message passing address used will be:**
ip	–	nodes	Node_List	IP	Nodes
pools	RM_List	IP	MP_EUIDEVICE	NULL	RM
IP	MP_EUIDEVICE	*yes*	nodes	RM_List	IP
MP_EUIDEVICE	pools	RM_List	IP	MP_EUIDEVICE	NULL
RM	IP	MP_EUIDEVICE	*no*	nodes	Node_List
IP	Nodes	pools	Error	–	–
NULL	Error	–	–		
us	–	nodes	RM_List	US	N/A
pools	RM_List	US	N/A	NULL	RM
US	N/A	*yes*	nodes	RM_List	US
N/A	pools	RM_List	US	N/A	NULL
RM	US	N/A	*no*	nodes	Error
–	–	pools	Error	–	–
NULL	Error	–	–		
–	–	nodes	Node_List	IP	Nodes
pools	RM_List	IP	MP_EUIDEVICE	NULL	RM
US	N/A	*yes*	nodes	RM_List	US
N/A	pools	RM_List	US	N/A	NULL

Table 3 (Page 2 of 2). Node Allocation Summary

If	Then				
RM	US	N/A	*no*	nodes	Node_List
IP	Nodes	pools	Error	–	–
NULL	Error	–	–		

Note:

Node_List	means that the host list file is used to create the partition.
RM_List	means that the host list file is used to create the partition, but the nodes are requested from either LoadLeveler or the SP system Resource Manager.
RM	means that the partition is created by requesting nodes in **MP_RMPOOL** from the SP system Resource Manager.
Nodes	indicates that the external IP address of the processor node is used for communication.
MP_EUIDEVICE	indicates that the IP adapter address indicated by **MP_EUIDEVICE** is used for communication.

Step 3a: Set the MP_PROCS Environment Variable

Before you execute a program, you need to set the size of the partition. To do this, use the **MP_PROCS** environment variable or its associated command-line flag **-procs**. For example, say you want to specify the number of task processes as 6. You could:

Set the MP_PROCS environment variable:	Use the -procs flag when invoking the program:
ENTER **export MP_PROCS=6**	**ENTER** **poe** *program* **-procs 6**

Invoking parallel programs is discussed in more detail in "Step 5: Invoke the Executable" on page 34.

Notes:

1. Keep in mind that **MP_PROCS** sets the number of task processes per partition and does not necessarily correspond to the number of processor nodes. If tasks are time-sharing processor nodes, for example, the number of tasks will be greater than the number of nodes.
2. If you do not set **MP_PROCS**, the default number of task processes is 1 unless:
   - you are using LoadLeveler
   - you are using **MP_RMPOOL**
   - both **MP_NODES** and **MP_TASK_PER_NODES** are set.

   See "Step 3i: Set the MP_RMPOOL Environment Variable" on page 32 for more details.
3. The examples in this book assume use of the Korn shell. If you are using the C shell, you would have to use the **setenv** command rather than the **export** command. See "Setting POE Environment Variables" on page xiv for more information.

Step 3b: Set the SP_NAME Environment Variable

If all nodes to be used for the parallel job exist in a PSSP 2.3.0 or 2.4.0 partition, the **SP_NAME** environment variable should be set to the name of the control workstation of the SP system on which these nodes exist. This is the only case that results in POE contacting the Resource Manager rather than LoadLeveler for node allocation requests.

Step 3c: Create a Host List File

You need to create a host list file if:	You do not need to create a host list file if:
• you are using an RS/6000 network cluster. • you are using a mixed system with the Resource Manager and want to allocate some nodes not on the SP system. • you are using an SP system and want specific node allocation. • you are using an SP system with the Resource Manager and want non-specific node allocation from more than one pool.	you are using a LoadLeveler cluster or an SP system and want non-specific node allocation from a single pool.

A host list file specifies the processor nodes on which the individual tasks of your program should run. When you invoke a parallel program, your Partition Manager checks to see if you have specified a host list file. If you have, it reads the file to allocate processor nodes. The procedure for creating a host list file differs depending on whether you are using an RS/6000 network cluster, a LoadLeveler cluster , an SP system, or a mixed system. If you are using an RS/6000 network cluster, see “Creating a Host List File to Allocate Nodes of a Cluster.” If you are using a LoadLeveler cluster , an SP system, or a mixed system, see “Creating a Host List File to Allocate Nodes of an SP System” on page 21.

Creating a Host List File to Allocate Nodes of a Cluster: If you are using an RS/6000, a host list file simply lists a series of host names – one per line. These must be the names of remote nodes accessible from the Home Node. Lines beginning with an exclamation point (!) or asterisk (*) are comments. The Partition Manager ignores blank lines and comments. The host list file can list more names than are required by the number of program tasks. The additional names are ignored.

To understand how the Partition Manager uses a host list file to determine the nodes on which your program should run, consider the following example host list file:

```
! Host list file for allocating 6 tasks

* An asterisk may also be used to indicate a comment



host1_name

host2_name

host3_name

host4_name

host5_name

host6_name
```

The Partition Manager ignores the first two lines because they are comments, and the third line because it is blank. It then allocates *host1_name* to run task 0, *host2_name* to run task 1, *host3_name* to run task 2, and so on. If any of the processor nodes listed in the host list file are unavailable when you invoke your program, the Partition Manager returns a message stating this and does not run your program.

You can also have multiple tasks of a program share the same node by simply listing the same node multiple times in your host list file. For example, say your host list file contains the following:

```
host1_name

host2_name

host3_name

host1_name

host2_name

host3_name
```

Tasks 0 and 3 will run on *host1_name*, tasks 1 and 4 will run on *host2_name*, and tasks 2 and 5 will run on *host3_name*.

Creating a Host List File to Allocate Nodes of an SP System: If you are using
| a LoadLeveler cluster or SP system, you can use a host list file for either:

- | non-specific node allocation from one system pool only, when using
| LoadLeveler. When using the Resource Manager, you can have one or more
| SP system pools.
- specific node allocation. If you are using a mixed system and want to allocate nodes that are not on the SP system, you must use specific node allocation.

In either case, the host list file can contain a number of records – one per line. For specific node allocation, each record indicates a processor node. For non-specific
| node allocation you can have one system pool only, when using LoadLeveler .
| When using the Resource Manager , each record indicates an SP system pool.

Your host list file cannot contain a mixture of node and pool requests, so you must use one method or the other. The host list file can contain more records than required by the number of program tasks. The additional records are ignored.

For specific node allocation:: Each record is either a host name or IP adapter address of a specific processor node of the SP system. If you are using a mixed system and want to allocate nodes not on the SP system, you must request them by host name. Lines beginning with an exclamation point (!) and asterisk (*) are comments. The Partition Manager ignores blank lines and comments.

To understand how the Partition Manager uses a host list file to determine the SP system nodes on which your program should run, consider the following representation of a host list file.

```
! Host list file for allocating 6 tasks

host1_name
host2_name
host3_name
9.117.8.53
9.117.8.53
9.117.8.53
```

The Partition Manager ignores the first line because it is a comment, and the second because it is blank. It then allocates *host1_name* to run task 0, *host2_name* to run task 1, *host3_name* to run task 2, and so on. The last three nodes are requested by adapter IP address using dot decimal notation.

Notes:

1. You can also, on each of the records in the host list file, specify how the allocated node's adapter and CPU should be used. For more information, see “Specifying How a Node's Resources Are Used” on page 24.
2. If any of the processor nodes listed in the host list file are unavailable when you invoke your program, the Partition Manager returns a message stating this and does not run your program.

For non-specific node allocation from a number of pools: After installation of a LoadLeveler cluster or SP system, your system administrator divides its processor nodes into a number of pools. With LoadLeveler, each pool has an identifying pool name or number. With an SP system, each pool has an identifying pool number. Using LoadLeveler for non-specific node allocation, you need to supply the appropriate pool name or number. LoadLeveler does not use more than one pool. Using Resource Manager for non-specific node allocation from a number of pools, you need to supply the appropriate pool numbers.

If you require information about LoadLeveler pools, use the command **llstatus**. To use **llstatus** on a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the external node (see *Using and Administering LoadLeveler* for more information).

ENTER **llstatus -l** (lower case L)

- LoadLeveler lists information about pools in the LoadLeveler cluster.

If you require information about SP system pools, use the command **jm_status**. To use **jm_status** on a workstation that is external to the SP system, the *ssp.clients* fileset must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information).

ENTER **jm_status -P**

- The Resource Manager lists information about all SP system pools.

With regard to LoadLeveler, in a host list file intended for non-specific node allocation, each record is a pool name or number preceded by an at symbol (@). Lines beginning with an exclamation point (!) and asterisk (*) are comments. The Partition Manager ignores blank lines and comments.

To understand how the Partition Manager uses a host list file for non-specific node allocation, consider the following example host list file:

```
! Host list file for allocating 3 tasks with LoadLeveler

@6
@6
@6
```

The Partition Manager ignores the first line because it is a comment, and the second line because it is blank. The at (@) symbols tell the Partition Manager that these are pool requests. It connects to LoadLeveler to request three nodes from pool 6.

With regard to the Resource Manager only, in a host list file intended for non-specific node allocation from a number of pools, each record is a pool number preceded by an at symbol (@). Lines beginning with an exclamation point (!) and asterisk (*) are comments. The Partition Manager ignores blank lines and comments.

To understand how the Partition Manager uses a host list file for non-specific node allocation from a number of pools, consider the following example host list file:

```
! Host list file for allocating 6 tasks with the Resource Manager

@6
@6
@6
@12
@12
@12
```

The Partition Manager ignores the first line because it is a comment, and the second line because it is blank. The at (@) symbols tell the Partition Manager that these are pool requests. It connects to the SP system Resource Manager to request three nodes from pool 6, and three nodes from pool 12.

Notes:

1. When using the Resource Manager you can also, on each of the records in the host list file, specify how the allocated node's adapter and CPU should be used. For more information, see “Specifying How a Node's Resources Are Used” on page 24.
2. If there are insufficient nodes available in a requested pool when you invoke your program, the Partition Manager returns a message stating this, and does not run your program.

3. For more information on the **llstatus** command and LoadLeveler pools, refer to *Using and Administering LoadLeveler*. For more information on the **jm_status** command and SP system pools, refer to *IBM Parallel System Support Programs for AIX: Command and Technical Reference*.
4. If the number of program tasks is greater than the number of records in the host list file, the last record in the file is used for the remaining requests.

Specifying How a Node's Resources Are Used: When requesting nodes of an SP system, you can optionally request how each node's resources – its adapter and CPU – should be used. You can specify:

- Whether the node's adapter is to be *dedicated* or *shared*. If dedicated, only a single program task can use it for the same protocol. If *shared*, a number of tasks on that node can use it. If you are using the US communication subsystem library implementation, POE forces the adapter use to be dedicated (see Table 4 on page 25).
- Whether the node's CPU usage should be *unique* or *multiple*. If *unique*, only your program's tasks can use the CPU. If *multiple*, your program may share the node with other users.

Note: When using LoadLeveler, you can request how nodes are used with the **MP_CPU_USE** and/or **MP_ADAPTER_USE** environment variables, or their associated command line options. Usage specification in a host list file will be ignored when using LoadLeveler.

When Using a Host List File for Node Allocation: With regard to the Resource Manager , on each record of the host list file, you can make either or both of the specifications listed above . For example, if you wanted your program task to have exclusive use of both the adapter and CPU, the host list record would be:

```
host1_name dedicated unique
```

or

```
host1_name d u
```

This is the same for pool requests:

```
@6 dedicated unique
```

or

```
@6 d u
```

When Not Using a Host List File for Node Allocation: The environment variables **MP_ADAPTER_USE** and **MP_CPU_USE**, or the associated command line options (**-adapter_use** and **-cpu_use**) can be used to make either or both of these specifications. These specifications will then affect the resource usage for each node allocated from the pool specified using **MP_RMPOOL** or **-rmpool**. For example, if you wanted nodes from Resource Manager pool 5, and you wanted your program to have exclusive use of both the adapter and CPU, the following command line could be used:

poe [*program*] *-rmpool 5 -adapter_use d*[*edicated*]

-cpu_use u[*nique*] [*more_poe_options*]

Associated environment variables (**MP_RMPOOL**, **MP_ADAPTER_USE**, **MP_CPU_USE**) could also be used to specify any or all of the options in this example.

The following tables illustrate how node resources are used. Table 4 shows the default settings for adapter and CPU use, while Table 5 outlines how the two separate specifications determine how the allocated node's resources are used.

Table 4. Adapter/CPU Default Settings

	Adapter	**CPU**
If host list file contains non-specific pool requests:	Dedicated	Unique
If host list file requests specific nodes:	Shared [1]	Multiple
If host list file is not used nodes:	Dedicated[2]	Unique[3]

Note:

[1] For US jobs, adapter is dedicated.

[2] For IP jobs, adapter is shared.

[3] For IP jobs, CPU is multiple.

Table 5. Adapter/CPU Use under LoadLeveler

	If the Node's CPU is "Unique":	**If the Node's CPU is "Multiple":**
If the adapter use is "Dedicated":	Intended for production runs of high performance applications. Only the tasks of that parallel job use the adapter and CPU.	The adapter you specified with **MP_EUIDEVICE** is dedicated to the tasks of your parallel job. However, you and other users still have access to the CPU through another adapter.
If the adapter use is "Shared":	Only your program tasks have access to the node's CPU, but other program's tasks can share the adapter.	Both the adapter and CPU can be used by a number of your program's tasks and other users.

Table 6. Adapter/CPU Use under the Resource Manager

	If the Node's CPU is "Unique":	**If the Node's CPU is "Multiple":**
If the adapter use is "Dedicated":	Intended for production runs of high performance applications. Only one task uses the adapter and CPU.	The adapter you specified with **MP_EUIDEVICE** is dedicated to your program task. However, you and other users still have access to the CPU through another adapter.
If the adapter use is "Shared":	Only you have access to the node's CPU, but a number of your program's tasks can share the adapter.	Both the adapter and CPU can be used by a number of your program's tasks and other users.

Notes:

1. When using LoadLeveler, the US communication subsystem library does not require dedicated use of the SP switch on the node. Adapter use will be defaulted, as in Table 4, but shared usage may be specified.
2. When using the Resource Manager, the US communication subsystem library requires dedicated use of the SP switch on the node. If you are using the US communication subsystem for communication among processor nodes, POE forces adapter use to be dedicated. If you are using the US communication

subsystem and you specify adapter use to be shared, the specification is ignored.

3. Adapter/CPU usage specification is only enforced for jobs using LoadLeveler or the SP system Resource Manager for node allocation.

Generating an Output Host List File: When running parallel programs in a LoadLeveler cluster or on an SP system, you can generate an output host list file of the nodes allocated by LoadLeveler or the Resource Manager. When you have LoadLeveler or the Resource Manager perform non-specific node allocation from SP system pools, this enables you to learn which nodes were allocated. This information is vital if you want to perform some postmortem analysis or file cleanup on those nodes, or if you want to rerun the program using the same nodes. To generate a host list file, set the **MP_SAVEHOSTFILE** environment variable to a file name. You can specify this using a relative or full path name. As with most POE environment variables, you can temporarily override the value of **MP_SAVEHOSTFILE** using its associated command-line flag **-savehostfile**. For example, to save LoadLeveler's or the Resource Manager's node allocation into a file called */u/hinkle/myhosts*, you could:

Set the MP_SAVEHOSTFILE environment variable:	Use the -savehostfile flag when invoking the program:
ENTER export MP_SAVEHOSTFILE=/u/hinkle/myhosts	**ENTER poe** *program* **-savehostfile /u/hinkle/myhosts**

Each record in the output host list file will be the original non-specific pool request. Following each record will be comments indicating the specific node that was allocated. The specific node is identified by:

- hostname
- external IP address
- adapter IP address (which may be the same as the external IP address)

For example, using LoadLeveler, say the input host list file contains the following records:

```
@mypool
@mypool
@mypool
```

The following is a representation of the output hostlist file.

```
host1_name
! 9.117.11.47                9.117.8.53

!@mypool
host1_name
! 9.117.11.47                9.117.8.53

!@mypool
host1_name
! 9.117.11.47                9.117.8.53

!@mypool
```

Using the Resource Manager, say the input host list file contains the following records:

```
@6

@6

@6

@12

@12

@12
```

The following is a representation of the output hostlist file.

```
host1_name dedicated unique

! 9.117.11.47                  9.117.8.53

!@6

host2_name dedicated unique

! 9.117.11.47                  9.117.8.53

!@6

host3_name dedicated unique

! 9.117.11.47                  9.117.8.53

!@6

host4_name dedicated unique

! 9.117.11.47                  9.117.8.53

!@12

host5_name dedicated unique

! 9.117.11.47                  9.117.8.53

!@12

host6_name dedicated unique

! 9.117.11.47                  9.117.8.53

!@12
```

Note: The name of your output host list file can be the same as your input host list file. If a file of the same name already exists, it is overwritten by the output host list file.

Step 3d: Set the MP_HOSTFILE Environment Variable

You need to set the MP_HOSTFILE environment variable if:	**You do not need to set the MP_HOSTFILE environment variable if:**
• you are using a host list file other than the default *./host.list* • you are requesting non-specific node allocation without a host list file.	If your host list file is the default *./host.list*

The default host list file used by the Partition Manager to allocate nodes is called *host.list* and is located in your current directory. You can specify a file other than *host.list* by setting the environment variable **MP_HOSTFILE** to the name of a host list file, or by using either the **-hostfile** or **-hfile** flag when invoking the program. In either case, you can specify the file using its relative or full path name. For example, say you want to use the host list file *myhosts* located in the directory */u/hinkle*. You could:

Set the MP_HOSTFILE environment variable:	**Use the -hostfile flag when invoking the program:**
ENTER **export MP_HOSTFILE=/u/hinkle/myhosts**	**ENTER** **poe** *program* **-hostfile /u/hinkle/myhosts** or **poe** *program* **-hfile /u/hinkle/myhosts**

| If you are using LoadLeveler or the SP system Resource Manager for non-specific
| node allocation from a single pool specified by **MP_RMPOOL**, and a host list file
| exists in the current directory , you must set **MP_HOSTFILE** to an empty string or
| to the word “*NULL*”. Otherwise the Partition Manager uses the host list file. You can either:

Set the MP_HOSTFILE environment variable:	**Use the -hostfile flag when invoking the program:**
ENTER **export MP_HOSTFILE=** or **export MP_HOSTFILE=***""* or **export MP_HOSTFILE=***NULL*	**ENTER** **poe** *program* **-hostfile** *""* or **poe** *program* **-hostfile** *NULL*

Step 3e: Set the MP_RESD Environment Variable

| To indicate whether a job management system should be used, you set the
| **MP_RESD** environment variable to *yes* or *no*. As specified in Table 1 on page 16
| and Table 2 on page 18, **MP_RESD** controls whether or not the Partition Manager
| connects to LoadLeveler or the Resource Manager to allocate processor nodes.

| If you are allocating nodes that are **not** part of a LoadLeveler cluster, **MP_RESD**
| should be set to *no*. If **MP_RESD** is set to *yes*, only nodes within the LoadLeveler
| cluster are allocated.

| If you are allocating nodes of an RS/6000 network cluster, you do not have a job
| management system and should set **MP_RESD** to *no*. If you are using a mixed
| system, you may set **MP_RESD** to *yes*. However, the job management system only
| has knowledge of SP system nodes. To allocate any of the additional RS/6000
| processors which supplement the SP system nodes in a mixed system, you must
| also use a host list file.

As with most POE environment variables, you can temporarily override the value of **MP_RESD** using its associated command-line flag **-resd**. For example, to specify that you want the Partition Manager to connect to the Resource Manager, you could:

Set the MP_RESD environment variable:	Use the -resd flag when invoking the program:
ENTER **export MP_RESD=***yes*	**ENTER** **poe** *program* **-resd** *yes*

You can also set **MP_RESD** to an empty string. If set to an empty string, or if not set, the default value of **MP_RESD** is interpreted as *yes* or *no* depending on the context. Specifically, the value of **MP_RESD** will be determined by the value of **MP_EUILIB** and whether or not you are using a host list file. The following table shows how the context determines the value of **MP_RESD**.

MP_EUILIB setting	and you are using a host list file:	and you are not using a host list file:
If MP_EUILIB is set to ip, an empty string, the word "NULL", or if not set:	**MP_RESD** is interpreted as *no* by default, unless host list file includes pool requests.	**MP_RESD** is interpreted as *yes* by default.
If MP_EUILIB is set to us:	**MP_RESD** is interpreted as *yes* by default.	**MP_RESD** is interpreted as *yes* by default.

| **Notes:**

| 1. **MP_RESD** only specifies whether or not to use a job management system
| (LoadLeveler or the Resource Manager).

| 2. When the Resource Manager is used, the actual system you are using is
| identified by the environment variable **SP_NAME**, of the control workstation on
| the SP system.

| 3. When running POE from a workstation that is external to the LoadLeveler
| cluster, the *LoadL.so* fileset must be installed on the external node (see *Using*
| *and Administering LoadLeveler* and *IBM Parallel Environment for AIX:*
| *Installation* for more information).

| 4. When running POE from a workstation that is external to the SP system, and
| using the Resource Manager, the *ssp.clients* fileset must be installed on the
| external node (see *IBM Parallel Environment for AIX: Installation* for more
| information).

Step 3f: Set the MP_EUILIB Environment Variable

During execution, the tasks of your program can communicate via calls to message passing routines. The message passing routines in turn call communication subsystem library routines which enable the processor nodes to exchange the message data. Before you invoke your program, you need to decide which communication subsystem library implementation you wish to use – the Internet Protocol (IP) communication subsystem or the User Space (US) communication subsystem.

- *The IP communication subsystem library implementation* uses the Internet Protocol for communication among processor nodes. If you do not have the high performance switch feature, you must use the IP communication subsystem.
- *The US communication subsystem library implementation* uses the User Space protocol for dedicated use of a high performance communication adapter.

Programs that use LAPI must set **MP_EUILIB** (or **-euilib**) to *us*. It allows you to drive the switch adapter directly from your parallel tasks. You can only use the US communication subsystem when running on an SP system configured with the high performance switch feature.

The **MP_EUILIB** environment variable, or its associated command-line flag **-euilib**, is used to indicate which communication subsystem library implementation you are using. POE needs to know which communication subsystem implementation to dynamically link in as part of your executable when you invoke it. The following table shows the appropriate setting for **MP_EUILIB** depending on the communication subsystem library implementation you want and whether or not it has already been statically linked.

	and you want it dynamically linked when you invoke your program:
If you want the IP communication subsystem or US communication subsystem:	**MP_EUILIB** should be set to *ip* or *us* This specification is case-sensitive.

For example, say you want to dynamically link in the communication subsystem library at execution time. You could:

Set the MP_EUILIB environment variable:	**Use the -euilib flag when invoking the program:**
ENTER **export MP_EUILIB=ip or us**	**ENTER** **poe** *program* **-euilib ip or us**

Note: When you invoke a parallel program, your Partition Manager checks the value of **MP_EUILIB** and then looks to the directory */usr/lpp/ppe.poe/lib* for the message passing interface and the communication subsystem library implementation. If you are running on an RS/6000 network cluster, this is the actual location of the message passing interface. If you are running on an SP system, */usr/lpp/ppe.poe/lib* contains symbolic links to the actual location. Consult your system administrator for the actual location of the message passing library if necessary.

You can make POE look to a directory other than */usr/lpp/ppe.poe/lib* by setting the **MP_EUILIBPATH** environment variable or its associated command-line flag **-euilibpath**. For example, say the communication subsystem library implementations were moved to */usr/altlib*. To instruct the Partition Manager to look there, you could:

Set the MP_EUILIBPATH environment variable:	**Use the -euilibpath flag when invoking the program:**
ENTER **export MP_EUILIBPATH=/usr/altlib**	**ENTER** **poe** *program* **-euilibpath /usr/altlib**

The expected library for loading the communication subsystem library implementation is in directory */usr/lpp/ppe.poe/lib/***$MP_EUILIB**. Setting the **MP_EUILIBPATH** environment variable causes POE to try to load the communication subsystem library from the directory **$MP_EUILIBPATH/$MP_EUILIB**. If the communication subsystem library (*libmpci.a*) is not in the requested path, it will be loaded from the library path for the IP communication subsystem library implementation used when the program was compiled – **$MP_PREFIX***/ppe.poe/lib/ip*. **MP_PREFIX** can also be set by the user, but is normally */usr/lpp*. Thus the default library path is normally */usr/lpp/ppe.poe/lib/ip*, provided the library is not specified by the **MP_EUILIB** and/or **MP_EUILIBPATH** environment variables.

Step 3g: Set the MP_EUIDEVICE Environment Variable

You need to set the MP_EUIDEVICE environment variable if:	You do not need to set the MP_EUIDEVICE environment variable if:
you have set the **MP_EUILIB** environment variable to *ip*, and are using LoadLeveler or the Resource Manager.	you have set the **MP_EUILIB** environment variable to *us*. The Partition Manager assumes that **MP_EUIDEVICE** is *css0* – the high performance communication adapter.

If you are using the IP communication subsystem library implementation for communication among parallel tasks on an SP system, you can specify which adapter set to use for message passing – either Ethernet, FDDI, token-ring, or a high performance switch. The **MP_EUIDEVICE** environment variable and its associated command-line flag **-euidevice** are used to select an alternate adapter set for communication among processor nodes. If neither **MP_EUIDEVICE** device nor the **-euidevice** flag is set, the communication subsystem library uses the external IP address of each remote node. The following table shows the possible, case-sensitive, settings for **MP_EUIDEVICE**.

Setting the MP_EUIDEVICE environment variable to:	Selects:
en0	The Ethernet adapter
fi0	The FDDI adapter
tr0	The token-ring adapter
css0	The high performance switch adapter

For example, say you want to use IP over the high performance switch. The nodes have been initialized for IP as described in *IBM Parallel System Support Programs for AIX: Installation and Migration Guide*, and you have already set the **MP_EUILIB** environment variable to *ip*. To specify the high performance switch, you could:

Set the MP_EUIDEVICE environment variable:	Use the -euidevice flag when invoking the program:
ENTER **export MP_EUIDEVICE=css0**	**ENTER** **poe** *program* **-euidevice css0**

Notes:

1. If you do not set the **MP_EUIDEVICE** environment variable, the default is the adapter set used as the external network address.
2. If **MP_EUIDEVICE** is explicitly set to *en0* and LoadLeveler is being used for node allocation, the *en0* adapter must be configured in LoadLeveler. See *Using and Administering LoadLeveler* for more information.

Step 3h: Set the MP_MSG_API Environment Variable

The **MP_MSG_API** environment variable, or its associated command line option, is used to indicate to POE which message passing API is being used by the parallel tasks.

You need to set the MP_MSG_API environment variable if:	You do not need to set the MP_MSG_API environment variable if:
A parallel task is using LAPI alone or in conjunction with MPI.	A parallel task is using MPI only.

Step 3i: Set the MP_RMPOOL Environment Variable

You need to set the MP_RMPOOL environment variable if:	You do not need to set the MP_RMPOOL environment variable if:
You are using a LoadLeveler cluster oran SP system and want non-specific node allocation from a single pool.	You are allocating nodes using a host list file.

After installation of a LoadLeveler cluster or SP system, your system administrator divides its processor nodes into a number of pools. Each pool has an identifying pool name or number. When using LoadLeveler, and you want non-specific node allocation from a single pool, you need to set the **MP_RMPOOL** environment variable to the name or number of that pool. When using the Resource Manager, and you want non-specific node allocation from a single pool, you need to set the **MP_RMPOOL** environment variable to the number of that pool. The pool number you specify should consist of nodes configured for the appropriate communication subsystem library implementation. Check with your system administrator to learn which pools consist of nodes initialized for the US communication subsystem and which were initialized for the IP communication subsystem.

If you need information about available pools and are using LoadLeveler, use the command **llstatus**. To use **llstatus** on a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the external node (see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation* for more information).

ENTER **llstatus -l** (lower case L)

- LoadLeveler lists information about all LoadLeveler pools and/or features.

If you need information about available pools and are using the Resource Manager , use the command **jm_status** to get job manager status. To use **jm_status** on a workstation that is external to the SP system, the *ssp.clients* fileset must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information).

ENTER **jm_status -P**

- The Resource Manager lists information about all SP system pools.

As with most POE environment variables, you can temporarily override the value of **MP_RMPOOL** using its associated command-line flag **-rmpool**. To specify pool 6, for example, you could:

Set the MP_RMPOOL environment variable:	Use the -rmpool flag when invoking the program:
ENTER **export MP_RMPOOL=6**	**ENTER** **poe** *program* **-rmpool 6**

Notes:

1. For more information on the **llstatus** command and on LoadLeveler pools, refer to *Using and Administering LoadLeveler*.
2. For more information on the **jm_status** command and on SP system pools, refer to *IBM Parallel System Support Programs for AIX: Command and Technical Reference*.

3. When using LoadLeveler, if the value of the **MP_RMPOOL** environment variable is numeric, that pool number must be configured in LoadLeveler. If the value of **MP_RMPOOL** contains any non-numeric characters, that pool name must be configured as a *feature* in LoadLeveler. See *Using and Administering LoadLeveler* for more information.

In conjunction with **MP_RMPOOL**, when using LoadLeveler, the **MP_NODES** or **MP_TASKS_PER_NODE** environment variables or associated command line options may be used.

- **MP_NODES** or **-nodes** is used to specify the number of physical nodes on which to run the parallel tasks. It may be used alone or in conjunction with **-tasks_per_node** and/or **-procs**, as described in Table 7 below.
- **MP_TASKS_PER_NODE** or **-tasks_per_node** is used to specify the number of tasks to be run on each of the physical nodes. It may be used in conjunction with **-nodes** and/or **-procs**, as described in Table 7 below, but may not be used alone.

Table 7. LoadLeveler Node Allocation

MP_PROCS set?	MP_TASKS_PER_NODE set?	MP_NODES set?	Conditions and Results
Yes	Yes	Yes	**MP_TASKS_PER_NODE** multiplied by **MP_NODES** must equal **MP_PROCS**, otherwise an error occurs.
Yes	Yes	No	**MP_TASKS_PER_NODE** must divide evenly into **MP_PROCS**, otherwise an error occurs.
Yes	No	Yes	**MP_NODES** (*n*) must be less than or equal to **MP_PROCS** (*p*). If less than, LoadLeveler will allocate one task to each node, from 0 to *n* - 1, and will then allocate a second task to each of the nodes from 0 to *n* - 1, etc., until there are *p* tasks allocated. For example, if *n* = 3 and *p* = 5, 2 tasks will run on node 0, 2 tasks will run on node 1, and 1 task will run on node 2.
Yes	No	No	The parallel job will run with the indicated number of **MP_PROCS** (*p*) on *p* nodes.
No	Yes	Yes	The parallel job will consist of **MP_TASKS_PER_NODE** multiplied by **MP_NODES** tasks.
No	Yes	No	An error occurs. **MP_NODES** or **MP_PROCS** *must* be specified with **MP_TASKS_PER_NODE**.
No	No	Yes	One parallel task will be run on each of *n* nodes.
No	No	No	One parallel task will be run on one node.

Note: The examples in this table use the environment variable setting to illustrate each of the three options. The associated command line options may also be used.

Step 3j: Set the MP_AUTH Environment Variable

You need to set the MP_AUTH environment variable if:	You do not need to set the MP_AUTH environment variable if:
You are using DFS/DCE based user authorization and your system administrator has not defined the **MP_AUTH** value in **/etc/poe.limits**.	You are using AIX based user authorization defined by **/etc/hosts.equiv** or **.rhosts** entries, or your system administrator has defined the **MP_AUTH** value in **/etc/poe.limits**.

POE allows two types of user authorization:

1. AIX based user authorization, using entries in **/etc/hosts.equiv** or **.rhosts** files. This is the default POE user authorization method.
2. DFS/DCE based user authorization, using DCE credentials. If you plan to run POE jobs in a DFS environment, you must use DFS/DCE based user authorization.

| **Note:** If POE is run under LoadLeveler, LoadLeveler handles the user
| authorization, and the POE user authorization steps are skipped.

The type of user authorization is controlled by the **MP_AUTH** environment variable. The valid values are **AIX** (the default) or **DFS**.

The system administrator can also define the value for **MP_AUTH** in the **/etc/poe.limits** file. If **MP_AUTH** is specified in **/etc/poe.limits**, POE will override the value of the **MP_AUTH** environment variable, if different.

For more information on running POE in a DFS environment, see “Running POE within a Distributed File System” on page 53.

For more information on user authorization and on the **/etc/poe.limits** entries, see *IBM Parallel Environment for AIX: Installation*

Step 4: Start X-Windows Analysis Tools

If you wish to use either of the POE X-Windows analysis tools – the Program Marker Array or the System Status Array – you should start them before invoking the executable. For more information on these tools and how to start them, see Figure 1 on page 75 and “Using the System Status Array” on page 78.

Step 5: Invoke the Executable

Note: In order to perform this step, you need to have a user account on, and be able to remotely login to, each of the processor nodes. This requires that you have an *.rhosts* file set up in your home directory on each of the remote processor nodes. Alternatively, your user id on the home node can be authorized in the */etc/hosts.equiv* file on each remote node. For more information on the TCP/IP *.rhosts* file format, see *IBM General Concepts and Procedures for RS/6000*, and *IBM AIX Version 4 Files Reference*

The **poe** command enables you to load and execute programs on remote nodes. You can use it to:

- load and execute an SPMD program onto all nodes of your partition. For more information, see “Invoking an SPMD Program” on page 36.

- individually load the nodes of your partition. This capability is intended for MPMD programs. For more information, see "Invoking an MPMD Program" on page 36.
- load and execute a series of SPMD or MPMD programs, in individual job steps, on the same partition. For more information, see "Loading a Series of Programs as Job Steps" on page 38.
- run non-parallel programs on remote nodes. For more information, see "Invoking a Non-Parallel Program On Remote Nodes" on page 41.

When you invoke **poe**, the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and reproduces your local environment, on each processor node. The Partition Manager also passes the option list to each remote node. If your program is in a shared file system, the Partition Manager loads a copy of it on each node. If your program is in a private file system, you will have already manually copied your executable to the nodes using the **mprcp** or **mcp** command. If you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time.

Since the Partition Manager attempts to reproduce your local environment on each remote node, your current directory is important. When you invoke **poe**, the Partition Manager will, immediately before running your executable, issue the **cd** command to your current working directory on each remote node. If you are in a local directory that does not exist on remote nodes, you will get an error as the Partition Manager attempts to change to that directory on remote nodes. Typically, this will happen when you invoke **poe** from a directory under */tmp*. We suggest that you invoke **poe** from a file system that is mounted across the system. If it is important that the current directory be under */tmp*, make sure that directory exists on all the remote nodes. If you are running in the C shell, see "Running Programs Under the C Shell" on page 67.

Note: The Parallel Environment opens several file descriptors before passing control to the user. The Parallel Environment will not assign *specific* file descriptors other than standard in, standard out, and standard error.

Before using the **poe** command, you can first specify which programming model you are using by setting the **MP_PGMMODEL** environment variable to either *spmd* or *mpmd*. As with most POE environment variables, you can temporarily override the value of **MP_PGMMODEL** using its associated command-line flag **-pgmmodel**. For example, if you want to run an MPMD program, you could:

Set the MP_PGMMODEL environment variable:	Use the -pgmmodel flag when invoking the program:
ENTER **export MP_PGMMODEL=***mpmd*	**ENTER** **poe** *program* **-pgmmodel** *mpmd*

Note: If you do not set the **MP_PGMMODEL** environment variable or **-pgmmodel** flag, the default programming model is SPMD.

Note: If you load your executable from a mounted file system, you may experience an initial delay while the program is being initialized on all nodes. You may experience this delay even after the program begins executing, because individual pages of the program are brought in on demand. This is particularly apparent during initialization of the message passing interface; since individual nodes are synchronized, there are simultaneous demands on the network file transfer system. You can minimize this delay by copying the executable to a local file system on each node, using the **mcp** message passing file copy program.

Invoking an SPMD Program

If you have an SPMD program, you want to load it as a separate task on each node of your partition. To do this, follow the **poe** command with the program name and any options. The options can be program options or any of the POE command-line flags shown in Appendix B, "POE Environment Variables and Command-Line Flags" on page 139. You can also invoke an SPMD program by entering the program name and any options:

ENTER **poe** *program [options]*

or

program [options]

You can also enter **poe** without a program name:

ENTER **poe** *[options]*

● Once your partition is established, a prompt appears.

ENTER the name of the program you want to load. You can follow the program name with any program options or a subset of the POE flags.

Note: For National Language Support, POE displays messages located in an externalized message catalog. POE checks the **LANG** and **NLSPATH** environment variables, and if either is not set, it will set up the following defaults:

- **LANG=C**
- **NLSPATH=/usr/lib/nls/msg/%L/%N**

For more information about the message catalog, see "National Language Support" on page xiv.

Invoking an MPMD Program

Note: You must set the **MP_PGMMODEL** environment variable or **-pgmmodel** flag to invoke an MPMD program.

With an SPMD application, the name of the same executable is sent to, and runs on, each of the processor nodes of your partition. If you are invoking an MPMD application, you are dealing with more than one program and need to individually load the nodes of your partition.

For example, say you have two programs – *master* and *workers* – designed to run together and communicate via calls to message passing subroutines. The program *master* is designed to run on one processor node. The *workers* program is designed to run as separate tasks on any number of other nodes. The *master* program will coordinate and synchronize the execution of all the worker tasks. Neither program can run without the other, as *master* only does sends and the *workers* tasks only do receives.

You can establish a partition and load each node individually using:

- standard input (from the keyboard or redirected)
- a POE commands file

Loading Nodes Individually From Standard Input: To establish a partition and load each node individually using STDIN:

ENTER **poe** *[options]*

● The Partition Manager allocates the processor nodes of your partition. Once your partition is established, a prompt containing both the logical node identifier 0 and the actual host name it maps to, appears.

ENTER the name of the program you want to load on node 0. You can follow the program name with any program options or a subset of the POE flags.

● A prompt for the next node in the partition displays.

ENTER the name of the program you want to load on each processor node as you are prompted.

● When you have specified the program to run on the last node of your partition, the message “Partition loaded...” displays and execution begins.

For additional illustration, the following shows the command prompts that would appear, as well as the program names you would enter, to load the example *master* and *workers* programs. This example assumes that the **MP_PROCS** environment variable is set to 5.

```
0:host1_name> master [options]

1:host2_name> workers [options]

2:host3_name> workers [options]

3:host4_name> workers [options]

4:host5_name> workers [options]

Partition loaded...
```

```
% poe

0:host1_name> master [options]

1:host2_name> workers [options]

2:host3_name> workers [options]

3:host4_name> workers [options]

4:host5_name> workers [options]

Partition loaded...
```

Note: You can use some POE command-line flags on individual program names, but not those that are used to set up the partition. The flags you can use are mainly those having to do with VT trace file collection. They are:

- **-infolevel** or **-ilevel**
- **-ttempsize** or **-ttsize**
- **-tmpdir**
- **-samplefreq** or **-sfreq**

- **-tbuffwrap** or **-tbwrap**
- **-tbuffsize** or **-tbsize**
- **-euidevelop**

Loading Nodes Individually Using a POE Commands File: The **MP_CMDFILE** environment variable, and its associated command-line flag **-cmdfile**, let you specify the name of a POE commands file. You can use such a file when individually loading a partition – thus freeing STDIN. The POE commands file simply lists the individual programs you want to load and run on the nodes of your partition. The programs are loaded in task order. For example, say you have a typical master/workers MPMD program that you want to run as 5 tasks. Your POE commands file would contain:

```
master [options]

workers [options]

workers [options]

workers [options]

workers [options]
```

Once you have created a POE commands file, you can specify it using a relative or full path name on the **MP_CMDFILE** environment variable or **-cmdfile** flag. For example, if your POE commands file is */u/hinkle/mpmdprog*, you could:

Set the MP_CMDFILE environment variable:	Use the -cmdfile flag on the poe command:
ENTER **export MP_CMDFILE=/u/hinkle/mpmdprog**	**ENTER** **poe -cmdfile /u/hinkle/mpmdprog**

Once you have set the **MP_CMDFILE** environment variable to the name of the POE commands file, you can individually load the nodes of your partition. To do this:

ENTER **poe** *[options]*

● The Partition Manager allocates the processor nodes of your partition. The programs listed in your POE commands file are run on the nodes of your partition.

Loading a Series of Programs as Job Steps

By default, the Partition Manager releases your partition when your program completes its run. However, you can set the environment variable **MP_NEWJOB**, or its associated command-line flag **-newjob**, to specify that the Partition Manager should maintain your partition for multiple job steps.

For example, say you have three separate SPMD programs. The first one sets up a particular computation by adding some files to */tmp* on each of the processor nodes on the partition. The second program does the actual computation. The third program does some postmortem analysis and file cleanup. These three parallel programs must run as job steps on the same processor nodes in order to work correctly. While specific node allocation using a host list file might work, the requested nodes might not be available when you invoke each program. The better solution is to instruct the Partition Manager to maintain your partition after execution of each program completes. You can then read multiple job steps from:

- standard input
- a POE commands file using the **MP_CMDFILE** environment variable.

In either case, you must first specify that you want the Partition Manager to maintain your partition for multiple job steps. To do this, you could:

Set the MP_NEWJOB environment variable:	**Use the -newjob flag on the poe command:**
ENTER **export MP_NEWJOB=***yes*	**ENTER** **poe -newjob** *yes*

Notes:

1. You can only load a series of programs as job steps using the **poe** command. You cannot do this with either of the parallel debugger commands – **pdbx** and **pedb**.
2. **poe** is its own shell. Whether successive steps run after a step completes is a function of the exit code, as described in *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*

Reading Job Steps From Standard Input: Say you want to run three SPMD programs – *setup*, *computation*, and *cleanup* – as job steps on the same partition. Assuming STDIN is keyboard entry, **MP_PGMMODEL** is set to *spmd*, and **MP_NEWJOB** is set to *yes*, you would:

ENTER **poe** [*poe-options*]

● The Partition Manager allocates the processor nodes of your partition, and the following prompt displays:

```
0031-503 Enter program name (or quit):
```

ENTER *setup* [*program-options*]

● The program *setup* executes on all nodes of your partition. When execution completes, the following prompt displays:

```
0031-503 Enter program name (or quit):
```

ENTER *computation* [*program-options*]

● The program *computation* executes on all nodes of your partition. When execution completes, the following prompt displays:

```
0031-503 Enter program name (or quit):
```

ENTER *cleanup* [*program-options*]

● The program *cleanup* executes on all nodes of your partition. When execution completes, the following prompt displays:

```
0031-503 Enter program name (or quit):
```

ENTER **quit**

or

<**Ctrl-d**>

<**Ctrl-d**>

● The Partition Manager releases the nodes of your partition.

Notes:

1. You can also run a series of MPMD programs in job step fashion from STDIN. If **MP_PGMMODEL** is set to *mpmd*, the Partition Manager will, after each step completes, prompt you to individually reload the partition as described in "Loading Nodes Individually From Standard Input" on page 37.
2. When **MP_NEWJOB** is *yes*, the Partition Manager, by default, looks to STDIN for job steps. However, if the environment variable **MP_CMDFILE** is set to the name of a POE commands file as described in "Reading Job Steps From a POE Commands File" on page 40, the Partition Manger will look to the commands file instead. To ensure that job steps are read from STDIN, check that the **MP_CMDFILE** environment variable is unspecified.

Multi-Step STDIN for Newjob Mode: POE's STDIN processing model allows redirected STDIN to be passed to all steps of a newjob sequence, when the redirection is from a file. If redirection is from a pipe, POE does not distribute the input to each step, only to the first step.

Reading Job Steps From a POE Commands File: The **MP_CMDFILE** environment variable, and its associated command-line flag **-cmdfile**, lets you specify the name of a POE commands file. If **MP_NEWJOB** is *yes*, you can have the Partition Manager read job steps from a POE commands file. The commands file in this case simply lists the programs you want to run as job steps. For example, say you want to run the three SPMD programs *setup*, *computation*, and *cleanup* as job steps on the same partition. Your POE commands file would contain the following three lines:

```
setup [program-options]

computation [program-options]

cleanup [program-options]
```

`Program-options` represent the actual values you need to specify.

If you are loading a series of MPMD programs, the POE commands file is also responsible for individually loading the partition. For example, say you had three master/worker MPMD job steps that you wanted to run as 4 tasks on the same partition. The following is a representation of what your POE commands file would contain. `Options` represent the actual values you need to specify.

```
master1 [options]

workers1 [options]

workers1 [options]

workers1 [options]

master2 [options]

workers2 [options]

workers2 [options]

workers2 [options]

master3 [options]

workers3 [options]

workers3 [options]

workers3 [options]
```

While you could also redirect STDIN to read job steps from a file, a POE commands file gives you more flexibility by not tying up STDIN. You can specify a POE commands file using its relative or full path name. Say your POE commands file is called */u/hinkle/jobsteps*. To specify that the Partition Manager should read job steps from this file rather than STDIN, you could:

Set the MP_CMDFILE environment variable:	**Use the -cmdfile flag on the poe command:**
ENTER export MP_CMDFILE=/u/hinkle/jobsteps	**ENTER poe -cmdfile /u/hinkle/jobsteps**

Once **MP_NEWJOB** is set to *yes*, and **MP_CMDFILE** is set to the name of your POE commands file, you would:

ENTER **poe** [*poe-options*]

● The Partition Manager allocates the processor nodes of your partition, and reads job steps from your POE commands file. The Partition Manager does not release your partition until it reaches the end of your commands file.

Invoking a Non-Parallel Program On Remote Nodes

You can also use POE to run non-parallel programs on the remote nodes of your partition. Any executable (binary file, shell script, UNIX utility) is suitable, and it does not need to have been compiled with **mpcc**, **mpCC**, or **mpxlf**. For example, if you wanted to check the process status (using the AIX command **ps**) for all remote nodes in your partition, you would:

ENTER **poe ps**

● The process status for each remote node is written to standard out (STDOUT) at your home node. How STDOUT from all the remote nodes is handled at your home node depends on the output mode. See "Managing Standard Output (STDOUT)" on page 48 for more information.

Controlling Program Execution

This section describes a number of additional POE environment variables for monitoring and controlling program execution. It describes how to use the:

- **MP_EUIDEVELOP** environment variable to specify that you want to run your program in message passing develop mode. In this mode, more detailed checking of your program is performed.
- **MP_RETRY** environment variable to make POE wait for processor nodes to become available from the Resource Manager.
- **MP_RETRYCOUNT** environment variable to specify the number of times the Partition Manager should request nodes before returning.
- **MP_NOARGLIST** and **MP_FENCE** environment variable to make POE ignore arguments.
- **MP_STDINMODE** and **MP_HOLD_STDIN** environment variables to manage standard input.
- **MP_STDOUTMODE** environment variable to manage standard output.
- **MP_INFOLEVEL** environment variable to specify the level of messages you want reported to standard error.
- **MP_PMDLOG** environment variable to generate a diagnostic log on remote nodes.
- **MP_LABELIO** environment variable to label message output with task identifiers.
- **MP_COREDIR** environment variable to create a separate directory for each task's core file.
- **MP_PULSE** environment variable to ensure that remote nodes are communicating with the home node. See “Detecting Remote Node Failures” on page 57 for more information.
- | **MP_CHECKFILE** environment variable to define the base name of the checkpoint file when checkpointing or restarting a program. See “Checkpointing and Restarting Programs” on page 51 for more information.
- | **MP_CHECKDIR** environment variable to define the directory where the checkpoint file will reside when checkpointing or restarting a program. See “Checkpointing and Restarting Programs” on page 51 for more information.

For a complete listing of all POE environment variables, see Appendix B, “POE Environment Variables and Command-Line Flags” on page 139.

Specifying Develop Mode

You can run programs in one of two modes – *develop mode* or *run mode*. In develop mode, intended for developing applications, the message passing interface performs more detailed checking during execution. Because of the additional checking it performs, develop mode can significantly slow program performance. In run mode, intended for completed applications, only minimal checking is done. While run mode is the default, you can use the **MP_EUIDEVELOP** environment variable to specify message passing develop mode. As with most POE environment variables, **MP_EUIDEVELOP** has an associated command-line flag **-euidevelop**. To specify MPI develop mode, you could:

Set the MP_EUIDEVELOP environment variable:	Use the -euidevelop flag when invoking the program:
ENTER **export MP_EUIDEVELOP=***yes*	**ENTER** **poe** *program* **-euidevelop** *yes*

To later go back to run mode, set **MP_EUIDEVELOP** to *no*.

You can also use **MP_EUIDEVELOP** for **pedb** parameter checking by specifying the *DEB* value, for "debug."

Set the MP_EUIDEVELOP environment variable:	Use the -euidevelop flag when invoking the program:
ENTER **export MP_EUIDEVELOP=***DEB*	**ENTER** **poe** *program* **-euidevelop** *DEB*

| To stop parameter checking, set **MP_EUIDEVELOP** to *min*, for "minimum."

Making POE Wait for Processor Nodes

If you are using an SP system, and there are not enough available nodes to run your program, the Partition Manager, by default, returns immediately with an error. Your program does not run. Using the **MP_RETRY** and **MP_RETRYCOUNT** environment variables, however, you can instruct the Partition Manager to repeat the node request a set number of times at set intervals. Each time the Partition Manager repeats the node request, it displays the following message:

```
Retry allocation      ......press control-C to terminate
```

The **MP_RETRY** environment variable, and its associated command-line flag **-retry**, specifies the interval (in seconds) to wait before repeating the node request. The **MP_RETRYCOUNT** environment variable, and its associated command-line flag **-retrycount**, specifies the number of times the Partition Manager should make the request before returning. For example, if you wanted to retry the node request five times at five minute (300 second) intervals, you could:

Set the MP_RETRY and MP_RETRYCOUNT environment variables:	Use the -retry and -retrycount flags when invoking the program:
ENTER **export MP_RETRY=***300* **export MP_RETRYCOUNT=***5*	**ENTER** **poe** *program* **-retry** *300* **-retrycount** *5*

Note: If the **MP_RETRYCOUNT** environment variable or the **-retrycount** command-line flag is used, the **MP_RETRY** environment variable or the **-retry** command-line flag must be set to at least one second.

Making POE Ignore Arguments

When you invoke a parallel executable, you can specify an argument list consisting of a number of program options and POE command-line flags. The argument list is parsed by POE – the POE command-line flags are removed and the remainder of the list is passed on to the program. If any of your program arguments are identical to POE command-line flags, however, this can cause problems. For example, say you have a program that takes the argument **-retry**. You invoke the program with the **-retry** option, but it does not execute correctly. This is because there is also a POE command-line flag **-retry**. POE parses the argument list and so the **-retry** option is never passed on to your program. There are two ways to correct this sort of problem. You can:

- make POE ignore the entire argument list using the **MP_NOARGLIST** environment variable.
- make POE ignore a portion of the argument list using the **MP_FENCE** environment variable.

Making POE Ignore the Entire Argument List

When you invoke a parallel executable, POE, by default, parses the argument list and removes all POE command-line flags before passing the rest of the list on to the program. Using the environment variable **MP_NOARGLIST**, you can prevent POE from parsing the argument list. To do this:

ENTER **export MP_NOARGLIST=***yes*

When the **MP_NOARGLIST** environment variable is set to *yes*, POE does not examine the argument list at all. It simply passes the entire list on to the program. For this reason, you can not use any POE command-line flags, but must use the POE environment variables exclusively. While most POE environment variables have associated command-line flags, **MP_NOARGLIST**, for obvious reasons, does not. To specify that POE should again examine argument lists, either set **MP_NOARGLIST** to *no*, or unset it.

ENTER **export MP_NOARGLIST=***no*

or

unset MP_NOARGLIST

Making POE Ignore a Portion of the Argument List

When you invoke a parallel executable, POE, by default, parses the entire argument list and removes all POE command-line flags before passing the rest of the list on to the program. You can use a fence, however, to prevent POE from parsing the remainder of the argument list. A *fence* is simply a character string you define using the **MP_FENCE** environment variable. Once defined, you can use the fence to separate those arguments you want parsed by POE from those you do not. For example, say you have a program that takes the argument **-retry**. Because there is also a POE command-line flag **-retry**, you need to put this argument after a fence. To do this, you could:

ENTER **export MP_FENCE=***Q*

poe *program* **-procs** *26* **-infolevel** *2 Q -retry RGB*

While this example defines Q as the fence, keep in mind that the fence can be any character string. Any arguments placed after the fence are passed by POE, unexamined, to the program. While most POE environment variables have associated command-line flags, **MP_FENCE** does not.

Managing Standard Input, Output, and Error

POE lets you control standard input (STDIN), standard output (STDOUT), and standard error (STDERR) in several ways. You can continue using the traditional I/O manipulation techniques such as redirection and piping, and can also:

- determine whether a single task or all parallel tasks should receive data from STDIN.

- determine whether a single task or all parallel tasks should write to STDOUT. If all tasks are writing to STDOUT, you can further define whether or not the messages are ordered by task id.
- specify the level of messages that will be reported to STDERR during program execution.
- specify that messages to STDOUT and STDERR should be labeled by task id.

Managing Standard Input (STDIN)

STDIN is the primary source of data going into a command. Usually, STDIN refers to keyboard input. If you use redirection or piping, however, STDIN could refer to a file or the output from another command (see "Using MP_HOLD_STDIN" on page 46). How you manage STDIN for a parallel application depends on whether or not its parallel tasks require the same input data. Using the environment variable **MP_STDINMODE** or the command-line flag **-stdinmode**, you can specify that:

- all tasks should receive the same input data from STDIN. This is *multiple input mode*.
- STDIN should be sent to a single task of your partition. This is *single input mode*.
- no task should receive input data from STDIN.

Multiple Input Mode: Setting **MP_STDINMODE** to *all* indicates that all tasks should receive the same input data from STDIN. The home node Partition Manager sends STDIN to each task as it is read.

To specify multiple input mode so all tasks receive the same input data from STDIN, you could:

Set the MP_STDINMODE environment variable:	**Use the -stdinmode flag when invoking the program:**
ENTER **export MP_STDINMODE=***all*	**ENTER** **poe** *program* **-stdinmode** *all*

Note: If you do not set the **MP_STDINMODE** environment variable or use the **-stdinmode** command-line flag, multiple input mode is the default.

Single Input Mode: There are times when you only want a single task to read from STDIN. To do this, you set **MP_STDINMODE** to the appropriate task id. For example, say you have an MPMD application consisting of two programs – *master* and *workers*. The program *master* is designed to run as a single task on one processor node. The *workers* program is designed to run as separate tasks on any number of other nodes. The *master* program handles all I/O, so only its task needs to read STDIN. If *master* is running as task 0, you need to specify that only task 0 should receive STDIN. To do this, you could:

Set the MP_STDINMODE environment variable:	**Use the -stdinmode flag when invoking the program:**
ENTER **export MP_STDINMODE=***0*	**ENTER** **poe** *program* **-stdinmode** *0*

Using MP_HOLD_STDIN

The environment variable **MP_HOLD_STDIN** is used to defer sending of STDIN from the home node to the remote node(s) until the message passing library has been initialized. The variable must be set to “yes” when using POE to invoke a program which: (1) has been compiled with **mpcc**, **mpxlf**, or **mpCC** and their **_r** equivalents for the threaded environment, *and* (2) will be reading STDIN from other than the keyboard (redirection or piping). Failing to export this environment variable when running these programs could likely result in the user program hanging.

In addition, if a program invoked using POE has not been compiled with **mpcc**, **mpxlf**, or **mpCC**, the environment variable must not be set (or set to “no”) to ensure that STDIN is delivered to the remote node(s).

To set **MP_HOLD_STDIN** correctly, you need to know the relative order of your program's use of stdin data and initialization of the message passing library.

The discussion immediately below applies to the signal handling message passing library (MPI/MPL), which is initialized before the user's executable gets control.

The subsequent section addresses the question for the threaded MPI library.

Using Redirected STDIN

Note: Wherever the following description refers to a POE environment variable (starting with **MP_**), the use of the associated command line option produces the same effect, with the exception of **MP_HOLD_STDIN**, which has no associated command line option.

A POE process can use its STDIN in two ways. First, if the program name is not supplied on the command line and no command file (**MP_CMDFILE**) is specified, POE uses STDIN to resolve the names of the programs to be run as the remote tasks. Second, any “remaining” STDIN is then distributed to the remote tasks as indicated by the **MP_STDINMODE** and **MP_HOLD_STDIN** settings. In this dual STDIN model, redirected STDIN can then pose two problems:

1. If using job steps (**MP_NEWJOB=yes**), the “remaining” STDIN is always consumed by the remote tasks during the first job step.
2. If POE attempts program name resolution on the redirected STDIN, program behavior can vary when using job steps, depending on the type of redirection used and the size of the redirected STDIN.

The first problem is addressed in POE by performing a rewind of STDIN between job steps (only if STDIN is redirected from a file, for reasons beyond the scope of this document). The second problem is addressed by providing an additional setting for **MP_STDINMODE** of “none,” which tells POE to only use STDIN for program name resolution. As far as STDIN is concerned, “none” ever gets delivered to the remote tasks. This provides an additional method of reliably specifying the program name to POE, by redirecting STDIN from a file or pipe, or by using the shell's here-document syntax in conjunction with the “none” setting. If **MP_STDINMODE** is not set to “none” when POE attempts program name resolution on redirected STDIN, program behavior is undefined.

The following scenarios describe in more detail the effects of using (or not using) an **MP_STDINMODE** of “none” when redirecting (or not redirecting) STDIN, as shown in the example:

		Is STDIN Redirected?	
		Yes	No
Is MP_STDINMODE set to "none"?	Yes	A	B
	No	C	D

Scenario A

POE will use the redirected STDIN for program name resolution, only if no program name is supplied on the command line (**MP_CMDFILE** is ignored when **MP_STDINMODE=none**). No STDIN is distributed to the remote tasks. No rewind of STDIN is performed when **MP_STDINMODE=none**. If **MP_HOLD_STDIN** is set to “yes,” this is ignored because no STDIN is being distributed.

Scenario B

POE will use the keyboard STDIN for program name resolution, only if no program name is supplied on the command line (**MP_CMDFILE** is ignored when **MP_STDINMODE=none**). No STDIN is distributed to the remote tasks. No rewind of STDIN is performed when **MP_STDINMODE=none** (also, STDIN is not from a file). If **MP_HOLD_STDIN** is set to “yes,” this is ignored because no STDIN is being distributed.

Scenario C

POE will use the redirected STDIN for program name resolution, if required, and will distribute “remaining” STDIN to the remote tasks. If STDIN is intended to be used for program name resolution, program behavior is *undefined* in this case, since POE was not informed of this by setting **STDINMODE** to “none” (see Problem 2 above). If STDIN is redirected from a file, POE will rewind STDIN between each job step. If **MP_HOLD_STDIN** is set to “yes,” this feature will behave accordingly.

Scenario D

POE will use the keyboard STDIN for program name resolution, if required. Any “remaining” STDIN is distributed to the remote tasks. No rewind of STDIN is performed since STDIN is not from a file. If **MP_HOLD_STDIN** is set to “yes,” it is ignored because STDIN is not redirected.

Using Redirected STDIN with the Threaded MPI Library

If the user's executable is compiled with the threaded MPI library, message passing initialization occurs when MPI_Init is called, not before POE gives the user program control. If MPI_Init is called before any STDIN data is read, the discussions of the previous section apply. If, however, all STDIN is read before MPI_Init is called, then **MP_HOLD_STDIN** should be set to “no,” to allow the STDIN data to be sent to the user's executable by POE.

Managing Standard Output (STDOUT)

STDOUT is where the data coming from the command will eventually go. Usually, STDOUT refers to the display. If you use redirection or piping, however, STDOUT could refer to a file or another command. How you manage STDOUT for a parallel application depends on whether you want output data from one task or all tasks. If all tasks are writing to STDOUT, you can also specify whether or not output is ordered by task id. Using the environment variable **MP_STDOUTMODE**, you can specify that:

- all tasks should write output data to STDOUT asynchronously. This is *unordered output mode*.
- output data from each parallel task should be written to its own buffer, and later all buffers should be flushed, in task order, to STDOUT. This is *ordered output mode*.
- a single task of your partition should write to STDOUT. This is *single output mode*.

Unordered Output Mode: Setting **MP_STDOUTMODE** to *unordered* specifies that all tasks should write output data to STDOUT asynchronously. To specify unordered output mode, you could:

Set the MP_STDOUTMODE environment variable:	Use the -stdoutmode flag when invoking the program:
ENTER **export MP_STDOUTMODE=***unordered*	**ENTER** **poe** *program* **-stdoutmode** *unordered*

Notes:

1. If you do not set the **MP_STDOUTMODE** environment variable or use the **-stdoutmode** command-line flag, unordered output mode is the default.
2. If you are using unordered output mode, you will probably want the messages labeled by task id. Otherwise it will be difficult to know which task sent which message. See "Labeling Message Output" on page 49 for more information.
3. You can also specify unordered output mode from your program by calling the MP_STDOUTMODE or mpc_stdoutmode Parallel Utility Function. Refer to *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference* for more information.
4. Although the above environment variable and Parallel Utility Function are both described as "MP_STDOUTMODE," they are each used independently for their specific purposes.

Ordered Output Mode: Setting **MP_STDOUTMODE** to *ordered* specifies ordered output mode. In this mode, each task writes output data to its own buffer. Later, all the task buffers are flushed, in order of task id, to STDOUT. The buffers are flushed when:

- any one of the individual task buffers fills
- execution of the program completes.
- all tasks explicitly flush the buffers by calling the MP_FLUSH or mpc_flush Parallel Utility Function.
- tasks change output mode using calls to Parallel Utility Functions. For more information on Parallel Utility Functions, refer to *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*

Note: When running the parallel application under **pdbx** with **MP_STDOUTMODE** set to ordered, there will be a difference in the ordering from when the application is run directly under **poe**. The buffer size available for the application's STDOUT is smaller because **pdbx** uses some of the buffer, so the task buffers fill up more often.

To specify ordered output mode, you could:

Set the MP_STDOUTMODE environment variable:	Use the -stdoutmode flag when invoking the program:
ENTER **export MP_STDOUTMODE=***ordered*	**ENTER** **poe** *program* **-stdoutmode** *ordered*

Note: You can also specify ordered output mode from your program by calling the MP_STDOUTMODE or mpc_stdoutmode Parallel Utility Function. Refer to *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference* for more information.

Single Output Mode: You can specify that only one task should write its output data to STDOUT. To do this, you set **MP_STDOUTMODE** to the appropriate task id. For example, say you have an SPMD application in which all the parallel tasks are sending the exact same output messages. For easier readability, you would prefer output from only one task – task 0. To specify this, you could:

Set the MP_STDOUTMODE environment variable:	Use the -stdoutmode flag when invoking the program:
ENTER **export MP_STDOUTMODE=***0*	**ENTER** **poe** *program* **-stdoutmode** *0*

Note: You can also specify single output mode from your program by calling the MP_STDOUTMODE or mpc_stdoutmode Parallel Utility Function. Refer to *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference* for more information.

Labeling Message Output

You can set the environment variable **MP_LABELIO**, or use the **-labelio** flag when invoking a program, so that output from the parallel tasks of your program are labeled by task id. While not necessary when output is being generated in *single* mode, this ability can be useful in *ordered* and *unordered* modes. For example, say the output mode is *unordered*. You are executing a program and receiving asynchronous output messages from all the tasks. This output is not labeled, so you do not know which task has sent which message. It would be clearer if the unordered output was labeled. For example:

```
  7: Hello World

  0: Hello World

  3: Hello World

 23: Hello World

 14: Hello World

  9: Hello World
```

To have the messages labeled with the appropriate task id, you could:

Set the MP_LABELIO environment variable:	**Use the -labelio flag when invoking the program:**
ENTER **export MP_LABELIO=***yes*	**ENTER** **poe** *program* **-labelio** *yes*

To no longer have message output labeled, set the **MP_LABELIO** environment variable to *no*.

Setting the Message Reporting Level for Standard Error (STDERR)

You can set the environment variable **MP_INFOLEVEL** to specify the level of messages you want from POE. You can set the value of **MP_INFOLEVEL** to one of the integers shown in the following table. The integers *0*, *1*, and *2* give you different levels of informational, warning, and error messages. The integers *3* through *6* indicate debug levels that provide additional debugging and diagnostic information. Should you require help from the IBM Support Center in resolving a PE-related problem, you will probably be asked to run with one of the debug levels. As with most POE environment variables, you can override **MP_INFOLEVEL** when you invoke a program. This is done using either the **-infolevel** or **-ilevel** flag followed by the appropriate integer.

This integer:	**Indicates this level of message reporting:**	**In other words:**
0	Error	Only error messages from POE are written to STDERR.
1	Normal	Warning and error messages from POE are written to STDERR. This level of message reporting is the default.
2	Verbose	Informational, warning, and error messages from POE are written to STDERR.
3	Debug Level 1	Informational, warning, and error messages from POE are written to STDERR. Also written is some high-level debugging and diagnostic information.
4	Debug Level 2	Informational, warning, and error messages from POE are written to STDERR. Also written is some high- and low-level debugging and diagnostic information.
5	Debug Level 3	Debug level 2 messages plus some additional loop detail.
6	Debug Level 4	Debug level 3 messages plus other informational error messages for the greatest amount of diagnostic information.

Let us say you want the POE message level set to verbose. The following table shows the two ways to do this. You could:

Set the MP_INFOLEVEL environment variable:	**Use the -infolevel flag when invoking the program:**
ENTER **export MP_INFOLEVEL**=*2*	**ENTER** **poe** *program* **-infolevel** *2* or **poe** *program* **-ilevel** *2*

As with most POE command-line flags, the **-infolevel** or **-ilevel** flag temporarily override their associated environment variable.

Generating a Diagnostic Log on Remote Nodes

Using the **MP_PMDLOG** environment variable, you can also specify that diagnostic messages should be logged to a file in */tmp* on each of the remote nodes of your partition. The log file is named *mplog.pid.n*, where *pid* is the AIX process id of the Partition Manager Daemon, and *n* is the task number. Should you require help from the IBM Support Center in resolving a PE-related problem, you will probably be asked to generate these diagnostic logs.

The ability to generate diagnostic logs on each node is particularly useful for isolating the cause of abnormal termination, especially when the connection between the remote node and the home node Partition Manager has been broken. As with most POE environment variables, you can temporarily override the value of **MP_PMDLOG** using its associated command-line flag **-pmdlog**. For example, to generate a **pmd** log file, you could:

Set the MP_PMDLOG environment variable:	Use the -pmdlog flag when invoking the program:
ENTER **export MP_PMDLOG=***yes*	**ENTER** **poe** *program* **-pmdlog** *yes*

Note: By default, **MP_PMDLOG** is set to *no*. No diagnostic logs are generated. It is not suggested that you run **MP_PMDLOG** routinely. Running this will greatly impact performance and fill up your file system space.

Checkpointing and Restarting Programs

You can set the environment variables **MP_CHECKFILE** and **MP_CHECKDIR** to checkpoint or restart a program, which was previously compiled with the **mpcc_chkpt**, **mpCC_chkpt**, or **mpxlf_chkpt** commands. Only POE/MPI applications submitted under LoadLeveler in batch mode are able to be checkpointed. Checkpointing of interactive POE applications is not allowed.

The program's execution will be suspended when the **mp_chkpt()** function is reached. See *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference* for the description of the **mp_chkpt** function. At that point, the state of the application is captured, along with all data, and saved to the file pointed to by the **MP_CHECKFILE** and **MP_CHECKDIR** variables.

MP_CHECKFILE defines the base name of the checkpoint file. **MP_CHECKDIR** defines the directory where the checkpoint file will reside. If the **MP_CHECKFILE** variable is not specified, the program cannot be checkpointed. The file name specified by **MP_CHECKFILE** may include the full path, in which case the **MP_CHECKDIR** variable will be ignored. If **MP_CHECKDIR** is not defined and **MP_CHECKFILE** does not specify a full path name, then **MP_CHECKFILE** is used as a relative path name from the current working directory.

Only programs compiled with the checkpoint compile scripts (**mpcc_chkpt**, **mpCC_chkpt**, or **mpxlf_chkpt**) that call the **mp_chkpt** function can be checkpointed.

When the checkpoint file is created during the checkpointing phase, the task id and a version id are appended to the base file name to differentiate between checkpoint files from different instances of the program.

There are certain limitations associated with checkpointing an application. Please refer to *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference* for specific details.

Restarting a Checkpointed Program

A program can be restarted by executing POE, using **MP_CHECKFILE** and **MP_CHECKDIR** to point to the checkpoint file from the previously checkpointed program. The checkpoint file must be valid and accessible to all tasks specific when invoking POE. The application can be restarted on the same or a different set of nodes. However, the number of tasks must remain the same.

During the restart processing, the version and content of the checkpoint file are verified internally by POE to ensure consistency and accuracy. Any discrepancies, such as a mismatch in versions of the program files, will be reported. That is, the versions of the specified checkpoint files are not the same across all tasks.

The checkpoint file will be read, and the program will be restored to an executing state, after retrieving the program state and data information from the file. When execution is completely restored, the checkpoint files are deleted.

If you are using the **MP_BUFFER_MEM** environment variable to change the maximum size of memory used by the communication subsystem while checkpointing a program, please be aware that the amount of space needed for the checkpointing files will be increased by the value of **MP_BUFFER_MEM**.

Checkpointing File Management

The ability to checkpoint or restart programs is controlled by the definition and availability of the checkpoint files, as specified by the **MP_CHECKFILE** environment variable.

The specified file may be defined on the local file system (JFS) of the node on which the instance of the program is running, or it may be defined in some shared file system (such as NFS, AFS, DFS, GPFS, etc.). When the file is in a local file system, then in order to perform process migration, the checkpoint file will have to be moved to the new system on which the process is to be restarted. If the old system crashed and is unavailable, it may not be possible to restart the program. It may be necessary, therefore, to use some kind of file management to avoid such a problem. If migration is not desired, it is sufficient to place checkpoint files in the local JFS file system.

The program checkpoint files can be large, and numerous. There is the potential need for significant amounts of available disk space to maintain the files. It is recommended that you do **not** use NFS, AFS, or DFS for managing checkpoint files. The nature of these systems is such that it takes a very long time to write and read large files. The use of GPFS or JFS is recommended.

If a local JFS file system is used, the checkpoint file must be written to each remote task's local file system during checkpointing. Consequently, during a restart, each remote task's local file system must be able to access the checkpoint file from the previously checkpointed program. This is of special concern when opting to restart a program on a different set of nodes from which it was checkpointed. The local checkpoint file may need to be relocated to any new nodes. For these reasons, it is suggested that GPFS be the file system best suited for checkpoint and restart file management.

Running POE within a Distributed File System

This section gives you instructions on how to run POE within a Distributed File System (DFS). Included is a description of the **poeauth** command, which allows you to copy DFS credentials to all nodes on which you want to run POE jobs.

| **Note:** When running POE under LoadLeveler, LoadLeveler handles all user
| authorization instead of POE.

Setting Up Your System to Run POE

In order to run POE jobs from DFS, you will need to copy the DFS/DCE credentials files to each node you wish to run on, using the **poeauth** command. You should be set up with a DFS account, and after you login, you access your DCE user credentials by doing a **dce_login**.

DCE credentials are defined on a per user basis, therefore each user must use **poeauth** to copy the credentials prior to running a POE job on a DFS/DCE system.

To be able to run the **poeauth** command, you should make some initial file and directory changes. In your pool or host list file you define all nodes on which you want to run POE jobs. You change directories to a local non-DFS file system, for example, */tmp*. The **poeauth** command sets up the DFS credentials for POE, therefore you cannot be in a DFS directory (as the current directory) to run it.

The execution of the **poeauth** command is dependent upon the type of user authorization specified by the **MP_AUTH** environment variable – either AIX or DFS/DCE authorization.

When AIX user authorization is selected (either by setting **MP_AUTH=AIX** or allowing it as the default), and your home directory resides in DFS, your user name must be properly authorized to access those nodes in the */etc/hosts.equiv* file on each node. You should remove all entries from the *.rhosts* files on each node, and allow the */etc/hosts.equiv* file to authorize the users on each node. Otherwise, POE will not be able to authorize users properly. Once DFS credentials are established, you can use a *.rhosts* file.

The **dce_login** sets up a new shell. As a result, you should set up any environment variables needed to run **poeauth** or other POE applications (such as **MP_AUTH**) after doing the **dce_login**.

Running the poeauth Command

You should run the **poeauth** command from task 0 that had a **dce_login**. You can use any POE command line flag or environment variable with **poeauth**, because it is a POE application. Each user must run **poeauth** before running any POE applications. When the credentials are copied, there is no need to use **poeauth** until the credentials expire (at which time you will need to copy them using **poeauth** again). After you run the **poeauth** command successfully, you can run POE from DFS. For more information on the **poeauth** command, see Appendix A, “Parallel Environment Commands” on page 83.

Note: Credentials files need to exist on the home node (task 0), that is, from where **dce_login** was performed. The **poeauth** command needs to be run from task 0.

Checking for Errors

When POE returns error messages related to an inability to change to a DFS directory or a problem copying a file to a DFS directory, it most likely means there is a problem with the DFS credentials on that task or node. Check to see if the credentials were properly copied with **poeauth**, or if they have expired (use the **klist** command).

Since **poeauth** is a POE application, if you try to run it when the credentials have expired, POE will encounter an error accessing the expired credentials.

If the credentials have expired, you must do another **dce_login** and run the **poeauth** command again.

POE maintains a master control file in */tmp* to keep track of the credentials. If */tmp* is periodically cleaned out or the file is accidentally erased before your credentials expire, POE will not be able to access your DCE credentials and you may get errors related to the inability to access credentials. If this occurs, you will need to run the **poeauth** command again to redefine your credentials to POE.

Chapter 3. Managing POE Jobs

This chapter describes the tasks involved with managing POE jobs. It includes the following:

| - Scenarios for allocating nodes with LoadLeveler
- Scenarios for allocating nodes with the Resource Manager
- Information about IBM LoadLeveler support
- Appropriate environment variable information to use when running your applications.

Multi-Task Core File

With the **MP_COREDIR** environment variable, you can create a separate directory to save a core file for each task. The corresponding command line option is **-coredir**. Creating this type of directory is useful when you are running a parallel job on one node, and the job dumps a core file. By checking the directory, you can see which task dumped the file. When setting **MP_COREDIR**, you specify the first attribute of the directory name. The second attribute is the task id. If you do not specify a directory, the default is *coredir*. The subdirectory containing each task's core file is named *coredir.taskid*. The following examples show what happens when you set the environment variable:

```
Example 1:


MP_COREDIR=my_parallel_cores

MP_PROCS=2


run generates core files


Core files will be located at:


/current directory/my_parallel_cores.0/core

/current directory/my_parallel_cores.1/core


Example 2:


MP_COREDIR not specified

MP_PROCS=2


run generates core files


Core files will be located at:


/current directory/coredir.0/core

/current directory/coredir.1/core
```

Stopping a POE Job

You can stop (suspend) a POE job by pressing <**Ctrl-z**> or by sending POE a SIGTSTP signal. POE stops, and sends a SIGSTOP signal to all the remote tasks, which stops them. To resume the parallel job, issue the **fg** or **bg** command to POE. A SIGCONT signal will be sent to all the remote tasks to resume them.

Cancelling and Killing a POE Job

You can cancel a POE job by pressing <**Ctrl-c**> or <**Ctrl-**>. This sends POE a SIGINT or SIGQUIT signal respectively. POE terminates all the remote tasks, completes the generation of the VT trace file, and exits.

If POE is killed or terminated before the remote nodes are shut down, direct communication with the parallel job will be lost. In this situation, use the **poekill** script as a POE command, or individually via **rsh**, to terminate the partition. **poekill** kills all instantiations of the program name on a remote node by sending it a SIGTERM signal. See the **poekill** script in */usr/lpp/ppe.poe/bin*, and the description of the **poekill** command in Appendix A, “Parallel Environment Commands” on page 83.

Note: *Do not* kill the **pmd**s using the **poekill** command. This will ensure that your remote processes will continue running.

Detecting Remote Node Failures

POE and the Partition Manager use a *pulse* detection mechanism to periodically check each remote node to ensure that it is actively communicating with the home node. You specify the time interval (or *pulse* interval), of these checks with the **-pulse** flag or the **MP_PULSE** environment variable. When POE starts a parallel job, During an execution of a POE job, POE and the Partition Manager daemons check at the interval you specify that each node is running. When a node failure is detected, POE terminates the job on all remaining nodes and issues an error message.

The default pulse interval is 600 seconds (10 minutes). You can increase or decrease this value with the **-pulse** flag or the **MP_PULSE** environment variable. To completely disable the pulse function, specify an interval value of 0 (zero). For the PE debugging facility **MP_PULSE** is disabled.

Considerations for Using the SP Switch

The SP switch supports dedicated User Space (US) and IP sessions, running concurrently on a single node. Users of IP communication programs that are not using a job management system (LoadLeveler or the Resource Manager), may treat this adapter like any other IP-supporting adapter. In this case, the adapter name is *css0*.

While US message passing programs must use a job management system to allocate nodes, IP message passing programs may use a job management system, but are not required to. When using LoadLeveler, nodes may be requested by name or number from one system pool only. When using the Resource Manager, nodes may be requested by number or by specifying one or more node pools to be used. When specifying node pools, the following rules apply:

- All the nodes in a pool should support the same combination of IP and US protocols. In other words, all the nodes should be able to run:
  - the IP protocol

    or
  - the US protocol

  or
  – the IP and US protocols concurrently.

- In order to run the IP protocol, the IP switch addresses must be configured and started. In order to run the US protocol, the switch node numbers must be configured. For more information regarding these protocols and LoadLeveler, see *Using and Administering LoadLeveler* . For more information regarding these protocols and the Resource Manager, see *IBM Parallel System Support Programs for AIX: Installation and Migration Guide* .
- By default, requests for the US message passing protocol also request exclusive use of the node; the job management system will not allocate concurrent IP message passing programs on this node. You can override this default so that the node can be used for both IP and US programs by specifying “multiple” CPU usage.
- By default, requests for the IP message passing protocol also request multiple use of the node; the job management system can allocate both IP and US message passing programs on this node. You can override this default so that the node is designated for exclusive use by specifying “unique” CPU usage.
- When running a batch parallel program under LoadLeveler, the adapter and CPU are allocated as specified by the *network keyword* in the LoadLeveler Job Command File. See *Using and Administering LoadLeveler* for more information.

Scenarios for Allocating Nodes With LoadLeveler

This section provides some examples of how someone would allocate nodes using LoadLeveler.

Scenario 1: Explicit Allocation

A POE user, Paul, wishes to run a US job 1 in nodes A, B, C, and D. He doesn't mind sharing the node with other jobs, as long as they are not also running in US. To do this, he specifies MP_EUIDEVICE=css0, MP_EUILIB=us, MP_PROCS=4, MP_CPU_USE=multiple, and MP_ADAPTER_USE=dedicated. In his host file, he also specifies:

```
node_A
node_B
node_C
node_D
```

The POE Partition Manager (PM) sees that this is a US job, and asks LoadLeveler for dedicated use of the css0 adapter on nodes A, B, C, and D and shared use of the CPU on those nodes. LoadLeveler then allocates the nodes to the job, recording that the css0/US session on A, B, C, and D has been reserved for dedicated use by this job, but that the node may also be shared by other users.

While job 1 is running, another POE user, Dan, wants to run another US job, job 2, on nodes B and C, and is willing to share the nodes with other users. He specifies MP_EUIDEVICE=css0, MP_EUILIB=us, and MP_PROCS=2, MP_CPU_USE=multiple, and MP_ADAPTER_USE=dedicated. In his host file, he also specifies:

```
node_B
node_C
```

The PM, as before, asks LoadLeveler for dedicated use of the css0/US adapter on nodes B and C. LoadLeveler determines that this adapter has already been reserved for dedicated use on nodes B and C, and does not allocate the nodes again to job 2. The allocation fails, and POE job 2 cannot run.

While job 1 is running, a second POE user, John, wishes to run IP/switch job 3 on nodes A, B, C, and D, but doesn't mind sharing the node and the SP switch with other users. He specifies MP_EUIDEVICE=css0, MP_EUILIB=ip, MP_PROCS=4, MP_CPU_USE=multiple, and MP_ADAPTER_USE=shared. In his host file, he also specifies;

```
node_A
node_B
node_C
node_D
```

The POE PM asks LoadLeveler, as requested by John, for shared use of the css0/ip adapter and CPU on nodes A, B, C, and D. LoadLeveler determines that job 1 permitted other jobs to run on those nodes as long as they did not use the css0/US session on them. The allocation succeeds, and POE IP/switch job 3 runs concurrently with POE US job 1 on A, B, C, and D.

The scenario above, illustrates a situation in which users do not mind sharing nodes with other users' jobs. If a user wants his POE job to have dedicated access to nodes or the css0 adapter on nodes, he would indicate that in the environment by setting MP_CPU_USE=unique instead of *multiple*. If job 1 had done that, then job 3 would not have been allocated to those nodes and, therefore, would not have been able to run.

Scenario 2: Implicit Allocation

In this scenario, all nodes have both css0/US and css0/IP sessions configured, and are assigned to pool 2.

In this example, we have eight nodes; A, B, C, D, E, F, G, H.

Job 1: Job1 is interactive, and requests 4 nodes for US using MP_RMPOOL.

```
MP_PROCS=4

MP_RMPOOL=2

MP_EUILIB=us
```

LoadLeveler allocates nodes A, B, C, and D for dedicated adapter (forced for US) and dedicated CPU (default for MP_RMPOOL).

Job 2: Job 2 is interactive, and requests six nodes for US using host.list.

```
MP_PROCS=6

MP_HOSTFILE=./host.list

MP_EUILIB=us

MP_CPU_USE=multiple
MP_ADAPTER_USE=shared
host.list

     @2
```

POE forces the adapter request to be dedicated, even though the user specified shared. Multiple (shared CPU) is supported, but in this case LoadLeveler doesn't have six nodes, either for CPU or for adapter, so the job fails.

Job 3: Job 3 is interactive and requests six nodes for IP using MP_RMPOOL.

```
MP_PROCS=6

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU, but LoadLeveler only has four nodes available for CPU use, so the job fails.

Job 4: Job 4 is interactive and requests three nodes for IP using MP_RMPOOL.

```
MP_PROCS=3

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU. LoadLeveler allocates nodes E, F, and G.

Job 5: Job 5 is interactive and requests two nodes for IP using MP_RMPOOL.

```
MP_PROCS=2

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU. LoadLeveler allocates two nodes from the list E, F, G, H (the others are assigned as dedicated to job 1).

Scenario 3: Implicit Allocation

In this scenario, all nodes have both css0/US and css0/IP sessions configured, and are assigned to pool 2.

In this example, we have eight nodes; A, B, C, D, E, F, G, H

Job 1: Job 1 is interactive and requests four nodes for US using host.list.

```
MP_PROCS=4

MP_HOSTFILE=./host.list

MP_EUILIB=us

MP_CPU_USE=multiple
MP_ADAPTER_USE=dedicated
host.list

     @2
```

LoadLeveler allocates nodes A, B, C, and D for dedicated adapter (forced for US), and shared CPU.

Job 2: Job 2 is interactive and requests six nodes for US using host.list.

```
MP_PROCS=6

MP_HOSTFILE=./host.list

MP_EUILIB=us

MP_CPU_USE=multiple
MP_ADAPTER_USE=shared
host.list

     @2
```

POE forces the adapter request to be dedicated, even though the user has specified shared. Multiple (shared CPU) is supported, but in this case, LoadLeveler doesn't have six nodes for the adapter request, so the job fails.

Job 3: Job 3 is interactive and requests six nodes for IP using MP_RMPOOL.

```
MP_PROCS=6

MP_HOSTFILE=NULL

MP_EUILIB=ip

MP_RMPOOL=2
```

The defaults are shared adapter and shared CPU. LoadLeveler allocates six nodes for IP from the pool.

Job 4: Job 4 is interactive and requests three nodes for IP using MP_RMPOOL.

```
MP_PROCS=3

MP_HOSTFILE=NULL

MP_EUILIB=ip

MP_RMPOOL=2
```

The defaults are shared adapter and shared CPU. LoadLeveler allocates three nodes from the pool.

Scenarios for Allocating Nodes With the Resource Manager

This section provides some examples of how someone would allocate nodes using the Resource Manager.

Scenario 1: Explicit Allocation

A POE user, Paul, wishes to run a US job 1 in nodes A, B, C, and D. He doesn't mind sharing the node with other jobs, as long as they are not also running in US. To do this, he specifies MP_EUIDEVICE=css0, MP_EUILIB=us, and MP_PROCS=4. In his host file, he also specifies:

```
node_A dedicated multiple

node_B dedicated multiple

node_C dedicated multiple

node_D dedicated multiple
```

The POE Partition Manager (PM) sees that this is a US job, and asks the RM for dedicated use of the css0 adapter on nodes A, B, C, and D (regardless of whether you specify *dedicated* or *shared* in the host file), and shared use of the CPU on those nodes. The RM then allocates the nodes to the job, recording that the css0/US session on A, B, C, and D has been reserved for dedicated use by this job, but that the node may also be shared by other users.

While job 1 is running, another POE user, Dan, wants to run another US job, job 2, on nodes B and C, and is willing to share the nodes with other users. He specifies MP_EUIDEVICE=css0, MP_EUILIB=us, and MP_PROCS=2. In his host file, he also specifies:

```
node_B dedicated multiple

node_C dedicated multiple
```

The PM, as before, asks the RM for dedicated use of the css0/US adapter on nodes B and C. The RM determines that this adapter has already been reserved for dedicated use on nodes B and C, and does not allocate the nodes again to job 2. The allocation fails, and POE job 2 cannot run.

While job 1 is running, a second POE user, John, wishes to run IP/switch job 3 on nodes A, B, C, and D, but doesn't mind sharing the node and the High Performance Communication Adapter with other users. He specifies MP_EUIDEVICE=css0, MP_EUILIB=ip, MP_PROCS=4. In his host file, he also specifies;

```
node_A shared multiple

node_B shared multiple

node_C shared multiple

node_D shared multiple
```

The POE PM asks the RM, as requested by John, for shared use of the css0/ip adapter and CPU on nodes A, B, C, and D. The RM determines that job 1 permitted other jobs to run on those nodes as long as they did not use the css0/US session on them. The allocation succeeds, and POE IP/switch job 3 runs concurrently with POE US job 1 on A, B, C, and D.

The scenario above, illustrates a situation in which users do not mind sharing nodes with other users' jobs. If a user wants his POE job to have dedicated access to nodes or the css0 adapter on nodes, he would indicate that in the host file by specifying *unique* instead of *multiple*. If job 1 had done that, then job 3 would not have been allocated to those nodes and, therefore, would not have been able to run.

Scenario 2: Implicit Allocation

In this scenario, all nodes have both css0/US and css0/IP sessions configured, and are assigned to pool 2.

In this example, we have eight nodes; A, B, C, D, E, F, G, H.

Job 1: Job1 is interactive, and requests 4 nodes for US using MP_RMPOOL.

```
MP_PROCS=4

MP_RMPOOL=2

MP_EUILIB=us
```

The RM allocates nodes A, B, C, and D for dedicated adapter (forced for US) and dedicated CPU (default for MP_RMPOOL).

Job 2: Job 2 is interactive, and requests six nodes for US using host.list.

```
MP_PROCS=6

MP_HOSTFILE=./host.list

MP_EUILIB=us

host.list

     @2 shared multiple
```

POE forces the adapter request to be dedicated, even though the user specified shared. Multiple (shared CPU) is supported, but in this case the RM doesn't have six nodes, either for CPU or for adapter, so the job fails.

Job 3: Job 3 is interactive and requests six nodes for IP using MP_RMPOOL.

```
MP_PROCS=6

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU, but the RM only has four nodes available for CPU use, so the job fails.

Job 4: Job 4 is interactive and requests three nodes for IP using MP_RMPOOL.

```
MP_PROCS=3

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU. The RM allocates nodes E, F, and G.

Job 5: Job 5 is interactive and requests two nodes for IP using MP_RMPOOL.

```
MP_PROCS=2

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU. The RM allocates two nodes from the list E, F, G, H (the others are assigned as dedicated to job 1).

Scenario 3: Implicit Allocation

In this scenario, all nodes have both css0/US and css0/IP sessions configured, and are assigned to pool 2.

In this example, we have eight nodes; A, B, C, D, E, F, G, H

Job 1: Job 1 is interactive and requests four nodes for US using host.list.

```
MP_PROCS=4

MP_HOSTFILE=./host.list

MP_EUILIB=us

host.list

     @2 dedicated multiple
```

The RM allocates nodes A, B, C, and D for dedicated adapter (forced for US), and shared CPU.

Job 2: Job 2 is interactive and requests six nodes for US using host.list.

```
MP_PROCS=6

MP_HOSTFILE=./host.list

MP_EUILIB=us

host.list

     @2 shared multiple
```

POE forces the adapter request to be dedicated, even though the user has specified shared. Multiple (shared CPU) is supported, but in this case, the RM doesn't have six nodes for the adapter request, so the job fails.

Job 3: Job 3 is interactive and requests six nodes for IP using MP_RMPOOL.

```
MP_PROCS=6

MP_HOSTFILE=NULL

MP_EUILIB=ip

MP_RMPOOL=2
```

The defaults are shared adapter and shared CPU. The RM allocates six nodes for IP from the pool. There is no attempt to load balance with Job 1.

Job 4: Job 4 is interactive and requests three nodes for IP using MP_RMPOOL.

```
MP_PROCS=3

MP_HOSTFILE=NULL

MP_EUILIB=ip

MP_RMPOOL=2
```

The defaults are shared adapter and shared CPU. The RM allocates three nodes from the pool. There is no attempt to load balance with jobs 1 and 3.

Submitting a Batch POE Job using IBM LoadLeveler

Note: POE version 2.4.0 is only compatible with LoadLeveler version 2.1.0. Submitting a POE version 2.4.0 batch job with an earlier version of LoadLeveler is not supported.

This section is intended for users who wish to submit batch POE jobs using IBM LoadLeveler, version 2.1.0. Refer to *Using and Administering LoadLeveler* for more information on using this job management system.

To submit a POE job using LoadLeveler, you need to build a LoadLeveler job file, which specifies:

- The number of nodes to be allocated
- Any POE options, passed via environment variables using Loadleveler's *environment* keyword, or passed as command line options using LoadLeveler's *argument* keyword.
- The path to your POE executable (usually /usr/bin/poe).
- Adapter specifications using the network keyword.

The following POE environment variables, or associated command line options, are validated, but not used, for batch jobs submitted using LoadLeveler .

- **MP_PROCS**
- **MP_RMPOOL**
- **MP_EUIDEVICE**
- **MP_HOSTFILE**
- **MP_SAVEHOSTFILE**
- **MP_PMDSUFFIX**
- **MP_RESD**
- **MP_RETRY**
- **MP_RETRYCOUNT**
- **MP_ADAPTER_USE**
- **MP_CPU_USE**
- **MP_NODES**
- **MP_TASKS_PER_NODE**

To run **myprog** on five nodes, using a Token ring adapter for IP message passing, with the message level set to the **info** threshold, you could use the following LoadLeveler job file. The arguments **myarg1** and **myarg2** are to be passed to **myprog**.

```
#!/bin/ksh

# @ input = myjob.in

# @ output = myjob.out

# @ error = myjob.error

# @ environment = COPY_ALL; \

    MP_EUILIB=ip; \

    MP_INFO_LEVEL=2

# @ executable = /usr/bin/poe

# @ arguments = myprog myarg1 myarg2

# @ min_processors = 5

# @ requirements = (Adapter == "tokenring")

# @ job_type = parallel

# @ checkpoint = no
```

To run **myprog** on 12 nodes from pool 2, using the User Space message passing interface with the message threshold set to **warning**, you could use the following LoadLeveler job file. See the documentation provided with the LoadLeveler program product for more information.

```
#!/bin/ksh

# @ input = myusjob.in

# @ output = myusjob.out

# @ error = myusjob.error

# @ environment = COPY_ALL; MP_EUILIB=us

# @ executable = /usr/bin/poe

# @ arguments = myprog -infolevel 1

# @ min_processors = 12

# @ requirements = (Pool == 2) && (Adapter == "hps_user")

# @ job_type = parallel

# @ checkpoint = no
```

Notes:

1. If you are using the POE dynamically linked message passing interface support, you must set the **MP_EUILIB** environment variable or the **-euilib** command line option.
2. The first token of the *arguments* string in the LoadLeveler job file must be the name of the program to be run under POE, unless:
   - You use the **MP_CMDFILE** environment variable or the **-cmdfile** command line option
   - The file you specify with the keyword *input* contains the name(s) of the programs to be run under POE.
3. When setting the environment string, make sure that no white space characters follow the backslash, and that there is a space in between the semicolon and backslash.
4. When LoadLeveler allocates nodes for parallel execution, POE and task 0 will be executed on the same node.
5. When LoadLeveler detects a condition that should terminate the parallel job, a SIGTERM will be sent to POE. POE will then send the SIGTERM to each parallel task in the partition. If this signal is caught or ignored by a parallel task, LoadLeveler will ultimately terminate the task.
6. Programs that call the **usrinfo** function with the **getinfo** parameter, or programs that use the **getinfo** function, are not guaranteed to receive correct information about the owner of the current process.
7. Programs that use LAPI and also the LoadLeveler **requirements** keyword to specify **Adapter="hps_user"**, must set the **MP_MSG_API** environment variable or associated command line option accordingly.
8. If the value of the **MP_EUILIB**, **MP_EUIDEVICE**, or **MP_MSG_API** environment variables that is passed as an argument to POE differs from the specification in the network statement of the job command file, the network specification will be used, and a warning message will be printed.

Running Programs Under the C Shell

During normal configuration of an SP system, the Automount Daemon (amd) is used to mount user directories. amd's maps use the symbolic file system links, rather than the physical file system links. While the Korn shell keeps track of file system changes, so that a directory is always available, this mapping does not take place in the C shell. This is because the C shell only maintains the physical file system links. As a result, users that run POE from a C shell may find that their current directory (for example /a/moms/fileserver/sis), is not known to amd, and POE fails with message 0031-214 (unable to change directory).

By default, POE uses the Korn shell **pwd** command to obtain the name of the current directory. This works for C shell users if the current directory is either:

- The home directory
- Not mounted by amd.

If neither of the above are true (for example, if the user's current directory is a subdirectory of the home directory), then POE provides another mechanism to determine the correct amd name; the **MP_REMOTEDIR** environment variable.

POE recognizes the **MP_REMOTEDIR** environment variable as the name of a command or Korn shell script that echoes a fully-qualified file name. **MP_REMOTEDIR** is run from the current directory from which POE is started.

If you do not set **MP_REMOTEDIR**, the command defaults to **pwd**, and is run as **ksh -c pwd**. POE sends the output of this command to the remote nodes and uses it as the current directory name.

You can set **MP_REMOTEDIR** to some other value and then export it. For example, if you set **MP_REMOTEDIR="echo /tmp"**, the current directory on the remote nodes becomes /tmp on that node, regardless of what it is on the home node.

The script **mpamddir** is also provided in */usr/lpp/ppe.poe/bin*, and the setting **MP_REMOTEDIR=mpamddir** will run it. This script determines whether or not the current directory is a mounted file system. If it is, the script searches the amd maps for this directory, and constructs a name for the directory that is known to amd. You can modify this script or create additional ones that apply to your installation.

Note: Programs that depend upon the name of the current directory for correct operation may not function properly with an alternate directory name. In this case, you should carefully evaluate how to provide an appropriate name for the current directory on the home nodes.

If you are executing from a subdirectory of your home directory, and your home directory is a mounted file system, it may be sufficient to replace the C shell name of the mounted file system with the contents of $HOME. One approach would be:

```
export MP_REMOTEDIR=pwd.csh
```

or for C shell users:

```
setenv MP_REMOTEDIR pwd.csh
```

where the file **pwd.csh** is:

```
#!/bin/csh -fe

# save the current working directory name

set oldpwd = `pwd`

# get the name of the home directory

cd $HOME

set hmpwd = `pwd`

# replace the home directory prefix with the contents of $HOME

set sed_home = `echo $HOME | sed 's/\//\\\//g'`

set sed_hmpwd = `echo $hmpwd | sed 's/\//\\\//g'`

set newpwd = `echo $oldpwd | sed "s/$sed_hmpwd/$sed_home/"`

# echo the result to be used by amd

echo $newpwd
```

Using MP_CSS_INTERRUPT

The **MP_CSS_INTERRUPT** environment variable may take the value of either **yes** or **no**. By default it is set to **no**. In certain applications, setting this value to **yes** will provide improved performance.

The following briefly summarizes some general application characteristics that could potentially benefit from setting **MP_CSS_INTERRUPT=yes**.

Applications which have the following characteristics may see performance improvements from setting the POE environment variable **MP_CSS_INTERRUPT** to **yes**:

- Applications that use nonblocking send or receive operations for communication.
- Applications that have non-synchronized sets of send or receive pairs. In other words, the send from node0 is issued at a different point in time with respect to the matching receive in node1.
- Applications that do not issue waits for nonblocking send or receive operations immediately after the send or receive, but rather do some computation prior to issuing the waits.

In all of the above cases, the application is taking advantage of the asynchronous nature of the nonblocking communication subroutines. This essentially means that the calls to the nonblocking send or receive routines do not actually ensure the transmission of data from one node to the next, but only post the send or receive and then return immediately back to the user application for continued processing. However, since the SP communication subsystem is a user space protocol and executes within the user's process, it must regain control from the application to complete asynchronous requests for communication.

The SP communication subsystem can regain control from the application in any one of three different methods:

1. Any subsequent calls to the SP communication subsystem to post send or receive, or to wait on messages.
2. A timer signal is received periodically to allow the communication subsystem to do recovery from transmission errors.
3. If the value of **MP_CSS_INTERRUPT** is set to **yes**, the communication subsystem device driver will send a signal to the user application when data is received or buffer space is available to transmit data.

Method 1 and Method 2 are always enabled. Method 3 is controlled by the POE environment variable **MP_CSS_INTERRUPT**, and is enabled when this variable is set to **yes**.

For those applications that have the characteristics mentioned above, this implies that when using asynchronous communication the completion of the communication must occur through one of the these three methods. In the case that **MP_CSS_INTERRUPT** is not enabled, only the first two methods are available to process communication. Depending upon the amount of time between the non-synchronized send or receive pairs, or between the nonblocking send or receive and the corresponding waits, the actual transmission of data may only complete at the matching wait call. If this is the case, it is possible that an application may see a performance degradation due to unnecessary processor stalling waiting for communication.

As an example, consider the following application template, where both processors execute the same code, and processor 0 sends and receives data from processor 1.

```
DO LOOP

   MP_SEND (A ...., msgid1)

   MP_RECV (B ...., msgid2)



   MP_WAIT (msgid2, nbytes)



   COMPUTE LOOP1 (uses B)



   MP_WAIT (msgid1, nbytes)



   COMPUTE LOOP2 (modifies A)

ENDDO
```

In this example, application B is guaranteed to be received after the wait for `msgid2`, and more than likely the data is actually received during the wait call. B can then be safely used in the `compute loop1`. A is not guaranteed to be sent until the wait for `msgid1`. Therefore, A cannot be modified until after this wait.

With **MP_CSS_INTERRUPT=no**, it is likely that processor0 receives B during the wait for `msgid2`, and enters the `compute loop1` before the send of A has completed. In this case, processor1 will stall waiting for the completion of the wait for `msgid2`, which will not complete until processor0 completes the `compute loop1` and reaches the wait for `msgid1`. The stalling of processor1 is directly related to the non-continuous flow of communication. If **MP_CSS_INTERRUPT=yes**, when the communication is ready to complete, the communication subsystem device driver sends a signal to the application and causes the application to immediately complete the communication. Therefore data flow is continuous and smooth. The send of A can be completed, even during the `compute loop1`, preventing the stalling of processor1 and improving overall performance of this application.

Finally, it should be noted that there is a cost associated with handling the signals when **MP_CSS_INTERRUPT** is set to **yes**. In some cases, this cost can degrade application performance. Therefore, **MP_CSS_INTERRUPT** should only be used for those applications that require it. For the IP version of the library, **MP_CSS_INTERRUPT=yes** enables UDP to send a SIGIO signal when a message packet is received.

Support for Performance Improvements

POE provides interfaces to improve interrupt mode latency in general, and to increase performance of the receive-and-call mechanism.

Interrupt Mode Improvements

When a node receives a packet and an interrupt is generated, the interrupt handler checks its tables for the process identifier (PID) of the user process and notifies the process. The signal handler or service thread waits for at least two times the interrupt delay, checking to see if more packets will arrive. Waiting for more packets avoids the cost of incurring an interrupt each time a new packet arrives (interrupt processing is very expensive). However, the more packets that arrive, the more delay time is increased. Therefore, with the functions you can either tune the delay parameter based on your application, and/or dynamically turn interrupts on or off at selected nodes.

For an application with few nodes exchanging small messages, it will help latency if you keep the interrupt delay small. For an application with a large number of nodes, or one which exchanges large messages, keeping the delay parameter large will help the bandwidth. A large delay allows multiple read transmissions to occur in a single read cycle. You should experiment with different values and use the functions described below to achieve desired performance, depending on the communication pattern.

MP_INTRDELAY is the environment variable which allows you to set the delay parameter for how long the signal handler or service thread waits for more data. The delay specified in the environment variable is set during initialization, before running the program. In this way, user programs can tune the delay parameter without having to recompile existing applications. If none is specified, the default value of 1 microsecond is used. The application can tune this parameter based on the communication pattern it has in different parts of the application.

Five application programming interfaces are provided to help you enable or disable interrupts on specific tasks, based on the communication patterns of the tasks. If a task is frequently in the communication library, then the application can turn interrupts off for that particular task for the duration of the program. The application can enable interrupts when the task is not going to be in the communication subsystem often. The enable or disable interfaces override the setting of the **MP_CSS_INTERRUPT** environment variable.

The first two functions allow you to query what the current delay parameter is and to set the delay parameter to a new value.

int mpc_queryintrdelay() - for C programs
void mp_queryintrdelay(int rc) - for Fortran programs

This function returns the current interrupt delay (in microseconds). If none was set by the user, the default is returned.

int mpc_setintrdelay(int val) - for C programs
void mp_setintrdelay(int val, int rc) - for Fortran programs

This function sets the delay parameter to the value, in microseconds, specified by "val." The function can be called at multiple places within the program to set the delay parameter to different values during execution.

The following three functions allow you to control dynamically masking interrupts on individual nodes, and query the state of interrupts. In the current system only “all” nodes or “none” can be selected to statically enable or disable running in interrupt mode.

int mpc_queryintr() - for C programs
void mp_queryintr(int rc) - for Fortran programs

This function returns 0 if the node on which it is executed has interrupts turned off, and it returns 1 otherwise.

int mpc_disableintr() - for C programs
void mp_disableintr(int rc) - for Fortran programs

This function disables interrupts on the node on which it is executed. Return code = 0, if successful, -1 otherwise.

int mpc_enableintr() - for C programs
void mp_enableintr(int rc) - for Fortran programs

This function enables interrupts on the node on which it is executed. Return code = 0, if successful, -1 otherwise.

Note: The last two of the above functions override the setting of the environment variable **MP_CSS_INTERRUPT**. If they are not used properly they can deadlock the application. Please use these functions *only* if you are sure of what you are doing. These functions are useful in reducing latency if the application is doing blocking recv/wait and interrupts are otherwise enabled. Interrupts should be turned off before executing blocking communication calls and turned on immediately after those calls.

All of the above functions can also be used for programs running IP.

Rcvncall Improvements

The **mpc_wait** function can be called just before re-posting the Rcvncall instead of in the beginning of the Rcvncall handler, if information provided by the wait function call (like length of message) is already available. This removes the wait time from the critical path for latency. The wait function provides the message id, the length of the message, and also cleans up the resources used by the previously posted Rcvncall. This applies to the signal-handling MPI/MPL library only.

Parallel File Copy Utilities

During the course of developing and running parallel applications on numerous nodes, the potential need exists to efficiently copy data and files to and from a number of places. POE provides three utilities for this reason:

1. **mcp** - to copy a single file from the home node to a number of remote nodes. This was discussed briefly in “Step 2: Copy Files to Individual Nodes” on page 13.
2. **mcpscat** - to copy a number of files from task 0 and scatter them in sequence to all tasks, in a round robin order.
3. **mcpgath** - to copy (or gather) a number of files from all tasks back to task 0.

mcp is for copying the same file to all tasks. The input file must reside on task 0. You can copy it to a new name on the other tasks, or to a directory. It accepts the source file name and a destination file name or directory, in addition to any POE command line argument, as input parameters.

mcpscat is intended for distributing a number of files in sequence to a series of tasks, one at a time. It will use a round robin ordering to send the files in a one to one correspondence to the tasks. If the number of files exceeds the number of tasks, the remaining files are sent in another round through the tasks.

mcpgath is for when you need to copy a number of files from each of the tasks back to a single location, task 0. The files must exist on each task. You can optionally specify to have the task number appended to the file name when it is copied.

Both **mcpscat** and **mcpgath** accept the source file names and a destination directory, in addition to any POE command line argument, as input parameters. You can specify multiple file names, a directory name (where all files in that directory, not including subdirectories, are copied), or use wildcards to expand into a list of files as the source. Wildcards should be enclosed in double quotes, otherwise they will be expanded locally, which may not produce the intended file name resolution.

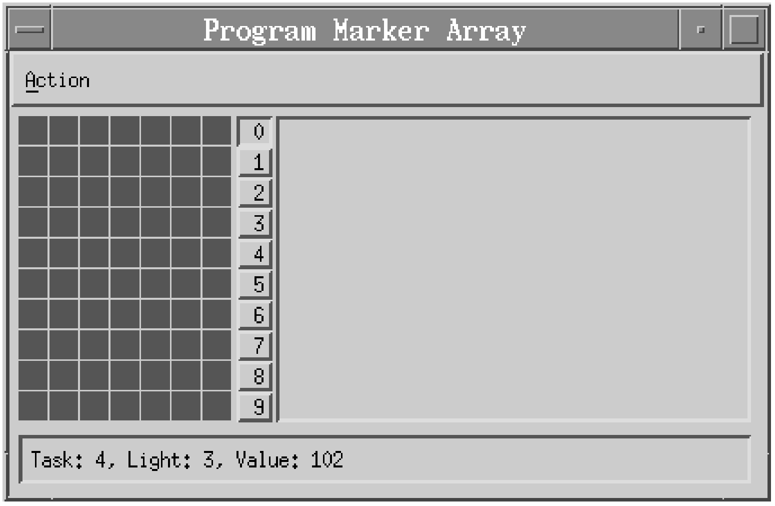
These utilities are actually message passing applications provided with POE. Their syntax is described in Appendix A, “Parallel Environment Commands” on page 83.

Chapter 4. Monitoring Program Execution and System Activity

This chapter describes how you monitor programs on the system. Included are instructions on how to use the Program Marker Array and System Status Array.

Using the Program Marker Array

The Program Marker Array (shown in Figure 1) is an X-Windows run-time monitoring tool. This window consists of a number of small squares called *lights* that change color under program control. Each task in a parallel program has its own row of lights, and Parallel Utility Function calls from those tasks can change light colors. The calls can also send strings to the PM Array.



Task: 4, Light: 3, Value: 102

Figure 1. The Program Marker Array

The ability to color lights on, and send strings to, the PM Array window enables a parallel program to provide you with immediate visual feedback as it executes. A program could begin by coloring lights red and then slowly move through the spectrum towards blue as it executes. If a program takes a long time to run, this would give you an indication that it was indeed progressing. Should the program not be progressing, the PM Array would indicate that as well. For example, lights "stuck" on a particular color could indicate that the program is stuck as well. The strings displayed could provide additional information on the program's progress. In addition, the Program Marker Array is distributed as source code, so you can customize the program as you see fit. The source code is located in the directory */usr/lpp/ppe.poe/samples/marker.*

In order to use the PM Array to monitor program execution, you need to:

1. In your C, C++, and/or Fortran program, insert subroutine calls to color lights on, and send strings to, the PM Array. This is described in "Step 1: Call PM Array Parallel Utility Functions" on page 76.
2. Compile and link the program using the **mpcc**, **mpCC**, or **mpxlf** command. These commands call the C, C++, or Fortran compilers while linking in the Partition Manager interface and Parallel Utility subroutines.

3. Set the environment variable **MP_PMLIGHTS** equal to the number of lights you would like displayed per program task. Alternatively, you can set the number of lights using a command-line flag when invoking your parallel program. See "Step 3: Set the Number of Lights" on page 76.
4. Issue the **pmarray** command to start the PM Array program as described in "Step 4: Open the PM Array Window" on page 77.
5. Invoke your parallel program and monitor its execution using the PM Array. This is described in "Step 5: Invoke the Program and Monitor its Execution" on page 77.

Step 1: Call PM Array Parallel Utility Functions

In order for the PM Array to display meaningful information at run time, you need to place calls to Parallel Utility Functions within your program. At run time, your program can then:

- color lights on, and/or send output strings to, the PM Array Window. This is done by calling (in C programs) the mpc_marker or (in Fortran programs) the MP_MARKER Parallel Utility Function.
- determine the number of lights displayed per task row. This is done by calling (in C programs) the mpc_nlights or (in Fortran programs) the MP_NLIGHTS Parallel Utility Function. Since the number of lights displayed for each task on the PM Array can vary from run to run, this capability is important. It enables your program to learn exactly how many lights are available to be set. It returns an integer value that can then be used by the program to resolve some conditional expression.

The syntax of these Parallel Utility Functions is shown in *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*

Step 2: Compile the Program

Once you have inserted calls to the **mpc_marker** and **mpc_nlights** functions or the **MP_MARKER** and **MP_NLIGHTS** subroutines into your program, you can compile it. Since this is the same procedure you follow when regularly compiling a parallel program with POE, see page 10 for more information. see "Step 1: Compile the Program" on page 10 for more information.

Step 3: Set the Number of Lights

When you open the PM Array window in the next step, the number of rows in the PM Array are set to the number of program tasks – the current setting of **MP_PROCS**. You can also specify the number of lights you want displayed per task row. To do this, set the environment variable **MP_PMLIGHTS** or specify the **-pmlights** command-line flag in "Step 5: Invoke the Program and Monitor its Execution" on page 77. in "Step 5: Invoke the Program and Monitor its Execution" on page 77.

For example, say you want five lights displayed per task in the PM Array. You could:

Set the MP_PMLIGHTS environment variable:	**Use the -pmlights flag when invoking your executable:**
ENTER **export MP_PMLIGHTS=***5*	**ENTER** **poe** *program* **-pmlights** *5*

As with most POE command-line flags, the **-pmlights** flag temporarily overrides its associated environment variable.

Notes:

1. Setting the **MP_PMLIGHTS** environment variable, or the **-pmlights** flag, to *0* indicates that you do not want your program to communicate with the PM Array tool.
2. If you reset the **MP_PMLIGHTS** environment variable, or the **-pmlights** flag, after the Program Marker Array tool is started, it will usually reset to the new number of lights. The only time it will not, however, is when the new value of **MP_PMLIGHTS** is *0*.

Step 4: Open the PM Array Window

The **pmarray** command starts the PM Array program. You will probably want to use the **& &** operator so the program runs in the background and does not tie up the aixterm window.

ENTER **pmarray & pmarray &**

● The PM Array window opens. The number of task rows displayed in the PM Array is equal to the current setting of **MP_PROCS**. The number of lights per task row is determined by the current setting of **MP_PMLIGHTS**. If, when you invoke your program in Step 5: Invoke the Program and Monitor its Execution, you override either of the environment variables using its associated command-line flag, the PM Array redisplays with the new number of rows and/or lights.

Note: The PM Array connects to the Partition Manager using a socket assigned, by default, to port 9999. If you get an error message indicating that the port is in use, specify a different port by setting the **MP_USRPORT** environment variable before entering the **pmarray** command. For example, to specify port 9998:

ENTER **export MP_USRPORT=***9998*

Step 5: Invoke the Program and Monitor its Execution

Finally, you invoke your program. As the program runs, the Parallel Utility Function calls placed within it change the color of lights on the PM Array. With appropriate mouse clicks on this window, you can:

- display details of a light
- display output strings from a task
- close the window and discontinue monitoring

Displaying Details of a Light

Each light on the PM Array is associated with a particular task, has a particular light number, and has a particular color value. You can display these details for each of the lights on the PM Array.

For example, say you have coded the PM Array subroutines into your program so that the lights slowly move during execution through the spectrum over color values 0 to 99. As the program runs, the lights start off black, and then turn brown, green, blue, and so on. By watching the lights as they change color, you get a general idea of the program's progress. For a more precise indication of the program's progress, you could display the actual color value number for a light. In this example, the closer this light's value is to 99, the closer execution is to being complete.

To display details of a light:

PLACE the cursor over any light on the PM Array.

PRESS the left mouse button.

● The following information displays in the text area at the bottom of the PM Array window:

- the task identifier number
- the light number
- the color value number

This information is not updated until you select another light.

Displaying Task Output

You can display output strings sent by the tasks of your program in the output display area of the PM Array window. This is the area to the right of the PM Array, and the strings displayed there are the ones you specified on the mpc_marker or MP_MARKER subroutine calls. Only one task's strings are displayed in this area at a time. By default, output from task 0 is displayed. You can select the task and display its output instead by pressing its task push button. Each task has a push button. It is just to the right of the task's row on the PM Array, and is labeled with the task identifier. To select, for example, task 3:

PRESS the task pushbutton labeled 3.

● Output strings from task 3 are displayed in the output display area. Only one string is displayed at a time.

Note: If a task not currently selected has sent new output to the PM Array window, its task push button will appear yellow.

Step 6: Close the PM Array Window

The PM Array window remains open after your parallel program completes executing. You could then repeat Step 5: Invoke the Program and Monitor its Execution to monitor the same, or a different program's execution. To close the PM Array window when you are done monitoring:

SELECT **Action → Quit**

Using the System Status Array

Note: VT must be installed in order to use the System Status Array. See *IBM Parallel Environment for AIX: Installation* for more information on installing VT.

The System Status Array is an X-Windows monitoring tool that lets you quickly survey the utilization of processor nodes. This tool is particularly useful if you are | not using a job management system , and so must manually schedule which nodes should be used to run a parallel program. The System Status Array lets you easily see the CPU utilization of each of your processor nodes. Using a host list file, you can then have your program run on those processor nodes you expect to be the least busy. The host list file can contain up to 255 nodes.

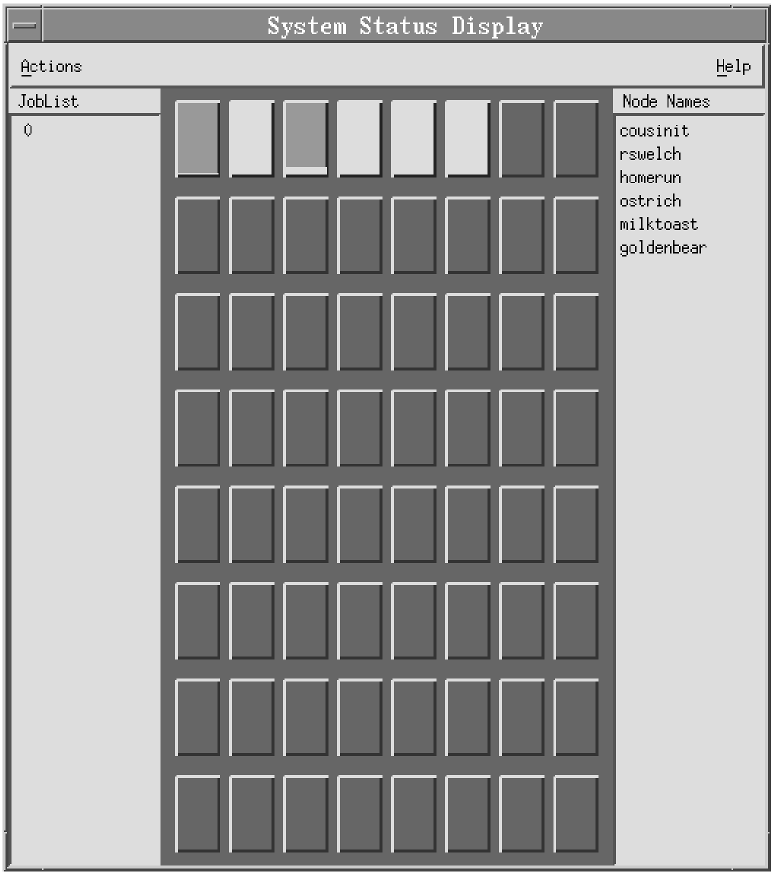


Figure 2. The System Status Array (actual array is 16 by 16 nodes)

Each square on the System Status Array represents a processor node of your SP system or cluster. The squares are colored pink and yellow to show the instantaneous percent of CPU utilization for each processor node. If a square were to appear all pink, it would be at 0 percent utilization. If a square were to appear all yellow, it would be at 100 percent utilization. The percent of the area colored yellow responds to the percent of CPU utilization. If a square appears gray, the node is unavailable for monitoring – either it does not have the Statistics Collector daemon (*digd*) running, or the System Status Array cannot communicate with it.

To the right of the Array is a node list which contains the name of each node in the Array. You use this list to identify the name of a node represented in the Array. The nodes are listed in order, left to right, starting with the top row of the Array. If you | are using the resource manager on an SP system, the nodes are displayed in the pool order returned by the **jm_status** command. To use **jm_status** on a workstation that is external to the SP system, the *ssp.clients* fileset must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information). For more information on this command, see *IBM Parallel System Support Programs for AIX: Command and Technical Reference* If you are using an | RS/6000 network cluster, or are using LoadLeveler on an SP system , the order of nodes displayed is determined by the order in which they are contacted.

Notes:

1. In order to use this tool, the Visualization Tool statistics collector daemon process (*digd*) needs to be running on each of the nodes you wish to monitor. This daemon feeds the System Status Array with the CPU information it displays. The program that interrogates the digd daemon is installed as part of the Visualization Tool's installation procedure.
2. Do not attempt to display more than 255 nodes at one time. If you wish to view more than 255 nodes, you can invoke multiple sessions using the **poestat** command (described below) to cover that many nodes.

Opening and Closing the System Status Array Window

The **poestat** command starts the System Status Array, and opens its window. How you invoke this command differs depending on whether you are monitoring an SP system or an RS/6000 network cluster. The method the Array uses to find the processor nodes to monitor is also different. In either case, you will probably want to use the **&** operator so the program runs in the background and does not tie up the aixterm window.

If you are monitoring an SP system:	**If you are monitoring an RS/6000 network cluster:**
First make sure the environment variable **SP_NAME** is set to the name of your control workstation. This environment variable identifies the Resource Manager you are using. When running **poestat** from a workstation that is external to an SP system, the *ssp.clients* file set must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information). **ENTER** **export SP_NAME=***control_workstation_name* To start the Array: **ENTER** **poestat &** ● The System Status Array window (shown in Figure 2 on page 79) opens. The System Status Array program connects to the Resource Manager to automatically select all the nodes of your SP system for monitoring.	**ENTER** **poestat -norm &** ● The System Status Array window (shown in Figure 2 on page 79) opens. The System Status Array program selects for monitoring each of the nodes on the LAN that have the Statistics Collector Daemon running. It also selects each of the nodes listed in the host list file indicated by the **MP_HOSTFILE** environment variable. If **MP_HOSTFILE** is not set, the default is *host.list* in your current directory. **Note:** If a node listed in the host list file references a node that is also on the LAN, the node will be represented twice on the Array.

Note that **poestat** allows you to specify a list of nodes to be monitored. You do this by setting the **MP_HOSTFILE** environment variable to **MP_HOSTFILE**=*name_of_host_list_file*. The default is *host.list*.

Note: For National Language Support, this X-Windows tool displays messages located in an externalized message catalog. If you get an error saying that a message catalog is not found, and want the default message catalog that we provide:

ENTER **export NLSPATH=/usr/lib/nls/msg/%L/%N**

export LANG=C

For more information about the message catalog, see "National Language Support" on page xiv.

When the System Status Array opens, all nodes display status. You can selectively stop and start displaying status for a single node or all nodes.

To toggle between displaying and not displaying status for a single node:

PLACE	the cursor over the name of the processor node.
PRESS	the left mouse button.

To toggle between displaying and not displaying status for all nodes:

PLACE	the cursor over the digit that appears in the column to the left of the Array under the word “JobList”.
PRESS	the left mouse button.

To close the System Status Array window once you have finished monitoring:

SELECT	**Actions → Done**

Appendix A. Parallel Environment Commands

This appendix contains the manual pages for the PE commands discussed throughout this book. Each manual page is organized into the sections listed below. The sections always appear in the same order, but some appear in all manual pages while others are optional.

NAME	Provides the name of the command described in the manual page, and a brief description of its purpose.
SYNOPSIS	Includes a diagram that summarizes the command syntax, and provides a brief synopsis of its use and function. If you are unfamiliar with the typographic conventions used in the syntax diagrams, see "Typographic Conventions" on page xii.
FLAGS	Lists and describes any required and optional flags for the command.
DESCRIPTION	Describes the command more fully than the **NAME** and **SYNOPSIS** sections.
ENVIRONMENT VARIABLES	Lists and describes any applicable environment variables.
EXAMPLES	Provides examples of ways in which the command is typically used.
FILES	Lists and describes any files related to the command.
RELATED INFORMATION	Lists commands, functions, file formats, and special files that are employed by the command, that have a purpose related to the command, or that are otherwise of interest within the context of the command.

mcp

NAME

mcp – Allows you to propagate a copy of a file to multiple nodes on an IBM POWERparallel system.

SYNOPSIS

mcp *infile* [*outfile*] [*POE options*]

In the command synopsis above, the infile is the name of the file to be copied. You can copy to a new name by specifying an outfile. If you do not provide the outfile name, the file will be placed in its current directory on each node. The outfile can be either an explicit output file name or a directory name. When a directory is specified, the file is copied with the same name to that directory.

DESCRIPTION

The **mcp** command allows you to propagate a copy of a file to multiple nodes on an IBM RS/6000 SP. The file must initially reside (or be NFS-mounted) on at least one node.

mcp is a POE program and, therefore, all POE options are available. You can set POE options with either command line flags or environment variables. The number of nodes to copy the file to (**-procs**), and the message passing protocol used to copy the file (**-euilib**) are the POE options of most interest. The input file must be readable from the node assigned to task 0.

Note: A POE job loads faster if a copy of the job resides on each node. For this reason, it is suggested that you use **mcp** to copy your executable to a file system such as /tmp, which resides on each node.

Return codes are:

Note: The actual command return code value is 128 plus the unsigned return code value. That is, a return code of -2 will give a value of 130. For more information, see the “Exit Status” section in *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*

-1	incorrect usage
-2	error opening input file
-3	error opening *to* file on originating node
-4	error writing data to *to* file on originating node
-5	no room on remote node's file system
-6	error opening file on remote node
-7	error writing data on remote node
-8	error renaming temp file to file name
-9	input file is empty
-10	invalid block size
-11	error allocating storage

ENVIRONMENT VARIABLES

MP_BLKSIZE	sets the block size used for copying the data. This can be a value between 1 and 8,000,000 (8 megabytes). The default is 100,000 (100K).

EXAMPLES

1. To copy a file from your current directory to the current directory on all nodes of a 16-processor system, using the High Performance Switch, enter:

```
mcp filename -procs 16 -euilib us
```

2. To copy a filename from your current directory to the /tmp directory on all nodes of a 16-processor system, using IP, enter:

```
mcp filename /tmp -procs 16 -euilib ip
```

3. To copy a file from your current directory to a different filename on all nodes of a 16-processor system, enter:

   ```
   mcp filename /tmp/newfilename -procs 16
   ```

RELATED INFORMATION

Commands: **mprcp**(1), **rcp**(1)

mcpgath

NAME

mcpgath – Takes files from each task of tasks 0 through task N and copies them back in sequence to task 0.

SYNOPSIS

mcpgath [**-ai**] *source ... destination* [*POE options*]

Source is one of the following:

- one or more existing file names - files will be copied with the same names to the destination directory on task 0. Each file name specified must exist on all tasks involved in the copy.
- a directory name - all files in that directory on each task are copied with the same names to the destination directory on task 0.
- an expansion of file names, using wildcards - files are copied with the same names to the destination directory. All wildcarded input strings must be enclosed in double quotes.

Destination is an existing destination directory name to where the data will be copied. The destination directory must be the last item specified before any POE flags.

FLAGS

-a	An optional flag that appends the task number to the end of the file name when it is copied to task 0. This is for task identification purposes, to know where the data came from. The **-a** and **-i** flags can be combined to check for existing files appended with the task number.
-i	An optional flag that checks for duplicate or existing files of the same name, and does not replace any existing file found. Instead, issues an error message and continues with the remaining files to be copied. The **-a** and **-i** flags can be combined to check for existing files appended with the task number.

See Chapter 2, Executing Parallel Programs for information on POE options.

DESCRIPTION

The **mcpgath** function determines the list of files to be gathered on each task. This function also resolves the source file, destination directory, and path names with any meta characters, wildcard expansions, etc. to come up with valid file names. Wildcards should be enclosed in double quotes, otherwise they will be expanded locally on the task from where the command is issued, which may not produce the intended file name resolution.

mcpgath is a POE program and, therefore, all POE options are available. You can set POE options with either command line flags or environment variables. The number of nodes to copy the file to (**-procs**), and the message passing protocol used to copy the file (**-euilib**) are the POE options of most interest.

Note: A default of 100K data block size is used for copying the data. This can be changed by updating the source found in */usr/lpp/ppe.poe/samples/mpi*, and compiling it with the **mpcc** command.

Return codes are:

| **Note:** The actual command return code value is 128 plus the unsigned return
| code value. That is, a return code of -2 will give a value of 130. For more
| information, see the “Exit Status” section in *IBM Parallel Environment for*
| *AIX: MPI Programming and Subroutine Reference*

-1	invalid number of arguments specified
-2	invalid option flag specified
-3	unable to resolve input file name(s)
-4	could not open input file for read
-5	no room on destination node's file system
-6	error opening file output file
-7	error creating output file
-8	error writing to output file
-9	**MPI_Send** of data failed
-10	final **MPI_Send** failed
-11	**MPI_Recv** failed
-12	invalid block size
-13	error allocating storage
-14	total number of tasks must be greater than one

ENVIRONMENT VARIABLES

MP_BLKSIZE	sets the block size used for copying the data. This can be a value between 1 and 8,000,000 (8 megabytes). The default is 100,000 (100K).

EXAMPLES

1. You can copy a single file from all tasks into the destination directory. For example, enter:

   ```
   mcpgath -a hello_world /tmp -procs 4
   ```

   This will copy the file *hello_world* (assuming it is a file and not a directory) from tasks 0 through 3 as to task 0:

   ```
   From task 0:  /tmp/hello_world.0

   From task 1:  /tmp/hello_world.1

   From task 2:  /tmp/hello_world.2

   From task 3:  /tmp/hello_world.3
   ```

2. You can specify any number of files as source files. The destination directory must be the last item specified before any POE flags. For example:

   ```
   mcpgath -a file1.a file2.a file3.a file4.a file5.a /tmp -procs 4
   ```

   will take *file1.a* through *file5.a* from the local directory on each task and copy them back to task 0. All files specified must exist on all tasks involved. The file distribution will be as follows:

```
From Task 0: /tmp/file1.a.0

From Task 1: /tmp/file1.a.1

From Task 2: /tmp/file1.a.2

From Task 3: /tmp/file1.a.3

From Task 0: /tmp/file2.a.0

From Task 1: /tmp/file2.a.1

From Task 2: /tmp/file2.a.2

From Task 3: /tmp/file2.a.3

From Task 0: /tmp/file3.a.0

From Task 1: /tmp/file3.a.1

From Task 2: /tmp/file3.a.2

From Task 3: /tmp/file3.a.3

From Task 0: /tmp/file4.a.0

From Task 1: /tmp/file4.a.1

From Task 2: /tmp/file4.a.2

From Task 3: /tmp/file4.a.3

From Task 0: /tmp/file5.a.0

From Task 1: /tmp/file5.a.1

From Task 2: /tmp/file5.a.2

From Task 3: /tmp/file5.a.3
```

3. You can specify wildcard values to expand into a list of files to be gathered. For this example, assume the following distribution of files before calling **mcpgath**:

   ```
   Task 0 contains file1.a and file2.a

   Task 1 contains file1.a only

   Task 2 contains file1.a, file2.a, and file3.a

   Task 3 contains file4.a, file5.a, and file6.a
   ```

   Enter:

   ```
   mcpgath -a "file*.a" /tmp -procs 4
   ```

   This will pass the wildcard expansion to each task, which will resolve into the list of locally existing files to be copied. This will result in the following distribution of files on task 0:

```
From Task 0: /tmp/file1.a.0

From Task 0: /tmp/file2.a.0

From Task 1: /tmp/file1.a.1

From Task 2: /tmp/file1.a.2

From Task 2: /tmp/file2.a.2

From Task 2: /tmp/file3.a.2

From Task 3: /tmp/file4.a.3

From Task 3: /tmp/file5.a.3

From Task 3: /tmp/file6.a.3
```

4. You can specify a directory name as the source, from which the files to be gathered are found. For this example, assume the following distribution of files before calling **mcpgath**:

   ```
   Task 0 /test contains file1.a and file2.a

   Task 1 /test contains file1.a only

   Task 2 /test contains file1.a and file3.a

   Task 3 /test contains file2.a, file4.a, and file5.a
   ```

   Enter:

   ```
   mcpgath -a /test /tmp -procs 4
   ```

   This results in the following file distribution:

   ```
   From Task 0: /tmp/file1.a.0

   From Task 0: /tmp/file2.a.0

   From Task 1: /tmp/file1.a.1

   From Task 2: /tmp/file1.a.2

   From Task 2: /tmp/file3.a.2

   From Task 3: /tmp/file2.a.3

   From Task 3: /tmp/file4.a.3

   From Task 3: /tmp/file5.a.3
   ```

mcpscat

NAME

mcpscat – Takes a number of files from task 0 and scatters them in sequence to all tasks, in a round robin order.

SYNOPSIS

mcpscat [**-f**] [**-i**] *source ... destination*
[*POE options*]

Source can be one of the following:

- a single file name - file is copied to all tasks
- a single file name that contains a list of file names (**-f** option)
- two of more file names - files will be distributed in a round robin order to the tasks
- an expansion of file names, using wildcards - files will be distributed in a round robin order to the tasks
- a directory name - all files in that directory are copied in a round robin order to the tasks.

Destination is an existing destination directory name to where the data will be copied.

FLAGS

-f	Is an optional flag that says the first file contains the names of the source files that are to be scattered. Each file name, in the file, must be specified on a separate line. No wildcards are supported when this option is used. Directory names are not supported in the file either. When this option is used, the **mcpscat** parameters should consist of a single source file name (for the list of files) and a destination directory. The files will then be scattered just as if they had all been specified on the command line in the same order as they are listed in the file.
-i	Checks for duplicate or existing files of the same name, and does not replace any existing file found. Instead, issues an error message and continues with the remaining files to be copied. Without this flag, the default action is to replace any existing files with the source file.

See Chapter 2, Executing Parallel Programs for information on POE options.

DESCRIPTION

The **mcpscat** function determines the order in which to distribute the files, using a round robin method, according to the list of nodes and number of tasks. Files are sent in a one-to-one correspondence to the nodes in the list of tasks. If the number of files specified is greater than the number of nodes, the remaining files are sent in another round through the list of nodes. Wildcards should be enclosed in double quotes, otherwise they will be expanded locally on the task from where the command is issued, which may not produce the intended file name resolution.

mcpscat is a POE program and, therefore, all POE options are available. You can set POE options with either command line flags or environment variables. The number of nodes to copy the file to (**-procs**), and the message passing protocol used to copy the file (**-euilib**) are the POE options of most interest.

Return codes are:

Note: The actual command return code value is 128 plus the unsigned return code value. That is, a return code of -2 will give a value of 130. For more information, see the “Exit Status” section in *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*

-1	invalid number of arguments specified
-2	invalid option flag specified
-3	unable to resolve input file name(s)
-4	could not open input file for read
-5	no room on destination node's file system
-6	error opening file output file
-7	error creating output file
-8	**MPI_Send** of data failed
-9	final **MPI_Send** failed
-10	**MPI_Recv** failed
-11	failed opening temporary file
-12	failed writing temporary file
-13	error renaming temp file to filename
-14	input file is empty (zero byte file size)
-15	invalid block size
-16	error allocating storage
-17	number of tasks and files do not match
-18	not enough memory for list of file names

ENVIRONMENT VARIABLES

MP_BLKSIZE	sets the block size used for copying the data. This can be a value between 1 and 8,000,000 (8 megabytes). The default is 100,000 (100K).

EXAMPLES

1. You can copy a single file to all tasks into the destination directory. For example, enter:

   ```
   mcpscat filename /tmp -procs 4
   ```

   This will take the file and distribute it to tasks 0 through 3 as */tmp/filename*.

2. You can specify any number of files as source files. The destination directory must be the last item specified before any POE flags. For example:

   ```
   mcpscat file1.a file2.a file3.a file4.a file5.a /tmp -procs 4
   ```

   will take *file1.a* through *file5.a* from the local directory and copy them in a round robin order to tasks 0 through 3 into */tmp*. The file distribution will be as follows:

   ```
   Task 0: /tmp/file1.a

   Task 1: /tmp/file2.a

   Task 2: /tmp/file3.a

   Task 3: /tmp/file4.a

   Task 0: /tmp/file5.a
   ```

3. You can specify the source files to copy in a file. For example:

   ```
   mcpscat -f file.list /tmp -procs 4
   ```

   will produce the same results as the previous example if as *file.list* contains five lines with the file names *file1.a* through *file5.a* in it.

4. You can specify wildcard values to expand into a list of files to be scattered. Enter:

   ```
   mcpscat "file*.a" /tmp -procs 4
   ```

   Assuming Task 0 contains *file1.a*, *file2.a*, *file3.a*, *file4.a*, and *file5.a* in its home directory, this will result in a similar distribution as in the previous example.

5. You can specify a directory name as the source, from which the files to be scattered are found. Assuming */test* contains *myfile.a*, *myfile.b*, *myfile.c*, *myfile.d*, *myfile.f*, and *myfile.g* on Task 0, enter:

   ```
   mcpscat /test /tmp -procs 4
   ```

   This results in the following file distribution:

   ```
   Task 0: /tmp/myfile.a

   Task 1: /tmp/myfile.b

   Task 2: /tmp/myfile.c

   Task 3: /tmp/myfile.d

   Task 0: /tmp/myfile.f

   Task 1: /tmp/myfile.g
   ```

mpamddir

NAME

mpamddir – echoes an amd-mountable directory name.

SYNOPSIS

mpamddir

or, if you're using the Parallel Environment for AIX:

export *MP_REMOTEDIR=mpamddir*

This script determines whether or not the current directory is a mounted file system. If it is, it looks to see if it appears in the amd maps, and constructs a name for the directory that is known to amd. You can modify this script, or create additional ones that apply to your installation.

By default, POE uses the Korn shell **pwd** command to obtain the name of the current directory to pass to the remote nodes for execution. This works for C shell users if the current directory is:

- The home directory
- Not mounted by amd, the AutoMount Daemon.

If this is not the case, (for example, if the user's current directory is a subdirectory of the home directory), then you can supply your own script for providing the name of the current directory on the remote nodes.

To use **mpamddir** as the script for providing the name, export the environment variable **MP_REMOTEDIR**, and set it to **mpamddir**.

RELATED INFORMATION

Commands: **ksh**(1), **poe**(1), **csh**(1)

---

mpcc

NAME

mpcc – Invokes a shell script to compile C programs.

SYNOPSIS

mpcc [*cc_flags*]... *program.c*

The **mpcc** shell script compiles C programs while linking in the Partition Manager, Message Passing Interface (MPI), and/or Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **cc** command can also be used on **mpcc**. For a complete listing of these flag options, refer to the manual page for the AIX **cc** command. Typical options to **mpcc** include:

-v
causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** and **pedb** debuggers, and is also needed by the Source Code view of the Visualization Tool (**vt**).

-o
names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

DESCRIPTION

The **mpcc** shell script calls the AIX **xlc** compiler. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in “Creating a Static Executable” on page 11 in place of this script.

Flags are passed by **mpcc** to the **xlc** command, so any of the **xlc** options can be used on the **mpcc** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically loaded.

ENVIRONMENT VARIABLES

MP_PREFIX
sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a C program, enter:

```
mpcc program.c -o program
```

FILES

When you compile a program using **mpcc**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi.a (Message Passing Interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc.a (POE version of libc.a)

RELATED INFORMATION

Commands: **mpcc_r**(1), **mpCC**(1), **mpCC_r**(1), **mpxlf**(1), **cc**(1), **pdbx**(1), **pedb**(1), **vt**(1), **xprofiler**(1)

---

mpcc_chkpt

NAME

mpcc_chkpt – Invokes a shell script to compile checkpointable C programs.

SYNOPSIS

mpcc_chkpt [*cc_flags*]... **-us | -ip** *program.c*

The **mpcc_chkpt** shell script compiles C programs while linking in the Partition Manager, Message Passing Interface (MPI), and support code for parallel Checkpoint/Restart. It builds an executable with no shared objects.

FLAGS

Most of the compiler flags normally accepted by the **cc** command can also be used on **mpcc_chkpt**. For a complete listing of these flag options, refer to the manual page for the AIX **cc** command. Typical options to **mpcc_chkpt** include:

-v causes a “verbose” output listing of the shell script.

-g Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** and **pedb** debuggers, and is also needed by the Source Code view of the Visualization Tool (**vt**).

-o names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p	enables profiling with the **prof** command. For more information, see the appendix on "Profiling Programs" in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*
-pg	enables profiling with the **xprofiler** and **gprof** commands. For more information, see the "Xprofiler" chapter and the appendix on "Profiling Programs" in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*
-ip	specifies that the executable is bound with the UDP/IP message passing support library.
-us	specifies that the executable is bound with the RS/6000 SP User Space message passing library. Executables using this option should be compiled on an RS/6000 SP node compatible with the node on which execution will occur.

DESCRIPTION

The **mpcc_chkpt** shell script invokes the AIX **cc** command. The Partition Manager, message passing interface, and checkpoint support code are automatically linked in. The script creates an executable with no shared obects. This executable must be run on a node of the same machine type and having the same level of system software as the machine on which the executable is built. The executable is not binary compatible over changes to the system software.

Flags are passed by **mpcc_chkpt** to the **cc** command, so most of the **cc** options can be used on the **mpcc_chkpt** shell script. Options which would generate shared objects should not be used.

At execution time, the value specified by the **MP_EUILIB** environment variable or the **-euilib** flag must match the **-ip | -us** option specified when this command was run.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a C program, enter:

```
mpcc_chkpt program.c -o program
```

FILES

When you compile a program using **mpcc_chkpt**, the following libraries are automatically included:

/usr/lpp/ppe.poe/lib/libmpi.a (Message Passing Interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc.a (POE version of libc.a)

| /usr/lpp/LoadL/nfs/lib/chkrst.a (LoadLeveler checkpoint/support)

| When the **-us** option is selected, the following libraries are included:

| /usr/lpp/ppe.poe/lib/us/libmpci.a (PSSP message passing interface)
| /usr/lpp/ssp/css/lib/hal.a (PSSP User Space adapter interface)
| /usr/lib/swclock.o (PSSP Switch clock interface)

| When the **-ip** option is selected, the following libraries are included:

| /usr/lpp/ppe.poe/lib/ip/libmpci.a (PSSP message passing interface)

| RELATED INFORMATION

| Commands: **mpcc**(1), **mpCC_chkpt**(1), **mpxlf_chkpt**(1)

---

mpcc_r

NAME

mpcc_r – Invokes a shell script to compile C programs which use threaded MPI.

SYNOPSIS

mpcc_r [*cc_flags*]... *program.c*

The **mpcc_r** shell script compiles C programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and Low-level Applications Programming Interface (LAPI).

FLAGS

Any of the compiler flags normally accepted by the **xlc_r** or **cc_r** command can also be used on **mpcc_r**. For a complete listing of these flag options, refer to the manual page for the AIX **cc_r** command. Typical options to **mpcc_r** include:

-v causes a “verbose” output listing of the shell script.

-g Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** and **pedb** debuggers, and is also needed by the Source Code view of the Visualization Tool (**vt**).

-o names the executable.

| **-d7** compiles the program with POSIX Threads Draft 7 base MPI and
| compatibility libraries. Otherwise, POSIX Threads Draft 10 base
| libraries are used.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p	enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*
-pg	enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

DESCRIPTION

The **mpcc_r** shell script calls the AIX **xlc_r** compiler. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in “Creating a Static Executable” on page 11 in place of this script.

Flags are passed by **mpcc_r** to the **xlc_r** command, so any of the **xlc_r** options can be used on the **mpcc_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a C program, enter:

```
mpcc_r program.c -o program
```

FILES

When you compile a program using **mpcc_r**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi_r.a (Message Passing Interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd_r.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc_r.a (POE version of libc_r.a)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

RELATED INFORMATION

Commands: **mpCC**(1), **mpCC_r**(1), **mpcc**(1), **cc**(1), **pdbx**(1), **pedb**(1), **vt**(1), **xprofiler**(1)

---

mpCC

NAME

mpCC – Invokes a shell script to compile C++ programs.

SYNOPSIS

mpCC [*xlC_flags*]... *program.C*

The **mpCC** shell script compiles C++ programs while linking in the Partition Manager, Message Passing Interface (MPI), and/or Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **xlC** command can also be used on **mpCC**. For a complete listing of these flag options, refer to the manual page for the **xlC** command. Typical options to **mpCC** include:

-v
causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references. This object file is needed by the Source Code view of the Visualization Tool (**vt**).

-o
names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

DESCRIPTION

The **mpCC** shell script calls the AIX **xlC** compiler. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in “Creating a Static Executable” on page 11 in place of this script.

Flags are passed by **mpCC** to the **xlC** command, so any of the **xlC** options can be used on the **mpCC** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a C++ program, enter:

```
mpCC program.C -o program
```

FILES

When you compile a program using **mpCC**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc.a (POE version of libc.a)

RELATED INFORMATION

Commands: **mpCC_r**(1), **mpcc**(1), **mpcc_r**(1), **mpxlf**(1), **xlC**(1), **pdbx**(1), **pedb**(1), **vt**(1), **xprofiler**(1)

| ## mpCC_chkpt

| ## NAME

| **mpCC_chkpt** – Invokes a shell script to compile checkpointable C++ programs.

| ## SYNOPSIS

| **mpCC_chkpt** [*xlC_flags*]... **-us | -ip** *program.C*

| The **mpCC_chkpt** shell script compiles C++ programs while linking in the Partition
| Manager, Message Passing Interface (MPI), and support code for parallel
| Checkpoint/Restart. It builds an executable with no shared objects.

FLAGS

Most of the compiler flags normally accepted by the **xlC** command can also be used on **mpCC_chkpt**. For a complete listing of these flag options, refer to the manual page for the AIX **xlC** command. Typical options to **mpCC_chkpt** include:

-v	causes a “verbose” output listing of the shell script.
-g	Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** and **pedb** debuggers, and is also needed by the Source Code view of the Visualization Tool (**vt**).
-o	names the executable.
-l (lower-case L)	names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.
-I (upper-case i)	names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.
-p	enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*
-pg	enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*
-ip	specifies that the executable is bound with the UDP/IP message passing support library.
-us	specifies that the executable is bound with the RS/6000 SP User Space message passing library. Executables using this option should be compiled on an RS/6000 SP node compatible with the node on which execution will occur.

DESCRIPTION

The **mpCC_chkpt** shell script invokes the AIX **xlC** command. The Partition Manager, message passing interface, and checkpoint support code are automatically linked in. The script creates an executable with no shared obects. This executable must be run on a node of the same machine type and having the same level of system software as the machine on which the executable is built. The executable is not binary compatible over changes to the system software.

Flags are passed by **mpCC_chkpt** to the **xlC** command, so most of the **xlC** options can be used on the **mpCC_chkpt** shell script. Options which would generate shared objects should not be used.

At execution time, the value specified by the **MP_EUILIB** environment variable or the **-euilib** flag must match the **-ip | -us** option specified when this command was run.

ENVIRONMENT VARIABLES

MP_PREFIX sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a C++ program, enter:

```
mpCC_chkpt program.C -o program
```

FILES

When you compile a program using **mpCC_chkpt**, the following libraries are automatically included:

/usr/lpp/ppe.poe/lib/libmpi.a (Message Passing Interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc.a (POE version of libc.a)
/usr/lpp/LoadL/full/lib/chkrst.a (LoadLeveler checkpoint/support)

When the **-us** option is selected, the following libraries are included:

/usr/lpp/ppe.poe/lib/us/libmpci.a (PSSP message passing interface)
/usr/lpp/ssp/css/lib/hal.a (PSSP User Space adapter interface)
/usr/lib/swclock.o (PSSP Switch clock interface)

When the **-ip** option is selected, the following libraries are included:

/usr/lpp/ppe.poe/lib/ip/libmpci.a (PSSP message passing interface)

RELATED INFORMATION

Commands: **mpCC**(1), **mpcc_chkpt**(1), **mpxlf_chkpt**(1)

mpCC_r

NAME

mpCC_r – Invokes a shell script to compile C++ programs which use threaded MPI.

SYNOPSIS

mpCC_r [*xlC_flags*]... *program.C*

The **mpCC_r** shell script compiles C++ programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and Low-level Applications Programming Interface (LAPI).

FLAGS

Any of the compiler flags normally accepted by the **xlC_r** command can also be used on **mpCC_r**. For a complete listing of these flag options, refer to the manual page for the **xlC_r** command. Typical options to **mpCC_r** include:

-v causes a “verbose” output listing of the shell script.

-g Produces an object file with symbol table references. This object file is needed by the Source Code view of the Visualization Tool (**vt**).

-o names the executable.

| **-d7** compiles the program with POSIX Threads Draft 7 base MPI and compatibility libraries. Otherwise, POSIX Threads Draft 10 base libraries are used.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-pg enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

DESCRIPTION

The **mpCC_r** shell script calls the AIX **xlC_r** compiler. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in “Creating a Static Executable” on page 11 in place of this script.

Flags are passed by **mpCC_r** to the **xlC_r** command, so any of the **xlC_r** options can be used on the **mpCC_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a C++ program, enter:

```
mpCC_r program.C -o program
```

FILES

When you compile a program using **mpCC_r**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi_r.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd_r.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc_r.a (POE version of libc_r.a)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

RELATED INFORMATION

Commands: **mpcc_r**(1), **mpcc**(1), **mpCC**(1), **xlC**(1), **pdbx**(1), **pedb**(1), **vt**(1), **xprofiler**(1)

mprcp

NAME

mprcp – copies a file from the home node to a list of remote hosts.

SYNOPSIS

mprcp *host_list_file file_id*

The **mprcp** shell script uses the **rcp** command to copy the file specified by *file_id* to all the remote hosts listed in the specified *host_list_file*. This *file_id* must be specified by an absolute path name.

FLAGS

None.

DESCRIPTION

The **mprcp** shell script is typically used to distribute executables, data files, and (in order to use the parallel debuggers) source code files from the home node to all the remote nodes of the partition prior to invoking **poe**. This only needs to be done if the needed files are not in a shared file system, or are not part of a file collection which is distributed automatically.

For each remote host listed in the specified *host_list_file*, the size and date of *file_id* on the remote system is determined. If the file does not exist on the remote system, or if either the size or date differs from the corresponding statistic for the local system, the **rcp** command is used to copy the file to the remote system. The copied file retains the local system size and date and overlays any existing file of the same name. The remote copy uses the same userid as the local system. If the remote host cannot be contacted, or if the **rcp** command fails, an error message is printed and the script exits.

EXAMPLES

To send a copy of the executable *program* to all the processor nodes listed in *host.list* in your current directory, enter:

```
mprcp host.list $PWD/program
```

RELATED INFORMATION

Commands: **rcp**(1), **mcp**(1)

mpxlf

NAME

mpxlf – Invokes a shell script to compile Fortran programs.

SYNOPSIS

mpxlf [*xlf_flags*]... *program.f*

The **mpxlf** shell script compiles Fortran programs while linking in the Partition Manager, Message Passing Interface (MPI), and/or Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf**. For a complete listing of these flag options, refer to the manual page for the **xlf** command. Typical options to **mpxlf** include:

-v causes a “verbose” output listing of the shell script.

-g Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** and **pedb** debuggers, and is also needed by the Source Code view of the Visualization Tool.

-o names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p	enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*
-pg	enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

DESCRIPTION

The **mpxlf** shell script calls the **xlf** compiler. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in “Creating a Static Executable” on page 11 in place of this script.

Flags are passed by **mpxlf** to the **xlf** command, so any of the **xlf** options can be used on the **mpxlf** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

| **mpxlf**, **mpxlf_r**, **mpxlf90**, **mpxlf90_r**, **mpxlf_chkpt**, **mpxlf90_chkpt** have been updated to support High Performance Fortran (HPF) Version 1, Release 3. In doing so, if other Fortran compilers are installed on your system in addition to HPF 1.3, you may need to use a new environment variable with the compiler script in order to use HPF 1.3.

| **xlf** and HPF use different compiler paths and stanzas. Therefore, the scripts will now check for the compiler level installed, and will do the following:

- If only **xlf** is installed on the system, it will be used.
- If only HPF is installed, only HPF 1.3 or greater is supported and it will be used.
- If both **xlf** and HPF 1.3 are installed, **xlf** is used as the default, unless the customer overrides it by specifying the **MP_HPF** environment variable.

| As such, the POE Fortran compile scripts check for a new environment variable, **MP_HPF**, to determine if the HPF should be used. Customers with both HPF and **xlf** installed should set **MP_HPF=YES** when they desire to use the HPF 1.3 compiler.

| Customers without both HPF and **xlf** installed do not need to set the **MP_HPF** variable.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

MP_HPF	If High Performance Fortran (HPF) 1.3 is installed along with other Fortran compilers, set this to **YES** if the HPF 1.3 compiler is to be used. Otherwise the default **xlf** compiler will be used.

EXAMPLES

To compile a Fortran program, enter:

```
mpxlf program.f -o program
```

FILES

When you compile a program using **mpxlf**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc.a (POE version of libc.a)

RELATED INFORMATION

Commands: **mpcc**(1), **xlf**(1), **pdbx**(1), **pedb**(1), **vt**(1), **xprofiler**(1)

mpxlf_chkpt

NAME

mpxlf_chkpt – Invokes a shell script to compile checkpointable FORTRAN programs.

SYNOPSIS

mpxlf_chkpt [*xlf_flags*]... **-us | -ip** *program.f*

The **mpxlf_chkpt** shell script compiles FORTRAN programs while linking in the Partition Manager, Message Passing Interface (MPI), and support code for parallel Checkpoint/Restart. It builds an executable with no shared objects.

FLAGS

Most of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf_chkpt**. For a complete listing of these flag options, refer to the manual page for the AIX **xlf** command. Typical options to **mpxlf_chkpt** include:

-v	causes a “verbose” output listing of the shell script.
-g	Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** and **pedb** debuggers, and is also needed by the Source Code view of the Visualization Tool (**vt**).
-o	names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-ip
specifies that the executable is bound with the UDP/IP message passing support library.

-us
specifies that the executable is bound with the RS/6000 SP User Space message passing library. Executables using this option should be compiled on an RS/6000 SP node compatible with the node on which execution will occur.

DESCRIPTION

The **mpxlf_chkpt** shell script invokes the AIX **xlf** command. The Partition Manager, message passing interface, and checkpoint support code are automatically linked in. The script creates an executable with no shared obects. This executable must be run on a node of the same machine type and having the same level of system software as the machine on which the executable is built. The executable is not binary compatible over changes to the system software.

Flags are passed by **mpxlf_chkpt** to the **xlf** command, so most of the **xlf** options can be used on the **mpxlf_chkpt** shell script. Options which would generate shared objects should not be used.

At execution time, the value specified by the **MP_EUILIB** environment variable or the **-euilib** flag must match the **-ip | -us** option specified when this command was run.

mpxlf, **mpxlf_r**, **mpxlf90**, **mpxlf90_r**, **mpxlf_chkpt**, **mpxlf90_chkpt** have been updated to support High Performance Fortran (HPF) Version 1, Release 3. In doing so, if other Fortran compilers are installed on your system in addition to HPF 1.3, you may need to use a new environment variable with the compiler script in order to use HPF 1.3.

xlf and HPF use different compiler paths and stanzas. Therefore, the scripts will now check for the compiler level installed, and will do the following:

- If only **xlf** is installed on the system, it will be used.
- If only HPF is installed, only HPF 1.3 or greater is supported and it will be used.

- If both **xlf** and HPF 1.3 are installed, **xlf** is used as the default, unless the customer overrides it by specifying the **MP_HPF** environment variable.

As such, the POE Fortran compile scripts check for a new environment variable, **MP_HPF**, to determine if the HPF should be used. Customers with both HPF and **xlf** installed should set **MP_HPF=YES** when they desire to use the HPF 1.3 compiler.

Customers without both HPF and **xlf** installed do not need to set the **MP_HPF** variable.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.
MP_HPF	If High Performance Fortran (HPF) 1.3 is installed along with other Fortran compilers, set this to **YES** if the HPF 1.3 compiler is to be used. Otherwise the default **xlf** compiler will be used.

EXAMPLES

To compile a FORTRAN program, enter:

```
mpxlf_chkpt program.f -o program
```

FILES

When you compile a program using **mpxlf_chkpt**, the following libraries are automatically included:

/usr/lpp/ppe.poe/lib/libmpi.a (Message Passing Interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc.a (POE version of libc.a)
/usr/lpp/LoadL/full/lib/chkrst.a (LoadLeveler checkpoint/support)

When the **-us** option is selected, the following libraries are included:

/usr/lpp/ppe.poe/lib/us/libmpci.a (PSSP message passing interface)
/usr/lpp/ssp/css/lib/hal.a (PSSP User Space adapter interface)
/usr/lib/swclock.o (PSSP Switch clock interface)

When the **-ip** option is selected, the following libraries are included:

/usr/lpp/ppe.poe/lib/ip/libmpci.a (PSSP message passing interface)

RELATED INFORMATION

Commands: **mpxlf**(1), **mpcc_chkpt**(1), **mpxlf90_chkpt**(1)

mpxlf_r

NAME

mpxlf_r – Invokes a shell script to compile Fortran programs and link them into a threaded environment.

SYNOPSIS

mpxlf_r [*xlf_flags*]... *program.f*

The **mpxlf_r** shell script compiles Fortran programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and Low-level Applications Programming Interface (LAPI).

Note: Only one thread can run a Fortran program.

FLAGS

Any of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf_r**. For a complete listing of these flag options, refer to the manual page for the **xlf** command. Typical options to **mpxlf_r** include:

-v causes a "verbose" output listing of the shell script.

-g Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** and **pedb** debuggers, and is also needed by the Source Code view of the Visualization Tool.

-o names the executable.

| **-d7** compiles the program with POSIX Threads Draft 7 base MPI and
| compatibility libraries. Otherwise, POSIX Threads Draft 10 base
| libraries are used. This flag can only be used with Fortran Version
| 5.1.1 or later.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p enables profiling with the **prof** command. For more information, see the appendix on "Profiling Programs" in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-pg enables profiling with the **xprofiler** and **gprof** commands. For more information, see the "Xprofiler" chapter and the appendix on "Profiling Programs" in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

DESCRIPTION

The **mpxlf_r** shell script calls the **xlf** compiler. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in “Creating a Static Executable” on page 11 in place of this script.

Flags are passed by **mpxlf_r** to the **xlf** command, so any of the **xlf** options can be used on the **mpxlf_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

| **mpxlf**, **mpxlf_r**, **mpxlf90**, **mpxlf90_r**, **mpxlf_chkpt**, **mpxlf90_chkpt** have been updated to support High Performance Fortran (HPF) Version 1, Release 3. In doing so, if other Fortran compilers are installed on your system in addition to HPF 1.3, you may need to use a new environment variable with the compiler script in order to use HPF 1.3.

| **xlf** and HPF use different compiler paths and stanzas. Therefore, the scripts will now check for the compiler level installed, and will do the following:

- If only **xlf** is installed on the system, it will be used.
- If only HPF is installed, only HPF 1.3 or greater is supported and it will be used.
- If both **xlf** and HPF 1.3 are installed, **xlf** is used as the default, unless the customer overrides it by specifying the **MP_HPF** environment variable.

| As such, the POE Fortran compile scripts check for a new environment variable, **MP_HPF**, to determine if the HPF should be used. Customers with both HPF and **xlf** installed should set **MP_HPF=YES** when they desire to use the HPF 1.3 compiler.

| Customers without both HPF and **xlf** installed do not need to set the **MP_HPF** variable.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.
MP_HPF	If High Performance Fortran (HPF) 1.3 is installed along with other Fortran compilers, set this to **YES** if the HPF 1.3 compiler is to be used. Otherwise the default **xlf** compiler will be used.

EXAMPLES

To compile a Fortran program, enter:

```
mpxlf_r program.f -o program
```

FILES

When you compile a program using **mpxlf_r**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi_r.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd_r.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc_r.a (POE version of libc_r.a)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

Note: Fortran Version 4.1.0.1, specifically the library libxlf90_t.a, must be available for both linking and execution.

RELATED INFORMATION

Commands: **mpcc_r**(1), **xlf**(1), **pdbx**(1), **pedb**(1), **vt**(1), **xprofiler**(1)

---

mpxlf90

NAME

mpxlf90 – Invokes a shell script to compile Fortran 90 programs.

SYNOPSIS

mpxlf90 [*xlf_flags*]... *program.f*

The **mpxlf90** shell script compiles Fortran 90 programs while linking in the Partition Manager, Message Passing Interface (MPI), and Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf90**. For a complete listing of these flag options, refer to the manual page for the **xlf** command. Typical options to **mpxlf90** include:

-v causes a “verbose” output listing of the shell script.

-g Produces an object file with symbol table references. This object file is needed by the Source Code view of the Visualization Tool.

-o names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p enables profiling with the **prof** command. For more information, see the appendix on "Profiling Programs" in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-pg enables profiling with the **xprofiler** and **gprof** commands. For more information, see the "Xprofiler" chapter and the appendix on "Profiling Programs" in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

DESCRIPTION

The **mpxlf90** shell script calls the **xlf** compiler. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in "Creating a Static Executable" on page 11 in place of this script.

Flags are passed by **mpxlf90** to the **xlf** command, so any of the **xlf** options can be used on the **mpxlf90** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

mpxlf, **mpxlf_r**, **mpxlf90**, **mpxlf90_r**, **mpxlf_chkpt**, **mpxlf90_chkpt** have been updated to support High Performance Fortran (HPF) Version 1, Release 3. In doing so, if other Fortran compilers are installed on your system in addition to HPF 1.3, you may need to use a new environment variable with the compiler script in order to use HPF 1.3.

xlf and HPF use different compiler paths and stanzas. Therefore, the scripts will now check for the compiler level installed, and will do the following:

- If only **xlf** is installed on the system, it will be used.
- If only HPF is installed, only HPF 1.3 or greater is supported and it will be used.
- If both **xlf** and HPF 1.3 are installed, **xlf** is used as the default, unless the customer overrides it by specifying the **MP_HPF** environment variable.

As such, the POE Fortran compile scripts check for a new environment variable, **MP_HPF**, to determine if the HPF should be used. Customers with both HPF and **xlf** installed should set **MP_HPF=YES** when they desire to use the HPF 1.3 compiler.

Customers without both HPF and **xlf** installed do not need to set the **MP_HPF** variable.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.
MP_HPF	If High Performance Fortran (HPF) 1.3 is installed along with other Fortran compilers, set this to **YES** if the HPF 1.3 compiler is to be used. Otherwise the default **xlf** compiler will be used.

EXAMPLES

To compile a Fortran 90 program, enter:

```
mpxlf90 program.f -o program
```

FILES

When you compile a program using **mpxlf90**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc.a (POE version of libc.a)

RELATED INFORMATION

Commands: **mpcc**(1), **mpCC**(1), **xlf**(1), **mpxlf**(1), **vt**(1), **xprofiler**(1)

mpxlf90_chkpt

NAME

mpxlf90_chkpt – Invokes a shell script to compile checkpointable Fortran 90 programs.

SYNOPSIS

mpxlf90_chkpt [*xlf90_flags*]... **-us | -ip** *program.f*

The **mpxlf90_chkpt** shell script compiles Fortran 90 programs while linking in the Partition Manager, Message Passing Interface (MPI), and support code for parallel Checkpoint/Restart. It builds an executable with no shared objects.

FLAGS

Most of the compiler flags normally accepted by the **xlf90** command can also be used on **mpxlf90_chkpt**. For a complete listing of these flag options, refer to the manual page for the AIX **xlf90** command. Typical options to **mpxlf90_chkpt** include:

-v	causes a “verbose” output listing of the shell script.

-g
: Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** and **pedb** debuggers, and is also needed by the Source Code view of the Visualization Tool (**vt**).

-o
: names the executable.

-l (lower-case L)
: names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

-I (upper-case i)
: names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p
: enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-pg
: enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-ip
: specifies that the executable is bound with the UDP/IP message passing support library.

-us
: specifies that the executable is bound with the RS/6000 SP User Space message passing library. Executables using this option should be compiled on an RS/6000 SP node compatible with the node on which execution will occur.

DESCRIPTION

The **mpxlf90_chkpt** shell script invokes the AIX **xlf90** command. The Partition Manager, message passing interface, and checkpoint support code are automatically linked in. The script creates an executable with no shared obects. This executable must be run on a node of the same machine type and having the same level of system software as the machine on which the executable is built. The executable is not binary compatible over changes to the system software.

Flags are passed by **mpxlf90_chkpt** to the **xlf90** command, so most of the **xlf90** options can be used on the **mpxlf90_chkpt** shell script. Options which would generate shared objects should not be used.

At execution time, the value specified by the **MP_EUILIB** environment variable or the **-euilib** flag must match the **-ip | -us** option specified when this command was run.

mpxlf, **mpxlf_r**, **mpxlf90**, **mpxlf90_r**, **mpxlf_chkpt**, **mpxlf90_chkpt** have been updated to support High Performance Fortran (HPF) Version 1, Release 3. In doing so, if other Fortran compilers are installed on your system in addition to HPF 1.3, you may need to use a new environment variable with the compiler script in order to use HPF 1.3.

xlf and HPF use different compiler paths and stanzas. Therefore, the scripts will now check for the compiler level installed, and will do the following:

- If only **xlf** is installed on the system, it will be used.
- If only HPF is installed, only HPF 1.3 or greater is supported and it will be used.
- If both **xlf** and HPF 1.3 are installed, **xlf** is used as the default, unless the customer overrides it by specifying the **MP_HPF** environment variable.

As such, the POE Fortran compile scripts check for a new environment variable, **MP_HPF**, to determine if the HPF should be used. Customers with both HPF and **xlf** installed should set **MP_HPF=YES** when they desire to use the HPF 1.3 compiler.

Customers without both HPF and **xlf** installed do not need to set the **MP_HPF** variable.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.
MP_HPF	If High Performance Fortran (HPF) 1.3 is installed along with other Fortran compilers, set this to **YES** if the HPF 1.3 compiler is to be used. Otherwise the default **xlf** compiler will be used.

EXAMPLES

To compile a FORTRAN program, enter:

```
mpxlf90_chkpt program.f -o program
```

FILES

When you compile a program using **mpxlf90_chkpt**, the following libraries are automatically included:

/usr/lpp/ppe.poe/lib/libmpi.a (Message Passing Interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc.a (POE version of libc.a)
/usr/lpp/LoadL/full/lib/chkrst.a (LoadLeveler checkpoint/support)

When the **-us** option is selected, the following libraries are included:

/usr/lpp/ppe.poe/lib/us/libmpci.a (PSSP message passing interface)
/usr/lpp/ssp/css/lib/hal.a (PSSP User Space adapter interface)
/usr/lib/swclock.o (PSSP Switch clock interface)

When the **-ip** option is selected, the following libraries are included:

/usr/lpp/ppe.poe/lib/ip/libmpci.a (PSSP message passing interface)

RELATED INFORMATION

Commands: **mpxlf90**(1), **mpcc_chkpt**(1), **mpxlf_chkpt**(1)

mpxlf90_r

NAME

mpxlf90_r – Invokes a shell script to compile Fortran 90 programs and link them into a threaded environment.

SYNOPSIS

mpxlf90_r [*xlf_flags*]... *program.f*

The **mpxlf90_r** shell script compiles Fortran 90 programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and Low-level Applications Programming Interface (LAPI).

Note: Only one thread can run a Fortran program.

FLAGS

Any of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf90_r**. For a complete listing of these flag options, refer to the manual page for the **xlf** command. Typical options to **mpxlf90_r** include:

-v causes a "verbose" output listing of the shell script.

-g Produces an object file with symbol table references. This object file is needed by the Source Code view of the Visualization Tool.

-o names the executable.

-d7 compiles the program with POSIX Threads Draft 7 base MPI and compatibility libraries. Otherwise, POSIX Threads Draft 10 base libraries are used. This flag can only be used with Fortran Version 5.1.1 or later.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the FILES section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* is automatically included.

-p enables profiling with the **prof** command. For more information, see the appendix on "Profiling Programs" in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

-pg enables profiling with the **xprofiler** and **gprof** commands. For more information, see the "Xprofiler" chapter and the appendix on "Profiling Programs" in *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

DESCRIPTION

The **mpxlf90_r** shell script calls the **xlf** compiler. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in “Creating a Static Executable” on page 11 in place of this script.

Flags are passed by **mpxlf90_r** to the **xlf** command, so any of the **xlf** options can be used on the **mpxlf90_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

mpxlf, **mpxlf_r**, **mpxlf90**, **mpxlf90_r**, **mpxlf_chkpt**, **mpxlf90_chkpt** have been updated to support High Performance Fortran (HPF) Version 1, Release 3. In doing so, if other Fortran compilers are installed on your system in addition to HPF 1.3, you may need to use a new environment variable with the compiler script in order to use HPF 1.3.

xlf and HPF use different compiler paths and stanzas. Therefore, the scripts will now check for the compiler level installed, and will do the following:

- If only **xlf** is installed on the system, it will be used.
- If only HPF is installed, only HPF 1.3 or greater is supported and it will be used.
- If both **xlf** and HPF 1.3 are installed, **xlf** is used as the default, unless the customer overrides it by specifying the **MP_HPF** environment variable.

As such, the POE Fortran compile scripts check for a new environment variable, **MP_HPF**, to determine if the HPF should be used. Customers with both HPF and **xlf** installed should set **MP_HPF=YES** when they desire to use the HPF 1.3 compiler.

Customers without both HPF and **xlf** installed do not need to set the **MP_HPF** variable.

ENVIRONMENT VARIABLES

MP_PREFIX	sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.
MP_HPF	If High Performance Fortran (HPF) 1.3 is installed along with other Fortran compilers, set this to **YES** if the HPF 1.3 compiler is to be used. Otherwise the default **xlf** compiler will be used.

EXAMPLES

To compile a Fortran 90 program, enter:

```
mpxlf90_r program.f -o program
```

FILES

When you compile a program using **mpxlf90_r**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi_r.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libvtd_r.a (VT tracing subsystem)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
/usr/lpp/ppe.poe/lib/libc_r.a (POE version of libc_r.a)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

Note: Fortran Version 4.1.0.1, specifically the library libxlf90_t.a, must be available for both linking and execution.

RELATED INFORMATION

Commands: **mpcc_r**(1), **xlf**(1), **mpxlf_r**(1), **pdbx**(1), **pedb**(1), **vt**(1), **xprofiler**(1)

---

pmarray

NAME

pmarray – Starts the Program Marker Array, which is an X-Windows tool for monitoring a parallel executable's run.

SYNOPSIS

pmarray

The **pmarray** command starts the Program Marker Array X-Windows tool prior to invoking **poe**. This tool is used for run-time monitoring.

FLAGS

None.

DESCRIPTION

The **pmarray** command starts the Program Marker Array. This X-Windows run-time monitoring tool consists of a number of small squares, or lights. Each task in a parallel program has its own row of lights. Using calls to the Parallel Utility Functions enables a parallel program to control the appearance of the Program Marker Array Window. Calls to MP_MARKER (for Fortran programs) or mpc_marker (for C programs) enables a program to color lights on, and/or send output strings to the Window. Calls to MP_NLIGHTS (for Fortran programs) and mpc_nlights (for C programs) enables a program to determine the number of lights displayed per task row.

ENVIRONMENT VARIABLES

This command responds to the following environment variables:

MP_PMLIGHTS	Indicates the number of lights displayed per row on the Array. (If, when you invoke **poe**, you override **MP_PMLIGHTS** using the **-pmlights** flag, the Array will redisplay with the new number of lights per row.)
MP_PROCS	Indicates the number of program tasks. The **pmarray** command sets the number of task rows displayed in the Array equal to the value of **MP_PROCS**. (If, when you invoke **poe**, you override **MP_PROCS** using the **-procs** flag, the Array will redisplay with the new number of rows.)
MP_USRPORT	Indicates the port id used by the Partition Manager to connect to the Array. By default, the Partition Manager connects to the Array using a socket assigned to port 9999. If you get an error message indicating that the port is in use, specify a different port. Standard TCP/IP practice suggests using ports greater than 5000 and less than 10000.

EXAMPLES

To start the Program Marker Array program as a background process, and open its window, enter:

```
pmarray &
```

FILES

/usr/lib/X11/app-defaults/PMarray (Xdefaults file)

/usr/lib/ppe.poe/samples/PMarray.ad (X-Windows resource information)

RELATED INFORMATION

Commands: **poe**(1)

Subroutines: **mpc_marker**(3), **MP_MARKER**(3), **mpc_nlights**(3), **MP_NLIGHTS**(3)

“Monitoring Program Execution Using the Program Marker Array” in *IBM Parallel Environment for AIX: Operation and Use*

---

poe

NAME

poe – Invokes the Parallel Operating Environment (POE) for loading and executing programs on remote processor nodes.

SYNOPSIS

poe [**-h**] [*program*] [*program_options*]...
[**-adapter_use** *adapter_specifier*]
[**-cpu_use** *cpu_specifier*]
[**-euidevice** *device_specifier*]
[**-euilib** {*ip* **|** *us*}]
[**-euilibpath** *path_specifier*]
[{**-hostfile | -hfile**} *host_file_name*]
[**-procs** *partition_size*]
[**-pulse** *interval*]
[**-resd** {**yes | no**}]
[**-retry** *retry_interval*]
[**-retrycount** *retry_count*]
[**-rmpool** *pool_ID*]
[**-savehostfile** *output_file_name*]
[**-spname** *name*]
[**-cmdfile** *commands_file*]
[**-newjob** {**yes | no**}]
[**-pgmmodel** {**spmd | mpmd**}]
[**-labelio** {**yes | no**}]
[**-stdinmode** {**all | none |** *task_ID*}]
[**-stdoutmode** {**unordered | ordered |** *task_ID*}]
[{**-samplefreq | -sfreq**} *samp_int*]
[{**-tbuffsize | -tbsize**} *buffer_size*]
[{**-tbuffwrap | -tbwrap**} {**yes | no**}]
[**-tmpdir** *temp_trace_directory*]
[{**-tracedir | -tdir**} *final_trace_directory*]
[{**-tracefile | -tfile**} *trace_file_name*]
[{**-tracelevel | -tlevel**} *level_int*]
[{**-ttempsize | -ttsize**} *temp_file_size*]
[{**-infolevel | -ilevel**} *message_level*]
[**-pmdlog** {**yes | no**}]
[**-buffer_mem**] *memory_size*
[**-css_interrupt** {**yes | no**}]
[**-eager_limit** *size_limit*]
[**-intrdelay** *delay_parameter*]
[**-max_typedepth** *maximum_depth*]
[**-use_flow_control** {**yes | no**}]
[**-thread_stacksize**]
[**-single_thread** {**no | yes**}]
[**-wait_mode** {**poll | yield | sleep**}]
[**-polling_interval** *interval*]
[**-euidevelop** {**yes | no | deb | min | nor**}]
[**-pmlights** *number_of_lights*]
[**-usrport** *port_ID*] [*fence_string additional_options*]
[**-coredir**]

The **poe** command invokes the Parallel Operating Environment for loading and executing programs on remote processor nodes. The operation of POE is influenced by a number of POE environment variables. The flag options on this command are each used to temporarily override one of these environment variables. User *program_options* can be freely interspersed with the flag options. If no *program* is specified, POE will either prompt you for programs to load, or, if the

MP_CMDFILE environment variable is set, will load the partition using the specified commands file.

FLAGS

The **-h** flag, when used, must appear immediately after **poe**, and causes the **poe** man page, if it exists, to be printed to the screen.

The remaining flags you can specify on this command are used to temporarily override POE environment variables. For more information on valid values, and on what a particular flag sets, refer to the description of its associated environment variable in the ENVIRONMENT VARIABLES section. The following flags are grouped by function.

The following Partition Manager control flags override the associated environment variables.

-adapter_use	**MP_ADAPTER_USE**
-cpu_use	**MP_CPU_USE**
-euidevice	**MP_EUIDEVICE**
-euilib	**MP_EUILIB**
-euilibpath	**MP_EUILIBPATH**
-hostfile or **-hfile**	**MP_HOSTFILE**
-procs	**MP_PROCS**
-pulse	**MP_PULSE**
-resd	**MP_RESD**
-retry	**MP_RETRY**
-retrycount	**MP_RETRYCOUNT**
| **-msg_api**	**MP_MSG_API**
-rmpool	**MP_RMPOOL**
| **-nodes**	**MP_NODES**
| **-tasks_per_node**	**MP_TASKS_PER_NODE**
-savehostfile	**MP_SAVEHOSTFILE**
-spname	**SP_NAME**

The following Job Specification flags override the associated environment variables.

-cmdfile	**MP_CMDFILE**
-newjob	**MP_NEWJOB**
-pgmmodel	**MP_PGMMODEL**

The following I/O Control flags override the associated environment variables.

-labelio	**MP_LABELIO**
-stdinmode	**MP_STDINMODE**
-stdoutmode	**MP_STDOUTMODE**

The following VT Trace Collection flags override the associated environment variables.

-samplefreq or **-sfreq**	**MP_SAMPLEFREQ**
-tbuffsize or **-tbsize**	**MP_TBUFFSIZE**
-tbuffwrap or **-tbwrap**	**MP_TBUFFWRAP**
-tmpdir	**MP_TMPDIR**

-tracedir or **-tdir**	**MP_TRACEDIR**
-tracefile or **-tfile**	**MP_TRACEFILE**
-tracelevel or **-tlevel**	**MP_TRACELEVEL**
-ttempsize or **-ttsize**	**MP_TTEMPSIZE**

The following generation of diagnostic information flags override the associated environment variables.

-infolevel or **-ilevel**	**MP_INFOLEVEL**
-pmdlog	**MP_PMDLOG**

The following Message Passing flags override the associated environment variables.

-buffer_mem	**MP_BUFFER_MEM**
-css_interrupt	**MP_CSS_INTERRUPT**
-eager_limit	**MP_EAGER_LIMIT**
-intrdelay	**MP_INTRDELAY**
-max_typedepth	**MP_MAX_TYPEDEPTH**
-use_flow_control	**MP_USE_FLOW_CONTROL**
-thread_stacksize	**MP_THREAD_STACKSIZE**
-single_thread	**MP_SINGLE_THREAD**
-wait_mode	**MP_WAIT_MODE**
| **-polling_interval**	**MP_POLLING_INTERVAL**

The following are miscellaneous flags:

-euidevelop	Overrides the **MP_EUIDEVELOP** environment variable.
-pmlights	Determines the number of lights displayed (per row) on the Program Marker Array. This overrides the **MP_PMLIGHTS** environment variable. For more information on the Program Marker Array, refer to the manual page for the **pmarray** command.
-usrport	Overrides the **MP_USRPORT** environment variable.
-coredir	Overrides the **MP_COREDIR** environment variable.

DESCRIPTION

The **poe** command invokes the Parallel Operating Environment for loading and executing programs on remote nodes. You can enter it at your home node to:

- load and execute an SPMD program on all nodes of your partition.
- individually load the nodes of your partition with an MPMD job.
- load and execute a series of SPMD and MPMD programs, in individual job steps, on the same partition.
- run non-parallel programs on remote nodes.

The operation of POE is influenced by a number of POE environment variables. The flag options on this command are each used to temporarily override one of these environment variables. User *program_options* can be freely interspersed with the flag options, and *additional_options* not to be parsed by POE can be placed after a *fence_string* defined by the **MP_FENCE** environment variable. If no *program* is specified, POE will either prompt you for programs to load, or, if the

MP_CMDFILE environment variable is set, will load the partition using the specified commands file.

The environment variables and flags that influence the operation of this command fall into distinct categories of function. They are:

- **Partition Manager control**. The environment variables and flags in this category determine the method of node allocation, message passing mechanism, and the PULSE monitor function.
- **Job specification**. The environment variables and flags in this category determine whether or not the Partition Manager should maintain the partition for multiple job steps, whether commands should be read from a file or STDIN, and how the partition should be loaded.
- **I/O control**. The environment variables and flags in this category determine how I/O from the parallel tasks should be handled. These environment variables and flags set the input and output modes, and determine whether or not output is labeled by task id.
- **VT trace collection**. The environment variables and flags in this category determine if and how execution traces are collected for playback using the Visualization Tool (**vt**). They determine which types of traces are collected, and how trace storage is handled.
- **Generation of diagnostic information**. The environment variables and flags in this category enable you to generate diagnostic information that may be required by the IBM Support Center in resolving PE-related problems.
- **Message Passing Interface**. The environment variables and flags in this category enable you to specify values for tuning message passing applications.
- **Miscellaneous**. The additional environment variables and flags in this category enable additional error checking, and set a dispatch priority class for execution.

ENVIRONMENT VARIABLES

The environment variable descriptions in this section are grouped by function.

The following environment variables are associated with Partition Manager control.

MP_ADAPTER_USE
Determines how the node's adapter should be used. If using LoadLeveler, the US communication subsystem library does not require dedicated use of the SP switch on the node. Adapter use will be defaulted, as in Table 4 on page 25, but shared usage may be specified. If using the Resource Manager, this value is only used when POE is requesting non-specific nodes via the **MP_RMPOOL** or **-rmpool** setting. Valid values are *dedicated* and *shared*. If not set, the default is dedicated for US jobs, or shared for IP jobs. The value of this environment variable can be overridden using the **-adapter_use** flag.

MP_AUTH
Determines the type of user authorization to be used. Valid values are **AIX** (the default) for AIX based user authorization, using **/etc/hosts.equiv** or **.rhosts** entries, and **DFS** for DFS/DCE based authorization. This value can be overridden

	with an entry in the **/etc/poe.limits** file. There is no associated command line flag.
MP_CPU_USE	Determines how the node's CPU should be used. If using LoadLeveler, the US communication subsystem library does not require unique CPU use on the node. CPU use will be defaulted, as in Table 4 on page 25, but multiple use may be specified. If using the Resource Manager, this value is only used when POE is requesting non-specific nodes via the **MP_RMPOOL** or **-rmpool** setting. Valid values are *multiple* and *unique*. If not set, the default is unique for US jobs, or multiple for IP jobs. The value of this environment variable can be overridden using the **-cpu_use** flag.
MP_EUIDEVICE	Determines the adapter set to use for message passing. Valid values are *en0* (for Ethernet), *fi0* (for FDDI), *tr0* (for token-ring), and *css0* (for the SP system's high performance switch feature).
MP_EUILIB	Determines the communication subsystem library implementation to use for communication – either the IP communication subsystem or the US communication subsystem. In order to use the US communication subsystem, you must have an SP system configured with its high performance switch feature. Valid, case-sensitive, values are **ip** (for the IP communication subsystem) or **us** (for the US communication subsystem). The value of this environment variable can be overridden using the **-euilib** flag.
MP_EUILIBPATH	Determines the path to the message passing and communication subsystem libraries. This only needs to be set if the libraries are moved. Valid values are any path specifier. The value of this environment variable can be overridden using the **-euilibpath** flag.
MP_HOSTFILE	Determines the name of a host list file for node allocation. Valid values are any file specifier. If not set, the default is *host.list* in your current directory. The value of this environment variable can be overridden using the **-hostfile** or **-hfile** flags.
MP_PROCS	Determines the number of program tasks. Valid values are any number from 1 to 128. If not set, the default is 1. The value of this environment variable can be overridden using the **-procs** flag.
MP_PULSE	The interval (in seconds) at which POE checks the remote nodes to ensure that they are communicating with the home node. The default interval is 600 seconds (10 minutes). To disable the pulse function, specify an interval of 0 (zero) seconds. The pulse function is automatically disabled when running the **pdbx** or **pedb** debuggers. You can override the value of this environment variable with the **-pulse** flag.
MP_REMOTEDIR	Specifies the name of a script which echoes the name of the current directory to be used on the remote nodes. By default, the current directory is the current directory at the time that

	POE is run. You may need to specify this if the AutoMount Daemon is used to mount user file systems, and the user is not using the Korn shell. The script **mpamddir** is provided for mapping the C shell directory name to an AutoMount Daemon name.
MP_RESD	Determines whether or not the Partition Manager should connect to LoadLeveler or the SP system Resource Manager to allocate nodes. Valid values are either **yes** or **no**, and there is no default. The value of this environment variable can be overridden using the **-resd** flag.
MP_RETRY	Determines the period of time (in seconds) between processor node allocation retries if there are not enough processor nodes immediately available to run a program. This is only valid if you are using LoadLeveler or the SP system Resource Manager. Valid values are any integer greater than or equal to 0. The default is 0 (no retry). The value of this environment variable can be overridden using the **-retry** flag.
MP_RETRYCOUNT	The number of times (at the interval set by **MP_RETRY**) that the Partition Manager should attempt to allocate processor nodes. Valid values are any integer greater than or equal to 0. If not set, the default is 0. The value of this environment variable can be overridden using the **-retrycount** flag.
MP_MSG_API	Indicates to POE which message passing API is being used by the parallel tasks. You need to set this environment variable if a parallel task is using LAPI alone or in conjunction with MPI. You do not need to set it if a parallel task is using MPI only. The value of this environment variable can be overridden using the **-msg_api** flag.
MP_RMPOOL	With regard to LoadLeveler, determines the name or number of the pool that should be used for non-specific node allocation. With regard to the Resource Manager, determines the number of the SP system pool that should be used for non-specific node allocation. This environment variable/command-line flag only applies to LoadLeveler or the SP system Resource Manager. Valid values are any identifying pool name or number for LoadLeveler, and any identifying pool number for the Resource Manager. There is no default. The value of this environment variable can be overridden using the **-rmpool** flag.
MP_NODES	Specifies the number of physical nodes on which to run the parallel tasks. It may be used alone or in conjunction with **MP_TASKS_PER_NODE** and/or **MP_PROCS**, as described in Table 7 on page 33. The value of this environment variable can be overridden using the **-nodes** flag.
MP_TASKS_PER_NODE	Specifies the number of tasks to be run on each of the physical nodes. It may be used in conjunction with **MP_NODES** and/or **MP_PROCS**, as described in Table 7 on page 33, but may not be used alone. The value of this

	environment variable can be overridden using the **-tasks_per_node** flag.
MP_SAVEHOSTFILE	The name of an output host list file to be generated by the Partition Manager. Valid values are any relative or full path name. The value of this environment variable can be overridden using the **-savehostfile** flag.
MP_TIMEOUT	Controls the length of time POE waits before abandoning an attempt to connect to the remote nodes. The default is 150 seconds. **MP_TIMEOUT** also changes the length of time the communication subsystem will wait for a connection to be established during message passing initialization.
SP_NAME	Determines the job management system (LoadLeveler or the SP system Resource Manager) to use. If all nodes to be used for the parallel job exist in a PSSP 2.4.0 partition, the **SP_NAME** environment variable should be set to the name of the control workstation of the SP system on which these nodes exist. This is the only case that results in POE contacting the Resource Manager rather than LoadLeveler for node allocation requests. When running POE from a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the external node (see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation* for more information). When running POE from a workstation that is external to the SP system, and using the Resource Manager, the *ssp.clients* fileset must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information).
MP_CHECKFILE	Defines the base name of the checkpoint file when checkpointing or restarting a program. See “Checkpointing and Restarting Programs” on page 51 for more information.
MP_CHECKDIR	Defines the directory where the checkpoint file will reside when checkpointing or restarting a program. See “Checkpointing and Restarting Programs” on page 51 for more information.

The following environment variables are associated with Job Specification.

MP_CMDFILE	Determines the name of a POE commands file used to load the nodes of your partition. If set, POE will read the commands file rather than STDIN. Valid values are any file specifier. The value of this environment variable can be overridden using the **-cmdfile** flag.
MP_NEWJOB	Determines whether or not the Partition Manager maintains your partition for multiple job steps. Valid values are **yes** or **no**. If not set, the default is **no**. The value of this environment variable can be overridden using the **-newjob** flag.

MP_PGMMODEL	Determines the programming model you are using. Valid values are **spmd** or **mpmd**. If not set, the default is **spmd**. The value of this environment variable can be overridden using the **-pgmmodel** flag.

The following environment variables are associated with I/O Control.

MP_LABELIO Determines whether or not output from the parallel tasks are labeled by task id. Valid values are **yes** or **no**. If not set, the default is **no**. The value of this environment variable can be overridden using the **-labelio** flag.

MP_STDINMODE Determines the input mode – how STDIN is managed for the parallel tasks. Valid values are:

all	all tasks receive the same input data from STDIN.
none	no tasks receive input data from STDIN; STDIN will be used by the home node only.
n	STDIN is only sent to the task identified (*n*).

If not set, the default is **all**. The value of this environment variable can be overridden using the **-stdinmode** flag.

MP_HOLD_STDIN Determines whether or not sending of STDIN from the home node to the remote nodes is deferred until the message passing partition has been established. Valid values are **yes** or **no**. If not set, the default is **no**.

MP_STDOUTMODE Determines the output mode – how STDOUT is handled by the parallel tasks. Valid values are:

unordered	all tasks write output data to STDOUT asynchronously.
ordered	output data from each parallel task is written to its own buffer. Later, all buffers are flushed, in task order, to STDOUT.
a task id	only the task indicated writes output data to STDOUT.

If not set, the default is **unordered**. The value of this environment variable can be overridden using the **-stdoutmode** flag.

The following environment variables are associated with VT Trace Collection.

MP_SAMPLEFREQ Determines the interval (in milliseconds) at which AIX kernel statistics are sampled when executing a program with tracing on. Valid values are any integer greater than or equal to 0. If not set, the default is 10. The value of this environment variable can be overridden using the **-samplefreq** or **-sfreq** flags.

MP_TBUFFSIZE Determines the size (in bytes) of the buffer used when generating trace files. This may be specified as *nnn*K or *nnn*M. If not set, the default is 5M. The value of this environment variable can be overridden using the **-tbuffsize** or **-tbsize** flags.

MP_TBUFFWRAP	Determines that a wraparound storage approach for trace records should be used instead of the default three-tiered approach. With this approach, the system keeps overwriting the buffer instead of flushing it to a temp file. Valid values are either **yes** or **no**. If not set, the default is **no**. The value of this environment variable can be overridden using the **-tbuffwrap** or **-tbwrap** flags.
MP_TMPDIR	The temporary directory to which output trace files are written. Valid values are any path specifier. If not set, the default is /tmp/*username*. The value of this environment variable can be overridden using the **-tmpdir** flag.
MP_TRACEDIR	Determines the directory to which the final integrated trace file is built. Valid values are any path specifier. If not set, the default is the current directory. The value of this environment variable can be overridden using the **-tracedir** or **-tdir** flags.
MP_TRACEFILE	Determines the name of the output trace file created when executing a program with tracing on. Valid values are any file specifier. If not set, the default is the name of the program with the suffix *.trc* added. The value of this environment variable can be overridden using the **-tracefile** and **-tfile** flags.
MP_TRACELEVEL	Determines the level of VT tracing that should be generated during the execution of a program. Valid values are: **0** no trace records; **1** Application Markers; **2** AIX Kernel Statistic and Application Markers; **3** Message Passing, Collective Communication, and Application Markers; **9** all trace records. If not set, the default is **0** (no trace records). The value of this environment variable can be overridden using the **-tracelevel** or **-tlevel** flags.
MP_TTEMPSIZE	Determines the size (in bytes) of the temp file used when generating trace files. This may be specified as *nnn*M or *nnn*G. If not set, the default is 10M. The value of this environment variable can be overridden using the **-ttempsize** or **-ttsize** flags.

The following environment variables are associated with the generation of diagnostic information.

MP_INFOLEVEL Determines the level of message reporting. Valid values are:

0 error

1 warning and error

2 informational, warning, and error

3 informational, warning, and error. Also reports diagnostic messages for use by the IBM Support Center.

4, 5, 6 Informational, warning, and error. Also reports high- and low-level diagnostic messages for use by the IBM Support Center.

If not set, the default is **1** (warning and error). The value of this environment variable can be overridden using the **-infolevel** or **-ilevel** flags.

MP_PMDLOG Determines whether or not diagnostic messages should be logged to a file in */tmp* on each of the remote nodes. Typically, this environment variable/command-line flag is only used under the direction of the IBM Support Center in resolving a PE-related problem. Valid values are **yes** or **no**. If not set, the default is **no**. The value of this environment variable can be overridden using the **-pmdlog** flag.

MP_DEBUG_LOG Determines the level of diagnostic messages written to **$MP_tmp/dbelog.pid.taskid**. Typically, this environment variable/command-line flag is only used under the direction of the IBM Support Center in resolving a PE-related problem. This environment variable has no associated command-line flag.

MP_DEBUG_INITIAL_STOP
Determines the initial breakpoint in the application where **pdbx** or **pedb** will get control. **MP_DEBUG_INITIAL_STOP** should be specified as *file_name:line_number*. The *line_number* is the number of the line within the source file *file_name*; where *file_name* has been compiled with **-g**. The line number has to be one that defines executable code. In general, this is a line of code for which the compiler generates machine level code. Another way to view this is that the line number is one for which debuggers will accept a breakpoint. Another valid string for **MP_DEBUG_INITIAL_STOP** would be the *function_name* of the desired initial stopping point in the debugger. If this variable is not specified, the default is to stop at the first executable source line in the main routine. This environment variable has no associated command-line flag.

MP_PMDSUFFIX When using LoadLeveler, this environment variable determines a string to be appended to the normal partition manager daemon executable. The normal partition manager daemon executable specified is */etc/pmdv2*. By setting **MP_PMDSUFFIX**, you can append a string to *pmdv2*. If **MP_PMDSUFFIX** is set to *abc*, for example, then the partition manager daemon that gets run on each node of the parallel task is */etc/pmdv2abc*. When using the Resource Manager, this environment variable determines a string to be appended to the normal tcp service. The normal tcp service specified in */etc/services* is named *pmv2*. By setting **MP_PMDSUFFIX**, you can append a string to *pmv2*. If **MP_PMDSUFFIX** is set to *abc*, for example, then the service requested in */etc/services* is *pmv2abc*. Using the environment variable with LoadLeveler or the Resource Manager as described above permits testing of alternate versions of the Partition Manager daemon. Typically, this environment variable is only used under the direction of the IBM Support

Center in resolving a PE-related problem. Valid values are any string. This environment variable has no associated command-line flag.

The following environment variables are associated with the Message Passing Interface.

MP_BUFFER_MEM Changes the maximum size of memory used by the communication subsystem to buffer early arrivals. The default is 2.8 megabytes for IP and 64 megabytes for US. However, if checkpointing a program, for US the default will be 2.8 megabytes. If you are using this environment variable to change the maximum size of memory used by the communication subsystem while checkpointing a program, please be aware that the amount of space needed for the checkpointing files will be increased by the value of **MP_BUFFER_MEM**.

MP_CSS_INTERRUPT
Determines whether or not arriving message packets cause interrupts. This may provide better performance for certain applications. Valid values are **yes** and **no**. If not set, the default is **no**.

MP_EAGER_LIMIT Changes the threshold value for message size, above which rendezvous protocol is used.

MP_INTRDELAY Allows user programs to tune the delay parameter without having to recompile existing applications.

MP_MAX_TYPEDEPTH
Changes the maximum depth of message buffer types.

MP_USE_FLOW_CONTROL
Limits the maximum number of outstanding messages posted by a sender.

MP_THREAD_STACKSIZE
Determines the additional stacksize allocated for user programs executing on an MPI service thread. If you allocate insufficient space, the program may encounter a SIGSEGV exception.

MP_SINGLE_THREAD
Avoids mutex lock overheads in a single threaded program. This is an optimization flag, with values of **no** and **yes**. The default value is **no**, which means multiple user message passing threads are assumed.

Note: MPI-IO cannot be used if this is set to **YES**. Results are undefined if this is **YES**, with multiple message passing threads in use.

MP_WAIT_MODE Determines how a thread or task behaves when it discovers it is blocked, waiting for a message to arrive. Values are **poll**, **yield**, and **sleep**. The default mode for the signal handling library is **poll** for US, and **sleep** for IP.

MP_POLLING_INTERVAL
Defines the polling interval in microseconds. The maximum interval is approximately 2 billion microseconds (2000

	seconds). The default is 180,000 microseconds for IP, and 400,000 microseconds for US.

The following are miscellaneous environment variables:

MP_EUIDEVELOP	Determines whether or not the message passing interface performs more detailed checking during execution. This additional checking is intended for developing applications, and can significantly slow performance. Valid values are **yes** or **no**, **deb** (for “debug”), **nor** (for “normal”), and **min** (for “minimum”) . The **debug** and **min** values are used to start and stop parameter checking. If not set, the default is **no**. The value of this environment variable can be overridden using the **-euidevelop** flag.
MP_FENCE	Determines a *fence_string* to be used for separating options you want parsed by POE from those you do not. Valid values are any string, and there is no default. Once set, you can then use the *fence_string* followed by *additional_options* on the **poe** command line. The *additional_options* will not be parsed by POE. This environment variable has no associated command-line flag.
MP_NOARGLIST	Determines whether or not POE ignores the argument list. Valid values are **yes** and **no**. If set to **yes**, POE will not attempt to remove POE command-line flags before passing the argument list to the user's program. This environment variable has no associated command-line flag.
MP_PMLIGHTS	Indicates the number of lights displayed per row on the Program Marker Array.
MP_PRIORITY	Determines a dispatch priority adjustment class for execution. See *IBM Parallel Environment for AIX: Installation* for more information on dispatch priority classes. Valid values are any of the dispatch priority classes set up by the system administrator in the file */etc/poe.priority*. This environment variable has no associated command-line flag.
MP_USRPORT	Indicates the port id used by the Partition Manager to connect to the Program Marker Array. By default, the Partition Manager connects to the Array using a socket assigned to port 9999. If you get an error message indicating that the port is in use, specify a different port. Standard TCP/IP practice suggests using ports greater than 5000 and less than 10000.
MP_COREDIR	Creates a separate directory for each task's core file.

EXAMPLES

1. Assume the **MP_PGMMODEL** environment variable is set to **spmd**, and **MP_PROCS** is set to *6*. To load and execute the SPMD program *sample* on the six remote nodes of your partition, enter:

   ```
   poe sample
   ```

2. Assume you have an MPMD application consisting of two programs – *master* and *workers*. These programs are designed to run together and communicate

via calls to message passing subroutines. The program *master* is designed to run on one processor node. The *workers* program is designed to run as separate tasks on any number of other nodes. The **MP_PGMMODEL** environment variable is set to **mpmd**, and **MP_PROCS** is set to *6*. To individually load the six remote nodes with your MPMD application, enter:

```
poe
```

Once the partition is established, the **poe** command responds with the prompt:

```
0:host1_name>
```

To load the *master* program as task 0 on host1_name, enter:

```
master
```

The **poe** command responds with a prompt for the next node to load. When you have loaded the last node of your partition, the **poe** command displays the message `Partition loaded...` and begins execution.

3. Assume you want to run three SPMD programs – *setup*, *computation*, and *cleanup* – as job steps on the same partition of nodes. The **MP_PGMMODEL** environment variable is set to *spmd*, and **MP_NEWJOB** is set to **yes**. You enter:

```
poe
```

Once the partition is established, the **poe** command responds with the prompt:

```
Enter program name (or quit):
```

To load the program *setup*, enter:

```
setup
```

The program setup executes on all nodes of your partition. When execution completes, the **poe** command again prompts you for a program name. Enter the program names in turn. To release the partition, enter:

```
quit
```

4. To check the process status (using the non-parallel command **ps**) for all remote nodes in your partition, enter:

```
poe ps
```

FILES

host.list (Default host list file)

RELATED INFORMATION

Commands: **mpcc**(1), **mpcc_r**(1), **mpCC**(1), **mpCC_r**(1), **mpxlf**(1), **mpxlf_r**(1), **pdbx**(1), **pedb**(1), **pmarray**(1), **vt**(1), **xprofiler**(1)

---

poeauth

NAME

poeauth – Allows you to copy Distributed File System (DFS) credentials to all nodes on which you want to run POE jobs.

SYNOPSIS

poeauth [*POE options*]

DESCRIPTION

The **poeauth** command allows you to copy DFS credentials to all nodes on which you want to run POE jobs. You can use any POE command line flag or environment variable with **poeauth**, because it is a POE application. Before running **poeauth**, you must run **dce_login** from task 0 (where DFS/DCE credentials reside). The credentials must reside on task 0 for **poeauth** to copy them. In order to be properly authorized, you must run **poeauth** before running any POE applications. When the credentials are copied, there is no need to use **poeauth** until the credentials expire.

Return codes are:

| **Note:** The actual command return code value is 128 plus the unsigned return code value. That is, a return code of -2 will give a value of 130. For more information, see the “Exit Status” section in *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*

-1	error reading KRB5CCNAME environment variable
-3	credentials files not found on home node, task 0
-4	could not open credentials file for read
-5	no room on destination node's filesystem
-6	error opening file output file
-7	error creating output file
-8	error writing to output file
-9	MPI_Send of data failed
-10	Final MPI_Send failed
-11	MPI_Recv failed
-13	error creating unique id for credentials
-14	error opening /tmp/poedce_master file
-15	error creating /tmp/poedce_master file
-16	error writing to /tmp/poedce_master file

EXAMPLES

1. To copy the credentials to the first 32 nodes listed in the host list file named “my_hosts” in your home directory, enter:

   ```
   poeauth -procs 32 -hfile $HOME/my_hosts -labelio yes -pmdlog yes
   ```

Note: If you were previously set up to use a PMD suffix, remember to set the **MP_PMDSUFFIX** environment variable and export it before running **poeauth**.

RELATED INFORMATION

Commands: **poe**(1), **pdbx**(1), **pedb**(1)

poekill

NAME

poekill – terminates all remote tasks for a give program.

SYNOPSIS

poe poekill pgm_name [**poe_options**]

or

rsh remote_node poekill pgm_name

poekill is a Korn shell script that searches for the existence of running programs (named **pgm_name**) owned by the user, and terminates them via SIGTERM signals. If run under POE, **poekill** uses the standard POE mechanism for identifying the set of remote nodes (host.list, Resource Manager, etc.). If run under rsh, **poekill** applies only to the node specified as remote_node.

FLAGS

When run as a POE program, standard POE flags apply.

DESCRIPTION

poekill determines the user ID of the user that submitted the command. It then uses the ID to obtain a list of active processes, which is filtered by the **pgm_name** argument into a scratch file in /tmp. The file is processed by an awk script that sends a SIGTERM signal (15) to each process in the list, and echoes the action back to the user. The scratch file is then erased, and the script exits with code of 0.

If you do not provide a pgm_name, an error message is printed and the script exits with a code of 0.

The pgm_name can be a substring of the program name.

RELATED INFORMATION

Commands: **rsh**(1), **poe**(1), **kill**(1)

poestat

NAME

poestat – Starts the System Status Array, which is an X-Windows tool for monitoring the operational status and CPU utilization of processor nodes.

SYNOPSIS

poestat [**-norm**]

The **poestat** command starts the System Status Array. This X-Windows tool lets you quickly survey the utilization of processor nodes. Normally, this would be run in the background.

FLAGS

-norm	Indicates that a job management system , which would normally be used to select nodes for monitoring, is not available. Instead, the System Status Array program selects for monitoring each of the nodes on the LAN that have the VT Statistics Collector Daemon (*digd*) running. It also selects each of the nodes listed in the host list file indicated by the **MP_HOSTFILE** environment variable. If you are monitoring nodes of an RS/6000 network cluster, you must use this flag.

DESCRIPTION

The **poestat** command starts the System Status Array. This X-Windows monitoring tool lets you quickly survey the utilization of processor nodes. The Array consists of a number of squares, each representing a processor node of your SP system or cluster. The squares are colored pink and yellow to show the instantaneous percent of CPU utilization for each processor node. If a square were to appear all pink, the node would be at 0 percent utilization. If a square were to appear all yellow, it would be at 100 percent utilization. To the right of the Array is a node list which contains the name of each node shown in the Array. The nodes are listed in the order in which they were contacted, left to right, starting with the top row of the Array. You use this list to identify the name of a node represented in the Array.

In order to use this tool, the Visualization Tool's Statistics Collector Daemon process (*digd*) needs to be running on each of the nodes you wish to monitor. The daemon feeds the System Status Array with the CPU information it displays, and is created on each of your nodes as part of the Visualization Tool's installation procedure. The *digd* statistics collector daemon can also feed information to the Visualization Tool. If a square on the Array appears gray, the node is unavailable for monitoring. It either does not have the Statistics Collector daemon running, or the Array cannot communicate with it.

ENVIRONMENT VARIABLES

MP_HOSTFILE	This environment variable is normally associated with node allocation. However, it is also checked by the **poestat** command when running with the **-norm** option. It determines the name of a host list file to use in selecting nodes for monitoring. If not set, the default is *host.list* in your current directory.
MP_RESD	Identifies whether or not node data for allocated nodes received from a job management system (LoadLeveler or the SP system Resource Manager) should be filtered with a host list file. If **MP_RESD=yes**, and a host list file is specified, only those nodes allocated by the job management system and listed in the host list file will be displayed. This option is useful if you would like to view selected job management system allocated nodes or view up to 512 nodes. To view 512 nodes, create two host list files. Each file can contain up to 255 nodes. Set **MP_HOSTFILE** to one of the host list files and start **poestat** in the background. Set **MP_HOSTFILE** to the other host list file and start another **poestat** in the background. Two instances of **poestat** will be running, each displaying a different set of 255 nodes. If **MP_RESD=no**, the job management system node data will not be displayed (similar to the **-norm** command line option). For more information, see "Step 3e: Set the MP_RESD Environment Variable" on page 28.
SP_NAME	Identifies the control workstation of the SP system. This should only be used when all nodes to be monitored exist in a PSSP 2.3.0 or 2.4.0 partition. This is the only case that results in **poestat** contacting the Resource Manager rather than LoadLeveler for node allocation requests. When running **poestat** from a workstation that is external to the SP system, and using the Resource Manager, the *ssp.clients* fileset must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information).

EXAMPLES

To start the System Status Array as a background process when the SP system Resource Manager is available, enter:

```
poestat &
```

To start the System Status Array as a background process to monitor an RS/6000 network cluster, enter:

```
poestat -norm &
```

To display 512 resource manager nodes, enter:

```
export MP_RESD=yes

export MP_HOSTFILE= a host list file with 1st 255 nodes

poestat &

export MP_HOSTFILE= a host list file with 2nd 255 nodes

poestat &
```

FILES

host.list (Default host list file)

RELATED INFORMATION

Commands: **vt**(1)

Appendix B. POE Environment Variables and Command-Line Flags

This appendix contains tables which summarize the environment variables and command-line flags discussed throughout this book. You can set these variables and flags to influence the execution of parallel programs, and the operation of certain tools. The command-line flags temporarily override their associated environment variable. The tables divide the environment variables and flags by function.

- Table 8 on page 140 summarizes the environment variables and flags for controlling the Partition Manager. These environment variables and flags enable you to specify such things as an input or output host list file, and the method of node allocation. For a complete description of the variables and flags summarized in this table, see Chapter 2, “Executing Parallel Programs” on page 9.
- Table 9 on page 142 summarizes the environment variables and flags for Job Specifications. These environment variables and flags determine whether or not the Partition Manager should maintain the partition for multiple job steps, whether commands should be read from a file or STDIN, and how the partition should be loaded. For a complete description of the variables and flags summarized in this table, see Chapter 2, “Executing Parallel Programs” on page 9.
- Table 10 on page 142 summarizes the environment variables and flags for determining how I/O from the parallel tasks should be handled. These environment variables and flags set the input and output modes, and determine whether or not output is labeled by task id. For a complete description of the variables and flags summarized in this table, see “Managing Standard Input, Output, and Error” on page 44.
- Table 11 on page 143 summarizes the environment variables and flags controlling VT trace collection. These environment variables and flags determine such things as which types of traces are collected, and how trace storage is handled. For a complete description of the variables and flags summarized in this table, see *IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*
- Table 12 on page 144 summarizes the environment variables and flags for collecting diagnostic information. These environment variables and flags enable you to generate diagnostic information that may be required by the IBM Support Center in resolving PE-related problems.
- Table 13 on page 144 summarizes the environment variables and flags for the Message Passing Interface. These environment variables and flags allow you to change message and memory sizes, as well as other message passing information.
- Table 14 on page 146 summarizes some miscellaneous environment variables and flags. These environment variables and flags provide control for the Program Marker Array, enable additional error checking, and let you set a dispatch priority class for execution.

You can use the POE command-line flags on the **poe**, **pdbx**, and **pedb** commands. You can also use some of these flags on program names when individually loading nodes from STDIN or a POE commands file. The flags you can use are mainly those having to do with VT trace collection. They are:

- **-infolevel** or **-ilevel**
- **-ttempsize** or **-ttsize**
- **-tmpdir**
- **-samplefreq** or **-sfreq**
- **-tbuffwrap** or **-tbwrap**
- **-tbuffsize** or **-tbsize**
- **-euidevelop**

In the tables that follow, a check mark (√) denotes those flags you can use when individually loading nodes. For more information on individually loading nodes, refer to "Invoking an MPMD Program" on page 36.

Table 8 (Page 1 of 3). POE Environment Variables/Command-Line Flags for Partition Manager Control

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
| MP_ADAPTER_USE -adapter_use	How the node's adapter should be used. If using LoadLeveler, the US communication subsystem library does not require dedicated use of the SP switch on the node. Adapter use will be defaulted, as in Table 4 on page 25, but shared usage may be specified. If using the Resource Manager, this value is only used when POE is requesting non-specific nodes via the **MP_RMPOOL** or **-rmpool** setting.	One of the following strings: *dedicated* Only a single program task can use the adapter. *shared* A number of tasks on the node can use the adapter.	Dedicated for US jobs, shared for IP jobs.
MP_AUTH (no associated command line flag)	The type of user authorization. This value can be overridden with an entry in **/etc/poe.limits**.	One of the following strings: *AIX* AIX based user security. *DFS* DFS/DCE bsed user security.	AIX
| MP_CPU_USE -cpu_use	How the node's CPU should be used. If using LoadLeveler, the US communication subsystem library does not require unique CPU use on the node. CPU use will be defaulted, as in Table 4 on page 25, but multiple use may be specified. If using the Resource Manager, this value is only used when POE is requesting non-specific nodes via the **MP_RMPOOL** or **-rmpool** setting.	One of the following strings: *unique* Only your program's task can use the CPU. *multiple* Your program may share the node with other users.	Unique for US jobs, multiple for IP jobs.
MP_EUIDEVICE -euidevice	The adapter set to use for message passing – either Ethernet, FDDI, token-ring, or the IBM RS/6000 SP's high performance switch adapter.	One of the following strings: *en0* Ethernet *fi0* FDDI *tr0* token-ring *css0* high performance switch	The adapter set used as the external network address.
MP_EUILIB -euilib	The communication subsystem library implementation to use for communication – either the IP communication subsystem or the User Space (US) communication subsystem. Programs that use LAPI must set **MP_EUILIB** (or **-euilib**) to *us*. In order to use the US communication subsystem, you must have an SP system configured with its high performance switch feature.	One of the following strings: *ip* The IP communication subsystem. *us* The US communication subsystem. **Note:** This specification is case-sensitive.	ip
MP_EUILIBPATH -euilibpath	The path to the message passing and communication subsystem libraries. This only needs to be set if the libraries are moved.	Any path specifier.	*/usr/lpp/ppe.poe/lib*
MP_HOSTFILE -hostfile -hfile	The name of a host list file for node allocation.	Any file specifier or the word NULL.	*host.list* in the current directory.

Table 8 (Page 2 of 3). POE Environment Variables/Command-Line Flags for Partition Manager Control

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_PROCS -procs	The number of program tasks.	Any number from 1 to the maximum supported configuration.	1
MP_PULSE -pulse	The interval (in seconds) at which POE checks the remote nodes to ensure that they are actively communicating with the home node. **Note:** **Pulse** is ignored for **pdbx** and **pedb**.	An integer greater than or equal to 0.	600
MP_RESD -resd	Whether or not the Partition Manager should connect to a job management system (LoadLeveler or the Resource Manager) to allocate nodes. **Notes:** 1. **MP_RESD** only specifies whether or not to use a job management system. 2. When the Resource Manager is used, the actual system you are using is identified by the environment variable **SP_NAME**, of the control workstation on the SP system. 3. When running POE from a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the external node (see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation* for more information). 4. When running POE from a workstation that is external to the SP system, and using the Resource Manager, the *ssp.clients* fileset must be installed on the external node (see *IBM Parallel Environment for AIX: Installation* for more information).	*yes no*	None
MP_RETRY -retry	The period of time (in seconds) between processor node allocation retries if there are not enough processor nodes immediately available to run a program. This is only valid if you are using the SP system Resource Manager.	An integer greater than or equal to 0.	0 (no retry)
MP_RETRYCOUNT -retrycount	The number of times (at the interval set by **MP_RETRY**) that the Partition Manager should attempt to allocate processor nodes.	An integer greater than or equal to 0.	0
MP_MSG_API -msg_api	To indicate to POE which message passing API is being used by the parallel tasks. You need to set this environment variable if a parallel task is using LAPI alone or in conjunction with MPI. You do not need to set it if a parallel task is using MPI only.	MPI LAPI or MPI,LAPI	MPI
MP_RMPOOL -rmpool	With regard to LoadLeveler, the name or number of the pool that should be used for non-specific node allocation. With regard to the Resource Manager, the number of the SP system pool that should be used for non-specific node allocation. This environment variable/command-line flag only applies to LoadLeveler or the SP system Resource Manager.	An identifying pool number.	None
MP_NODES -nodes	To specify the number of physical nodes on which to run the parallel tasks. It may be used alone or in conjunction with **MP_TASKS_PER_NODE** and/or **MP_PROCS**, as described in Table 7 on page 33.	Any number from 1 to the maximum supported configuration.	None
MP_TASKS_PER_NODE -tasks_per_node	To specify the number of tasks to be run on each of the physical nodes. It may be used in conjunction with **MP_NODES** and/or **MP_PROCS**, as described in Table 7 on page 33, but may not be used alone.	Any number from 1 to the maximum supported configuration.	None
MP_SAVEHOSTFILE -savehostfile	The name of an output host list file to be generated by the Partition Manager.	Any relative or full path name.	None

Table 8 (Page 3 of 3). POE Environment Variables/Command-Line Flags for Partition Manager Control

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_REMOTEDIR (no associated command line flag)	The name of a script which echoes the name of the current directory to be used on the remote nodes.	Any file specifier.	None
MP_TIMEOUT (no associated command line flag)	The length of time that POE waits before abandoning an attempt to connect to the remote nodes.	Any number greater than 0.	150 seconds
MP_CHECKFILE (no associated command line flag)	The base name of the checkpoint file.	Any file specifier.	None
MP_CHECKDIR (no associated command line flag)	The directory where the checkpoint file will reside.	Any path specifier.	None

Table 9. POE Environment Variables/Command-Line Flags for Job Specification

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_CMDFILE -cmdfile	The name of a POE commands file used to load the nodes of your partition. If set, POE will read the commands file rather than STDIN.	Any file specifier.	None
MP_NEWJOB -newjob	Whether or not the Partition Manager maintains your partition for multiple job steps.	*yes no*	*no*
MP_PGMMODEL -pgmmodel	The programming model you are using.	*spmd mpmd*	*spmd*

Table 10 (Page 1 of 2). POE Environment Variables/Command-Line Flags for I/O Control

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_LABELIO -labelio	Whether or not output from the parallel tasks is labeled by task id.	*yes no*	*no* (*yes* for **pdbx**)
MP_STDINMODE -stdinmode	The input mode. This determines how input is managed for the parallel tasks.	*all* All tasks receive the same input data from STDIN. *none* No tasks receive input data from STDIN; STDIN will be used by the home node only. a task id STDIN is only sent to the task identified.	*all*
MP_HOLD_STDIN (no associated command line flag)	Whether or not sending of STDIN from the home node to the remote nodes is deferred until the message passing partition has been established.	*yes no*	*no*

Table 10 (Page 2 of 2). POE Environment Variables/Command-Line Flags for I/O Control

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_STDOUTMODE -stdoutmode	The output mode. This determines how STDOUT is handled by the parallel tasks.	One of the following: *unordered* All tasks write output data to STDOUT asynchronously. *ordered* Output data from each parallel task is written to its own buffer. Later, all buffers are flushed, in task order, to STDOUT. a task id Only the task indicated writes output data to STDOUT.	*unordered*

Table 11. POE Environment Variables/Command-Line Flags for VT Trace Collection

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_SAMPLEFREQ -samplefreq √ -sfreq √	The interval (in milliseconds) at which AIX kernel statistics are sampled when executing a program with tracing on.	An integer greater than or equal to 0.	10
MP_TBUFFSIZE -tbuffsize √ -tbsize √	The size (in bytes) of the buffer used when generating trace files.	*nnn*K *nnn*M	1M
MP_TBUFFWRAP -tbuffwrap √ -tbwrap √	A wraparound storage approach for trace records instead of the default three-tiered approach. With this approach, the system keeps overwriting the buffer instead of flushing it to a temp file.	*yes no*	no
MP_TMPDIR -tmpdir √	The temporary directory to which output trace files are written. **Note:** This environment variable/flag is also used by the **pdbx** and **pedb** commands to determine the directory to which individual startup files are written for each **dbx** task. This is where the **pedb** remote debug logfile is written.	Any path specifier.	/tmp/*username* /tmp (for **pedb** and **pdbx**)
MP_TRACEDIR -tracedir -tdir	The directory to which the final integrated trace file is built.	Any path specifier.	The current directory.
MP_TRACEFILE -tracefile -tfile	The name of the output trace file created when executing a program with tracing on.	Any file specifier.	The name of the program with the suffix *.trc* added.
MP_TRACELEVEL -tracelevel -tlevel	The level of VT tracing that should be generated during the execution of a program. **Note:** pedb automatically sets the trace level to 0.	One of the following integers: 0 No trace records 1 Application Markers 2 AIX Kernel Statistic and Application Markers 3 Message Passing, Collective Communication, and Application Markers 9 All trace records	0
MP_TTEMPSIZE -ttempsize √ -ttsize √	The size (in bytes) of the temp file used when generating trace files.	*nnn*M *nnn*G	10M
Note: √ symbol denotes flags you can use when individually loading nodes.			

Table 12. POE Environment Variables/Command-Line Flags for Diagnostic Information

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_INFOLEVEL -infolevel √ -ilevel √	The level of message reporting.	One of the following integers: 0 Error 1 Warning and error 2 Informational, warning, and error 3 Informational, warning, and error. Also reports high-level diagnostic messages for use by the IBM Support Center. 4, 5, 6 Informational, warning, and error. Also reports high- and low-level diagnostic messages for use by the IBM Support Center.	1
MP_PMDLOG -pmdlog	Whether or not diagnostic messages should be logged to a file in */tmp* on each of the remote nodes. Typically, this environment variable/command-line flag is only used under the direction of the IBM Support Center in resolving a PE-related problem.	*yes no*	*no*
MP_DEBUG_LOG (no associated command-line flag)	The level of diagnostic messages written to **$MP_TMPDIR/debug_log.pid.taskid**. Typically, this environment variable/command-line flag is only used under the direction of the IBM Support Center in resolving a PE-related problem. **Note: MP_DEBUG_LOG** is only valid for **pedb**.	*0 - 4*	*0*
MP_DEBUG_INITIAL_STOP (no associated command-line flag)	The initial breakpoint in the application where **pdbx** or **pedb** will get control.	One of the following: *"filename":line_number* *function_name*	The first executable source line in the main routine.
MP_PMDSUFFIX (no associated command-line flag)	When using LoadLeveler, to determine a string to be appended to the normal partition manager daemon executable. The normal partition manager daemon executable specified is */etc/pmdv2*. By setting **MP_PMDSUFFIX**, you can append a string to *pmdv2*. If **MP_PMDSUFFIX** is set to *abc*, for example, then the partition manager daemon that gets run on each node of the parallel task is */etc/pmdv2abc*. When using the Resource Manager, to determine a string to be appended to the normal tcp service. The normal tcp service specified in */etc/services* is named *pmv2*. By setting **MP_PMDSUFFIX**, you can append a string to *pmv2*. If **MP_PMDSUFFIX** is set to *abc*, for example, then the service requested in */etc/services* is *pmv2abc*. Using the environment variable with LoadLeveler or the Resource Manager as described above permits testing of alternate versions of the Partition Manager daemon. Typically, this environment variable is only used under the direction of the IBM Support Center in resolving a PE-related problem.	Any string.	None
Note: √ symbol denotes flags you can use when individually loading nodes.			

Table 13 (Page 1 of 2). POE Environment Variables/Command-Line Flags for Message Passing Interface (MPI)

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_MAX_TYPEDEPTH -max_typedepth	To change the maximum depth of message derived data types.	An integer greater than or equal to 1.	*5*

Table 13 (Page 2 of 2). *POE Environment Variables/Command-Line Flags for Message Passing Interface (MPI)*

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:

MP_EAGER_LIMIT -eager_limit	To change the threshold value for message size, above which rendezvous protocol is used.
	To ensure that at least 32 messages can be outstanding between any 2 tasks, **MP_EAGER_LIMIT** will be adjusted based on the number of tasks according to the following table (when **MP_EAGER_LIMIT** and **MP_BUFFER_MEM** have not been set by the user):
	`Number of Tasks    MP_EAGER_LIMIT`
	`===============    ==============`
	`1 - 16    4096`
	`17 - 32    2048`
	`33 - 64    1024`
	`65 - 128    512`
		An integer less than or equal to 65536.	*4KB*

MP_BUFFER_MEM -buffer_mem	To change the maximum size of memory used by the communication subsystem to buffer early arrivals.	An integer less than or equal to 64MB.	*2800000 bytes* (IP) *64MB* (US)
MP_INTRDELAY -intrdelay	To tune the delay parameter without having to recompile existing applications.	An integer greater than 0.	*35* µ (TB2) *1* µ (TB3)
MP_USE_FLOW_CONTROL -use_flow_control	To limit the maximum number of outstanding messages posted by a sender.	*yes no*	*no*
MP_CSS_INTERRUPT -css_interrupt	Whether or not arriving packets generate interrupts. This may provide better performance for certain applications.	*yes no*	*no*
MP_TIMEOUT (no associated command-line flag)	To change the length of time the communication subsystem will wait for a connection to be established during message passing initialization.	An integer greater than 0.	*150 seconds*
MP_THREAD_STACKSIZE -thread_stacksize	To specify the additional stacksize allocated for user programs executing on an MPI service thread. If you allocate insufficient space, the program may encounter a SIGSEGV exception.	*nnnnn nnnK nnM* (where K=1024 bytes and M=1024*1024 bytes)	None
MP_SINGLE_THREAD -single_thread	To avoid mutex lock overheads in a program which is known to be single threaded.	*no yes*	*no*
MP_WAIT_MODE -wait_mode	To specify how a thread or task behaves when it discovers it is blocked, waiting for a message to arrive.	*poll yield sleep*	*poll* for US *sleep* for IP
MP_POLLING_INTERVAL -polling_interval	To change the polling interval, in microseconds.	An integer between 1 and 2 billion.	*400,000* for US *180,000* for IP

Table 14. Other POE Environment Variables/Command-Line Flags

The Environment Variable/Command-Line Flag(s):	Set:	Possible Values:	Default:
MP_DBXPROMPTMOD (no associated command-line flag)	A modified **dbx** prompt. The **dbx** prompt \n*(dbx)* is used by the **pdbx** command as an indicator denoting that a **dbx** subcommand has completed. This environment variable modifies that prompt. Any value assigned to it will have a "." prepended and will then be inserted in the \n*(dbx)* prompt between the "x" and the ")". This environment variable is useful when the string \n*(dbx)* is present in the output of the program being debugged.	Any string.	None
MP_EUIDEVELOP -euidevelop √	Whether or not the message passing interface performs more detailed checking during execution. This additional checking is intended for developing applications, and can significantly slow performance. You can also start and stop parameter checking with *deb* (for "debug") and *min* (for "minimum").	*yes* (for "develop") *no* (for "normal") *deb* (for "debug") *min* (for "minimum")	*no*
MP_FENCE (no associated command-line flag)	A "fence" character string for separating arguments you want parsed by POE from those you do not.	Any string.	None
MP_NOARGLIST (no associated command-line flag)	Whether or not POE ignores the argument list. If set to *yes*, POE will not attempt to remove POE command-line flags before passing the argument list to the user's program.	*yes no*	*no*
MP_PMLIGHTS -pmlights	The number of lights displayed (per row) on the Program Marker Array.	An integer greater than or equal to 0.	0
MP_PRIORITY (no associated command-line flag)	A dispatch priority class for execution. See *IBM Parallel Environment for AIX: Installation* for more information on dispatch priority classes.	Any of the dispatch priority classes set up by the system administrator in the file */etc/poe.priority*.	None
MP_USRPORT -usrport	The port id used by the Partition Manager to connect to the Program Marker Array.	Any positive integer less than 32767. Standard TCP/IP practice suggests using ports greater than 5000 and less than 10000.	9999
MP_COREDIR -coredir	To create a separate directory for each task's core file.	Any valid directory name.	*coredir.taskid*

Glossary of Terms and Abbreviations

This glossary includes terms and definitions from:

- The *Dictionary of Computing*, New York: McGraw-Hill, 1994.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol (A) after the definition.
- The *ANSI/EIA Standard - 440A: Fiber Optic Terminology*, copyright 1989 by the Electronics Industries Association (EIA). Copies can be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue N.W., Washington, D.C. 20006. Definitions are identified by the symbol (E) after the definition.
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

This section contains some of the terms that are commonly used in the Parallel Environment books and in this book in particular.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard *Vocabulary for Information Processing* (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3. ANSI definitions are preceded by an asterisk (*).

Other definitions in this glossary are taken from *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems* (GC20-1699).

A

address. A value, possibly a character or group of characters that identifies a register, a device, a particular part of storage, or some other data source or destination.

AIX. Abbreviation for Advanced Interactive Executive, IBM's licensed version of the UNIX operating system. AIX is particularly suited to support technical computing applications, including high function graphics and floating point computations.

AIXwindows Environment/6000. A graphical user interface (GUI) for the RS/6000. It has the following components:

- A graphical user interface and toolkit based on OSF/Motif
- Enhanced X-Windows, an enhanced version of the MIT X Window System
- Graphics Library (GL), a graphical interface library for the applications programmer which is compatible with Silicon Graphics' GL interface.

API. Application Programming Interface.

application. The use to which a data processing system is put; for example, topayroll application, an airline reservation application.

argument. A parameter passed between a calling program and a called program or subprogram.

attribute. A named property of an entity.

B

bandwidth. The total available bit rate of a digital channel.

blocking operation. An operation which does not complete until the operation either succeeds or fails. For example, a blocking receive will not return until a message is received or until the channel is closed and no further messages can be received.

breakpoint. A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

broadcast operation. A communication operation in which one processor sends (or broadcasts) a message to all other processors.

buffer. A portion of storage used to hold input or output data temporarily.

C

C. A general purpose programming language. It was formalized by ANSI standards committee for the C language in 1984 and by Uniforum in 1983.

C++. A general purpose programming language, based on C, which includes extensions that support an object-oriented programming paradigm. Extensions include:

- strong typing
- data abstraction and encapsulation
- polymorphism through function overloading and templates
- class inheritance.

call arc. The representation of a call between two functions within the Xprofiler function call tree. It appears as a solid line between the two functions. The arrowhead indicates the direction of the call; the function it points to is the one that receives the call. The function making the call is known as the *caller*, while the function receiving the call is known as the *callee*.

chaotic relaxation. An iterative relaxation method which uses a combination of the Gauss-Seidel and Jacobi-Seidel methods. The array of discrete values is divided into sub-regions which can be operated on in parallel. The sub-region boundaries are calculated using Jacobi-Seidel, whereas the sub-region interiors are calculated using Gauss-Seidel. See also *Gauss-Seidel*.

client. A function that requests services from a server, and makes them available to the user.

cluster. A group of processors interconnected through a high speed network that can be used for high performance computing. It typically provides excellent price/performance.

collective communication. A communication operation which involves more than two processes or tasks. Broadcasts, reductions, and the MPI_Allreduce subroutine are all examples of collective communication operations. All tasks in a communicator must participate.

command alias. When using the PE command line debugger, pdbx, you can create abbreviations for existing commands using the **pdbx alias** command. These abbreviations are know as *command aliases*.

Communication Subsystem (CSS). A component of the IBM Parallel System Support Programs for AIX that provides software support for the High Performance Switch. It provides two protocols; IP (Internet Protocol) for LAN based communication and US (user space) as a message passing interface that is optimized for performance over the switch. See also *Internet Protocol* and *User Space*.

communicator. An MPI object that describes the communication context and an associated group of processes.

compile. To translate a source program into an executable program.

condition. One of a set of specified values that a data item can assume.

control workstation. A workstation attached to the RS/6000 SP that serves as a single point of control allowing the administrator or operator to monitor and manage the system using IBM Parallel System Support Programs for AIX.

core dump. A process by which the current state of a program is preserved in a file. Core dumps are usually associated with programs that have encountered an unexpected, system-detected fault, such as a Segmentation Fault, or severe user error. The current program state is needed for the programmer to diagnose and correct the problem.

core file. A file which preserves the state of a program, usually just before a program is terminated for an unexpected error. See also *core dump*.

current context. When using either of the PE parallel debuggers, control of the parallel program and the display of its data can be limited to a subset of the tasks that belong to that program. This subset of tasks is called the *current context*. You can set the current context to be a single task, multiple tasks, or all the tasks in the program.

D

data decomposition. A method of breaking up (or decomposing) a program into smaller parts to exploit parallelism. One divides the program by dividing the data (usually arrays) into smaller parts and operating on each part independently.

data parallelism. Refers to situations where parallel tasks perform the same computation on different sets of data.

dbx. A symbolic command line debugger that is often provided with UNIX systems. The PE command line debugger, **pdbx**, is based on the **dbx** debugger.

debugger. A debugger provides an environment in which you can manually control the execution of a program. It also provides the ability to display the program's data and operation.

distributed shell (dsh). An IBM Parallel System Support Programs for AIX command that lets you issue commands to a group of hosts in parallel. See the *IBM RISC System/6000 Scalable POWERparallel Systems: Command and Technical Reference* (GC23-3900-00) for details.

domain name. The hierarchical identification of a host system (in a network), consisting of human-readable labels, separated by decimals.

E

environment variable. 1. A variable that describes the operating environment of the process. Common environment variables describe the home directory, command search path, and the current time zone. 2. A variable that is included in the current software environment and is therefore available to any called program that requests it.

event. An occurrence of significance to a task; for example, the completion of an asynchronous operation such as an input/output operation.

Ethernet. Ethernet is the standard hardware for TCP/IP LANs in the UNIX marketplace. It is a 10 megabit per second baseband type network that uses the contention based CSMA/CD (collision detect) media access method.

executable. A program that has been link-edited and therefore can be run in a processor.

execution. To perform the actions specified by a program or a portion of a program.

expression. In programming languages, a language construct for computing a value from one or more operands.

F

fairness. A policy in which tasks, threads, or processes must be allowed eventual access to a resource for which they are competing. For example, if multiple threads are simultaneously seeking a lock, then no set of circumstances can cause any thread to wait indefinitely for access to the lock.

FDDI. Fiber distributed data interface (100 Mbit/s fiber optic LAN).

file system. In the AIX operating system, the collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

fileset. 1) An individually installable option or update. Options provide specific function while updates correct an error in, or enhance, a previously installed product. 2) One or more separately installable, logically grouped units in an installation package. See also *Licensed Program Product* and *package*.

foreign host. See *remote host*.

Fortran. One of the oldest of the modern programming languages, and the most popular language for scientific and engineering computations. It's name is a contraction of *FORmula TRANslation*. The two most common Fortran versions are Fortran 77, originally standardized in 1978, and Fortran 90. Fortran 77 is a proper subset of Fortran 90.

function call tree. A graphical representation of all the functions and calls within an application, which appears in the Xprofiler main window. The functions are represented by green, solid-filled rectangles called function boxes. The size and shape of each function box indicates its CPU usage. Calls between functions are represented by blue arrows, called call arcs, drawn between the function boxes. See also *call arcs*.

function cycle. A chain of calls in which the first caller is also the last to be called. A function that calls itself recursively is not considered a function cycle.

functional decomposition. A method of dividing the work in a program to exploit parallelism. One divides the program into independent pieces of functionality which are distributed to independent processors. This is in contrast to data decomposition which distributes the same work over different data to independent processors.

functional parallelism. Refers to situations where parallel tasks specialize in particular work.

G

Gauss-Seidel. An iterative relaxation method for solving Laplace's equation. It calculates the general solution by finding particular solutions to a set of discrete points distributed throughout the area in question. The values of the individual points are obtained by averaging the values of nearby points. Gauss-Seidel differs from Jacobi-Seidel in that for the i+1st iteration Jacobi-Seidel uses only values calculated in the ith iteration. Gauss-Seidel uses a mixture of values calculated in the ith and i+1st iterations.

global max. The maximum value across all processors for a given variable. It is global in the sense that it is global to the available processors.

global variable. A variable defined in one portion of a computer program and used in at least one other portion of the computer program.

gprof. A UNIX command that produces an execution profile of C, Pascal, Fortran, or COBOL programs. The execution profile is in a textual and tabular format. It is useful for identifying which routines use the most CPU time. See the man page on **gprof**.

GUI (Graphical User Interface). A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

H

High Performance Switch. The high-performance message passing network, of the RS/6000 SP(SP) machine, that connects all processor nodes.

HIPPI. High performance parallel interface.

hook. **hook** is a **pdbx** command that allows you to re-establish control over all task(s) in the current context that were previously unhooked with this command.

home node. The node from which an application developer compiles and runs his program. The home node can be any workstation on the LAN.

host. A computer connected to a network, and providing an access method to that network. A host provides end-user services.

host list file. A file that contains a list of host names, and possibly other information, that was defined by the application which reads it.

host name. The name used to uniquely identify any computer on a network.

hot spot. A memory location or synchronization resource for which multiple processors compete excessively. This competition can cause a disproportionately large performance degradation when one processor that seeks the resource blocks, preventing many other processors from having it, thereby forcing them to become idle.

I

IBM Parallel Environment for AIX. A program product that provides an execution and development environment for parallel Fortran, C, or C++ programs. It also includes tools for debugging, profiling, and tuning parallel programs.

installation image. A file or collection of files that are required in order to install a software product on a RS/6000 workstation or on SP system nodes. These files are in a form that allows them to be installed or removed with the AIX **installp** command. See also *fileset*, *Licensed Program Product*, and *package*.

Internet. The collection of worldwide networks and gateways which function as a single, cooperative virtual network.

Internet Protocol (IP). 1) The TCP/IP protocol that provides packet delivery between the hardware and user processes. 2) The High Performance Switch library, provided with the IBM Parallel System Support Programs for AIX, that follows the IP protocol of TCP/IP.

IP. See *Internet Protocol*.

J

Jacobi-Seidel. See *Gauss-Seidel*.

| **job management system**.

| The software you use to manage the jobs across your
| system, based on the availability and state of system
| resources.

K

Kerberos. A publicly available security and authentication product that works with the IBM Parallel System Support Programs for AIX software to authenticate the execution of remote commands.

kernel. The core portion of the UNIX operating system which controls the resources of the CPU and allocates them to the users. The kernel is memory-resident, is said to run in *kernel mode* (in other words, at higher execution priority level than *user mode*) and is protected from user tampering by the hardware.

L

Laplace's equation. A homogeneous partial differential equation used to describe heat transfer, electric fields, and many other applications.

latency. The time interval between the instant at which an instruction control unit initiates a call for data transmission, and the instant at which the actual transfer of data (or receipt of data at the remote end) begins. Latency is related to the hardware characteristics of the system and to the different layers of software that are involved in initiating the task of packing and transmitting the data.

Licensed Program Product (LPP). A collection of software packages, sold as a product, that customers pay for to license. It can consist of packages and filesets a customer would install. These packages and filesets bear a copyright and are offered under the terms and conditions of a licensing agreement. See also *fileset* and *package*.

| **LoadLeveler**. A job management system that works
| with POE to allow users to run jobs and match
| processing needs with system resources, in order to
| better utilize the system.

local variable. A variable that is defined and used only in one specified portion of a computer program.

loop unrolling. A program transformation which makes multiple copies of the body of a loop, placing the copies also within the body of the loop. The loop trip count and index are adjusted appropriately so the new loop computes the same values as the original. This transformation makes it possible for a compiler to take additional advantage of instruction pipelining, data cache effects, and software pipelining.

See also *optimization*.

M

menu. A list of options displayed to the user by a data processing system, from which the user can select an action to be initiated.

message catalog. A file created using the AIX Message Facility from a message source file that contains application error and other messages, which can later be translated into other languages without having to recompile the application source code.

message passing. Refers to the process by which parallel tasks explicitly exchange program data.

MIMD (Multiple Instruction Multiple Data). A parallel programming model in which different processors perform different instructions on different sets of data.

MPMD (Multiple Program Multiple Data). A parallel programming model in which different, but related, programs are run on different sets of data.

MPI. Message Passing Interface; a standardized API for implementing the message passing model.

N

network. An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

node. (1) In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network. (2) In terms of the RS/6000 SP, a single location or workstation in a network. An SP node is a physical entity (a processor).

node ID. A string of unique characters that identifies the node on a network.

nonblocking operation. An operation, such as sending or receiving a message, which returns immediately whether or not the operation was completed. For example, a nonblocking receive will not wait until a message is sent, but a blocking receive will wait. A nonblocking receive will return a status value that indicates whether or not a message was received.

O

object code. The result of translating a computer program to a relocatable, low-level form. Object code contains machine instructions, but symbol names (such as array, scalar, and procedure names), are not yet given a location in memory.

optimization. A not strictly accurate but widely used term for program performance improvement, especially for performance improvement done by a compiler or other program translation software. An optimizing compiler is one that performs extensive code transformations in order to obtain an executable that runs faster but gives the same answer as the original. Such code transformations, however, can make code debugging and performance analysis very difficult because complex code transformations obscure the correspondence between compiled and original source code.

option flag. Arguments or any other additional information that a user specifies with a program name. Also referred to as *parameters* or *command line options*.

P

package. A number of filesets that have been collected into a single installable image of program products, or LPPs. Multiple filesets can be bundled together for installing groups of software together. See also *fileset* and *Licensed Program Product*.

parallelism. The degree to which parts of a program may be concurrently executed.

parallelize. To convert a serial program for parallel execution.

Parallel Operating Environment (POE). An execution environment that smooths the differences between serial and parallel execution. It lets you submit and manage parallel jobs. It is abbreviated and commonly known as POE.

parameter. * (1) In Fortran, a symbol that is given a constant value for a specified application. (2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. (3) A name in a procedure that is used to refer to an argument that is passed to the procedure. (4) A particular piece of information that a system or application program needs to process a request.

partition. (1) A fixed-size division of storage. (2) In terms of the RS/6000 SP, a logical definition of nodes to be viewed as one system or domain. System partitioning is a method of organizing the SP into groups of nodes for testing or running different levels of software of product environments.

Partition Manager. The component of the Parallel Operating Environment (POE) that allocates nodes, sets up the execution environment for remote tasks, and manages distribution or collection of standard input (STDIN), standard output (STDOUT), and standard error (STDERR).

pdbx. **pdbx** is the parallel, symbolic command line debugging facility of PE. **pdbx** is based on the **dbx** debugger and has a similar interface.

PE. The IBM Parallel Environment for AIX program product.

performance monitor. A utility which displays how effectively a system is being used by programs.

POE. See Parallel Operating Environment.

pool. Groups of nodes on an SP that are known to the Resource Manager, and are identified by a number.

point-to-point communication. A communication operation which involves exactly two processes or tasks. One process initiates the communication through a *send* operation. The partner process issues a *receive* operation to accept the data being sent.

procedure. (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a procedure call. (2) A set of related control statements that cause one or more programs to be performed.

process. A program or command that is actually running the computer. It consists of a loaded version of the executable file, its data, its stack, and its kernel data structures that represent the process's state within a multitasking environment. The executable file contains the machine instructions (and any calls to shared objects) that will be executed by the hardware. A process can contain multiple threads of execution.

The process is created via a **fork**() system call and ends using an **exit**() system call. Between **fork** and **exit**, the process is known to the system by a unique process identifier (pid).

Each process has its own virtual memory space and cannot access another process's memory directly. Communication methods across processes include pipes, sockets, shared memory, and message passing.

prof. A utility which produces an execution profile of an application or program. It is useful to identifying which routines use the most CPU time. See the man page for **prof**.

profiling. The act of determining how much CPU time is used by each function or subroutine in a program. The histogram or table produced is called the execution profile.

Program Marker Array. An X-Windows run time monitor tool provided with Parallel Operating Environment, used to provide immediate visual feedback on a program's execution.

pthread. A thread that conforms to the POSIX Threads Programming Model.

R

reduction operation. An operation, usually mathematical, which reduces a collection of data by one or more dimensions. For example, the arithmetic SUM operation is a reduction operation which reduces an array to a scalar value. Other reduction operations include MAXVAL and MINVAL.

remote host. Any host on a network except the one at which a particular operator is working.

remote shell (rsh). A command supplied with both AIX and the IBM Parallel System Support Programs for AIX that lets you issue commands on a remote host.

Report. In Xprofiler, a tabular listing of performance data that is derived from the gmon.out files of an application. There are five types of reports that are generated by Xprofiler, and each one presents different statistical information for an application.

| **Resource Manager**. A server that runs on one of the
| nodes of an RS/6000 SP (SP) machine. It prevents
| parallel jobs from interfering with each other, and
| reports job-related node information.

RISC. Reduced Instruction Set Computing (RISC), the technology for today's high performance personal computers and workstations, was invented in 1975.

S

shell script. A sequence of commands that are to be executed by a shell interpreter such as C shell, korn shell, or Bourne shell. Script commands are stored in a file in the same form as if they were typed at a terminal.

segmentation fault. A system-detected error, usually caused by referencing an invalid memory address.

server. A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server.

signal handling. A type of communication that is used by message passing libraries. Signal handling involves using AIX signals as an asynchronous way to move data in and out of message buffers.

source line. A line of source code.

source code. The input to a compiler or assembler, written in a source language. Contrast with object code.

SP. RS/6000 SP; a scalable system from two to 128 processor nodes, arranged in various physical configurations, that provides a high powered computing environment.

SPMD (Single Program Multiple Data). A parallel programming model in which different processors execute the same program on different sets of data.

standard input (STDIN). In the AIX operating system, the primary source of data entered into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output (STDOUT). In the AIX operating system, the primary destination of data produced by a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can go to a file or to another command.

stencil. A pattern of memory references used for averaging. A 4-point stencil in two dimensions for a given array cell, x(i,j), uses the four adjacent cells, x(i-1,j), x(i+1,j), x(i,j-1), and x(i,j+1).

subroutine. (1) A sequence of instructions whose execution is invoked by a call. (2) A sequenced set of instructions or statements that may be used in one or more computer programs and at one or more points in a computer program. (3) A group of instructions that can be part of another routine or can be called by another program or routine.

synchronization. The action of forcing certain points in the execution sequences of two or more asynchronous procedures to coincide in time.

system administrator. (1) The person at a computer installation who designs, controls, and manages the use of the computer system. (2) The person who is responsible for setting up, modifying, and maintaining the Parallel Environment.

System Data Repository. A component of the IBM Parallel System Support Programs for AIX software that provides configuration management for the SP system. It manages the storage and retrieval of system data across the control workstation, file servers, and nodes.

System Status Array. An X-Windows run time monitor tool, provided with the Parallel Operating Environment, that lets you quickly survey the utilization of processor nodes.

T

task. A unit of computation analogous to an AIX process.

thread. A single, separately dispatchable, unit of execution. There may be one or more threads in a process, and each thread is executed by the operating system concurrently.

tracing. In PE, the collection of data for the Visualization Tool (VT). The program is *traced* by collecting information about the execution of the program in trace records. These records are then accumulated into a trace file which a user visualizes with VT.

tracepoint. Tracepoints are places in the program that, when reached during execution, cause the debugger to print information about the state of the program.

trace record. In PE, a collection of information about a specific event that occurred during the execution of your program. For example, a trace record is created for each send and receive operation that occurs in your program (this is optional and may not be appropriate). These records are then accumulated into a trace file which allows the Visualization Tool to visually display the communications patterns from the program.

U

unrolling loops. See *loop unrolling*.

US. See *user space*.

user. (1) A person who requires the services of a computing system. (2) Any person or any thing that may issue or receive commands and message to or from the information processing system.

user space (US). A version of the message passing library that is optimized for direct access to the SP High Performance Switch, that maximizes the performance capabilities of the SP hardware.

utility program. A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program.

utility routine. A routine in general support of the processes of a computer; for example, an input routine.

V

variable. (1) In programming languages, a named object that may take different values, one at a time. The values of a variable are usually restricted to one data type. (2) A quantity that can assume any of a given set of values. (3) A name used to represent a data item whose value can be changed while the program is running. (4) A name used to represent data whose value can be changed, while the program is running, by referring to the name of the variable.

view. (1) In an information resource directory, the combination of a variation name and revision number that is used as a component of an access name or of a descriptive name.

Visualization Tool. The PE Visualization Tool. This tool uses information that is captured as your parallel program executes, and presents a graphical display of the program execution. For more information, *see IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference*

VT. See *Visualization Tool*.

X

X Window System. The UNIX industry's graphics windowing standard that provides simultaneous views of several executing programs or processes on high resolution graphics displays.

xpdbx. This is the former name of the PE graphical interface debugging facility, which is now called **pedb**.

Xprofiler. An AIX tool that is used to analyze the performance of both serial and parallel applications, via a graphical user interface. Xprofiler provides quick access to the profiled data, so that the functions that are the most CPU-intensive can be easily identified.

Index

Communicating Your Comments to IBM

IBM Parallel Environment for AIX
Operation and Use, Volume 1
Using the Parallel Operating Environment
Version 2 Release 4

Publication No. SC28-1979-02

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a reader's comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - FAX: (International Access Code)+1+914+432-9405
- If you prefer to send comments electronically, use this network ID:
  - IBM Mail Exchange: USIB6TC9 at IBMMAIL
  - Internet e-mail: mhvrcfs@us.ibm.com
  - World Wide Web: http://www.s390.ibm.com/os390

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

Reader's Comments — We'd Like to Hear from You

IBM Parallel Environment for AIX
Operation and Use, Volume 1
Using the Parallel Operating Environment
Version 2 Release 4

Publication No. SC28-1979-02

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: ____________________________________

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ]	As an introduction	[ ]	As a text (student)
[ ]	As a reference manual	[ ]	As a text (instructor)
[ ]	For another purpose (explain)		

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Name	Address
Company or Organization	
Phone No.	

Reader's Comments — We'd Like to Hear from You
SC28-1979-02

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie NY 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

SC28-1979-02

Cut or Fold
Along Line

Program Number: 5765-543

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC28-1979-02