Process Management Programmer's Guide

900 Series HP 3000 Computer Systems

HP Part No. 32650-90023
Printed in USA 19871101

E1187

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

Printing History

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

Edition	Date	Software Version
First Edition	November 1987	A.01.00

Preface

Process Management Programmer's Guide (32650-90023) is written for an experienced programmer who has a working knowledge of MPE/iX and is familiar with:

- A text editor
- At least one programming language
- Compiling, linking, and executing a program on MPE/iX

This manual contains detailed instructions describing how you can improve the performance of your program when you use system intrinsics that take advantage of MPE/iX process management capabilities. Intrinsics are available to create processes, manage processes, and delete processes.

This manual also describes how you can manage interactive sessions on MPE/iX by using system intrinsics to programmatically create and abort sessions.

This manual is part of the MPE/iX Programmer's Series that consists of the *MPE XL Intrinsics Reference Manual* (32650-90028) and a set of task-oriented programmer's guides. Refer to the MPE/iX Programmer's Series Documentation Map for an illustration of how this manual relates to the rest of the series.

This manual contains the following chapters:

Chapter One **Processes and Process Management** describes processes and process management under MPE/iX, and introduces system intrinsics that enable your program to perform process management tasks. This chapter also introduces session management intrinsics available on MPE/iX.

Chapter Two **Process Management Tasks** describes various ways you can use MPE XL process management intrinsics to create, activate, suspend, interrogate, and delete processes in MPE/iX.

Chapter Three **Deadlock Considerations** describes the conditions that can cause deadlock within your program or between different jobs/sessions.

Chapter Four **Managing a Session Programmatically** describes how you can programmatically create or abort an interactive session.

Contents

Tables

1

Processes and Process Management

In the MPE/iX family of computer systems, the process is the basic executable entity that competes for the resources managed by MPE/iX. The notion of many active processes sharing the resources of the computer system is fundamental to the efficient operation of the MPE/iX operating system. By sharing the CPU and system resources among multiple active processes, resource utilization is greatly increased and system throughput improved. *Process Management Programmer's Guide* (32650-90023) provides you with information that will assist you in your development of programs that utilize process management capabilities available in the MPE/iX operating system.

This chapter describes processes and process management under MPE/iX, and introduces process management capabilities and system intrinsics that MPE/iX makes available to you.

This chapter also describes the interactive session as a system-managed process, and introduces session management capabilities and system intrinsics that enable your program to create and abort interactive sessions.

What is a Process?

A process is the basic executable entity in the MPE/iX operating system. When you enter the following command:

```
:RUN PROGNAME.PUB.MYACCT
```

you are directing MPE/iX to create and manage a process that executes the code found in your program file PROGNAME.

A process, then, can be described as a program and its environment. The program consists of re-entrant and shared executable code. The environment, created and managed by MPE/iX, includes all resources that the program may reference or use that are exterior to the program. Resources may be main memory, I/O devices, and even other processes. In addition, MPE/iX must keep a record of the current state of the process and what is currently in the environment.

Process Components

A process in the MPE/iX operating system consists of sharable executable code, a private data area available to the process, and a data structure commonly referred to as a process control block that defines the process and contains pointers to information required for process management.

Process Identification

MPE/iX assigns each process a unique Process Identification Number (PIN) when the process is created. Both you and the system can use the PIN to identify a process throughout its life span. When the process is finally deleted from the system, the PIN is released, and can be used again to identify another process.

Life Cycle of a Process

A process exists in different states according to its past and present status and its present requirements for system resources. MPE/iX allows the user some control over a process' movement between three of these states.

- A process in a suspended state is not allowed control of the CPU until it receives an activation signal from a system intrinsic. When it suspends itself, a process must specify the process or processes that are permitted to reactivate it.
- A process in an active state is scheduled to gain control of the CPU (awaiting its turn to enter an executing state).
- A process in an executing state has control of the CPU. It leaves this state when it has used up its scheduled quantum of time (it enters an active state), when it is blocked or pre-empted by an interrupt, when it suspends itself (it enters a suspended state), or when it is deleted from the system.

During its life span (that is, between creation and deletion), a process progresses through these states in response to the following:

- The instruction sequence found in the program code
- The actions of other processes

Organization of Processes

The MPE family of operating systems is process-oriented in that the operating system deals almost exclusively with processes. Because MPE/iX provides a multiprogramming environment, many processes can share the CPU and other system resources to maximize system utilization, thus increasing system performance.

The organization of processes in MPE/iX is a directed rooted tree structure whose nodes correspond to processes. An arrow from one process P2 to another process (P3) means that P2, the parent process, is the creator of P3, the child process. The parent/child relationship between processes provides control from top to bottom throughout the process structure. In most cases, the parent process is responsible for what happens to a child process, including creation, deletion, and other special actions.

A process may have the capability to create one or more child processes. A process cannot exist without a parent process (the one exception being PROGEN, the first process created by MPE/iX and the ancestor of all other processes created on MPE/iX).

System Processes

PROGEN, the Progenitor, is the first process created during the initialization phase of MPE/iX. PROGEN creates child processes from a set of data specified at system configuration time. These child processes are system processes that provide parallel functions on behalf of the operating system. The system processes that perform operating system functions can each have their own structure of descendants (children, grandchildren, and so forth).

The SESSION and JOB Processes

In MPE/iX, a session is associated with an interactive access to the system, while a job is associated with a batch access. If PROGEN can be considered to be the ancestor of all processes in MPE/iX, then the system process SESSION is the common ancestor of all interactive sessions in the system. When you press a RETURN on an unowned terminal, a message is sent to SESSION. SESSION directs a JSMAIN process to allocate and initialize session resources for the new session. JSMAIN then creates and activates a Command Interpreter (CI) process, commonly referred to as the Root CI.

The system process JOB is the common ancestor of all jobs on the system. An attempt to start a job results in a message being sent to JOB. JOB directs a JSMAIN process to accomplish its task of establishing the environment for running batch jobs in the MPE/iX operating system.

The Command Interpreter Process

The CI is a process that provides you with interactive access to the MPE/iX operating system. A CI (or the Root CI) is created for each session logged on to MPE/iX. Your CI reads the commands you enter on the keyboard as data, places the data on its stack, and acts upon the data accordingly. For example, when you enter the following command:

```
:RUN EDITOR.PUB.SYS
```

you are directing your CI to create a child process to execute the code found in the program file EDITOR.PUB.SYS. Your CI is suspended until it is reactivated by the child process (when you exit EDITOR or enter BREAK mode).

Your CI is limited to having one child process at a time. Following the example given in Example 1-1 , if you are in EDITOR (the child process of your Root CI) and press the BREAK key, EDITOR is suspended and your Root CI is reactivated (indicated by the colon prompt appearing on your terminal). If you try to create a second child process with the command `:RUN MYPROG.PUB.MYACCT`, you are prompted with `ABORT? (YES/NO)` and must direct your CI either to abort EDITOR or to ignore the command.

If you respond with `NO` to the `ABORT? (YES/NO)` prompt, your CI will ignore your command to create a second process. If you respond `YES` with `ABORT? (YES/NO)` prompt, your CI first deletes from the system the process executing the EDITOR program, then the system proceeds to create the process that executes `MYPROG`.

```
:RUN EDITOR.PUB.SYS
HP32201A.07.17 EDIT/3000 WED, SEP 24, 1987 12:55 PM
© HEWLETT-PACKARD CO. 1985
\ [BREAK]
:RUN MYPROG.PUB.MYACCT
ABORT? (YES/NO) NO
COMMAND NOT ALLOWED IN BREAK.  (CIWARN 986)
: RUN MYPROG.PUB.MYACCT
ABORT? (YES/NO) YES

PROGRAM ABORTED PER USER REQUEST.  (CIERR 989)
: RUN MYPROG.PUB.MYACCT
                  .
                  .
                  .
```

Example 1-1. Attempting to Create Multiple Child Process from Your CI

As illustrated in Example 1-1, you are restricted to creating only one child process at a time while in the CI. Processes that you create from your Root CI are normally restricted from creating child processes because of the great amount of system overhead required to manage each process. If you take advantage of the special capability to create multiple child processes, you must take great care that the benefits gained by creating multiple processes to perform a set of tasks (increased program performance and throughput) are not offset by the liabilities caused by the additional processes contending for the CPU and other system resources (decreased system performance and response time).

User Processes

MPE/iX enables you to improve the performance of your program when you use system intrinsics that take advantage of MPE/iX process management capabilities. You can develop a large and/or complex program that creates multiple active processes to perform a set of tasks. The concurrent accomplishment of tasks (versus serial accomplishment) can greatly increase your program's performance and throughput.

Managing User Processes

If you have standard MPE/iX capabilities (for example, IA, BA, SF), you can use many system intrinsics to access operating system features, but you are allowed only limited programmatic control over processes. For example, your program cannot use system intrinsics to create a new process.

If you wish to design large and/or complex applications that use process management capabilities to full advantage, MPE/iX provides Process Handling (PH) as an optional capability.

PH intrinsics require that the operating system check for additional capabilities at program load time and/or execution time. To execute any of the PH intrinsics, you must have the correct capability assigned prior to running the program.

In order for a program containing PH intrinsics to execute successfully, the following criteria must be met:

- A System Manager (SM) or Account Manager (AM) must assign the PH capability to the group where the program is to execute; however, if the program file is a temporary file, PH must be assigned to the user who executes the program.
- You must assign the PH capability to the program file at link time (using the caplist parameter of the `:LINK` command). The user who links the program does not need to be assigned the PH capability in order to assign it to the program file.

MPE/iX provides system intrinsics that allow you to create, activate, suspend, delete, interrogate, and provide communication between processes. Table 1-1 lists the major tasks involved with managing processes and the intrinsics required to accomplish each of the tasks. Also noted is whether or not the intrinsic requires PH capability. Chapter 2 “Process Management Tasks”, describes how you can use the intrinsics listed in Table 1-1.

Table 1-1. Process Management Tasks/Intrinsics

To Accomplish This Task:	Use these Intrinsics:	PH required?
Create a process	CREATEPROCESS	YES
	CREATE	YES
Activate a suspended process	ACTIVATE	YES
	CREATEPROCESS	YES
	SUSPEND	YES
Suspend an executing process	SUSPEND	YES
	ACTIVATE	YES
	CREATEPROCESS	YES
	PAUSE	NO
	CAUSEBREAK	NO
Delete a process	TERMINATE	NO
	KILL	YES
	QUIT	NO
	QUITPROG	NO
Interrogate a process	FATHER	YES
	GETPROCID	YES
	GETORIGIN	YES
	GETPROCINFO	YES
	PROCTIME	NO
	PROCINFO	NO
	GETINTO	NO
	GETPRIORITY	YES
Provide communication between process	HPFOPEN, FOPEN	NO
	FREAD,FWRITE	NO
	MAIL	YES
	RECEIVEMAIL	YES
	SENDMAIL	YES

Note If you are interested in the task of providing communication between processes, refer to *Interprocess Communication Programmer's Guide* (32650-90019).

Managing a Session Programmatically

Normally, if you wish to create a session for yourself, you must be at a terminal and log on to the system by entering the `:HELLO` command, then pressing RETURN . A message is sent to the system SESSION process. SESSION directs a system JSMAIN process to allocate and initialize session resources for the new session. JSMAIN then create and activates a CI process.

Likewise, when you wish to exit the interactive session, you must be at the terminal and log off by entering the `:BYE` command at the colon prompt, then pressing RETURN. This action directs MPE/iX to delete your sessions from the system.

Because an interactive session is managed by the system, your program cannot use PH intrinsics to create, activate, and delete an interactive session. Instead, MPE/iX provides you with two additional intrinsics that allow you a very limited form of programmatic control over an interactive session:

- The `STARTSESS` intrinsic enables your program to create a session for yourself, or for other users, on an available (not currently assigned to a session) terminal without having to be at the specified terminal(s). Your SM or AM must assign the Programmatic Sessions (PS) capability to any user who plans to run a program that calls the `STARTSESS` intrinsic.
- The `ABORTSESS` intrinsic enables your program to remove (abort) an interactive session from your system.

A discussion on how you use `STARTSESS` and `ABORTSESS` to manage interactive sessions can be found in Chapter 4, "Managing a Session Programmatically".

2

Process Management Tasks

This chapter explains how you can use MPE/iX intrinsics to:

- Create a new process
- Activate a suspended process
- Suspend an executing process
- Delay a process from executing
- Request a process break
- Delete a process
- Interrogate a process
- Pass information to a process
- Reschedule a process
- Provide communication between processes

For complete descriptions of the intrinsics described in this chapter, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

This chapter describes how your program can call two kinds of process management intrinsics:

- Intrinsics requiring only standard capabilities, (for example, IA, BA, SF)
- Intrinsics requiring PH capability in addition to standard capabilities

Assigning Process Handling (PH) Capability

If you have standard MPE/iX capabilities, you can use many system intrinsics to access operating system features. However, PH intrinsics require that the operating system check for additional capabilities at program load and/or execution. To execute PH intrinsics, you must have the correct capability assigned prior to running the program.

In order for a program containing PH intrinsics to execute successfully, the following criteria must be met:

- You must assign the PH capability to the program file at link time (using the *caplist* parameter of the `:LINK` command). The user who links the program does not need to be assigned the PH capability in order to assign it to the program file.
- The System Manager (SM) or Account Manager (AM) must assign the PH capability to the group where the program is to execute; however, if the program file is a temporary file, PH must be assigned to the user who executes the program.

Creating a Process (PH Capability Required)

If your program has PH capability, it can call the CREATEPROCESS intrinsic or the CREATE intrinsic to create a child process. Both intrinsics direct MPE/iX to:

- Initialize resources
- Allocate a private data area (including a stack)
- Load the specified program into virtual memory
- Assign a Process Identification Number (PIN)
- Place the newly created child process in a suspended state.

CREATEPROCESS is the recommended intrinsic for process creation because it is designed to be more flexible and extendable than the CREATE intrinsic.

Using CREATEPROCESS

Your program can call the CREATEPROCESS intrinsic to create a child process. Your program can control the creation of the child process with the optional parameters *itemnums* and *items*. The information you can pass to MPE/iX through *itemnums/items* parameter pairs include:

- The names of the files to be used as $STDIN and $STDLIST for the child process.
- An option that indicates if the child process is to be activated immediately after it is created, and if the parent process should be suspended automatically when the child process is activated.
- A list of user-named executable libraries to be searched at load time for required external references.
- A user-created UNSAT procedure, to which all unsatisfied load-time external references may be directed.

This is an example of a CREATEPROCESS intrinsic call:

```
        .
        .
        .
   ERRORCODE := 0;
   PIN := 0;
   BNAME := 'MYPROG.PUB.MYACCT';
   ITEMNUMS[1] := 3;
   ITEMNUMS[2] := 0;
   ITEMS[1] := 1;
   CREATEPROCESS (ERRORCODE,PIN,BNAME,ITEMNUMS,ITEMS);
        .
        .
        .
```

The parameters specified in the example above are described below.

ERRORCODE	Returns a value that indicates the success or failure of the CREATEPROCESS call, and, if unsuccessful, the nature of the error.
PIN	Returns the PIN of the newly created child process.

BNAME	Passes the name of the program file the child process is to execute.
ITEMNUMS	Passes the value 3 in the first element of the array indicating that load option information is passed in the first element of the array ITEMS. The value 0 in the second element indicates the end of the option list.
ITEMS	Passes the value 1 in the first element of the array indicating that the calling process is to be reactivated when the child process is deleted.

Using CREATE

Because the CREATE intrinsic allows your program only limited control over the creation of a child process, it offers a subset of the parameters available with the CREATEPROCESS intrinsic.

This is an example of a CREATE intrinsic call:

```
        .
        .
        .
   BNAME := 'MYPROG.PUB.MYACCT';
   PIN := 0;
   LOADFLAGS := 1;
   CREATE (BNAME,,PIN,,LOADFLAGS);
        .
        .
        .
```

The parameters specified in the example above are described below. All other optional parameters are omitted.

BNAME	Passes the name of the program file the child process executes.
PIN	Returns the Process Identification Number (PIN) of the child process.
LOADFLAGS	Passes load option information. The value 1 specifies that the calling process is reactivated when the child process is deleted.

Activating a Process (PH Capability Required)

If your program has PH capability, it can call the ACTIVATE intrinsic to activate a process that has been previously suspended (or just created) by the actions of these intrinsics:

- SUSPEND
- ACTIVATE
- CREATEPROCESS
- CREATE

The ACTIVATE intrinsic moves the specified child or parent process from a suspended state to an active state. . In addition, the ACTIVATE intrinsic optionally enables your program to suspend itself as soon as it activates the specified process.

Your program can only activate the parent process or a child process. Also, your program must have permission to activate the specified process (refer to the discussion of suspending a process). For example, only the parent process can activate a newly created child process.

This is an example of an ACTIVATE intrinsic call:

```
        .
        .
        .
SUSP := 2;
CREATE (BNAME,,PIN,,FLAGS);
ACTIVATE (PIN,SUSP);
        .
        .
        .
```

The parameters specified in the example above are described below.

PIN	Passes the PIN of the child process created by CREATE.
SUSP	Passes activation information. A non-zero value specifies that the calling process is to be suspended when the process specified by PIN is activated. The value 2 specifies that only a child process is permitted to reactivate the suspended process.

Suspending a Process (PH Capability Required)

If your program has PH capability, it can call the SUSPEND intrinsic to suspend itself. In addition, the CREATEPROCESS and ACTIVATE intrinsics also enable your program to suspend itself (explained below).

When a process suspends itself, it moves from an executing state to a suspended state. The process remains suspended until:

- Another process calls the ACTIVATE intrinsic to reactivate it.
- Another process (or MPE/iX) deletes it from the system.

When your program suspends itself, it must specify which processes have permission to reactivate it. You use the *susp* parameter to specify one of the following three conditions:

- Only the parent process has permission to reactivate the suspended process.
- Any child process has permission to reactivate the suspended process.
- The parent process or any child process has permission to reactivate the suspended process.

Your program can optionally unlock a local Resource Identification Number (RIN) with the *rin* parameter of SUSPEND if it has the local RIN currently locked. Refer to the discussion of managing shared resources with RINs in *Resource Management Programmer's Guide* (32650-90024).

This is an example of a SUSPEND intrinsic call:

```
        .
        .
```

```
      .
SUSP := 3;
RIN := 3;
SUSPEND (SUSP,RIN);
      .
      .
      .
```

The parameters specified in the example above are described below.

SUSP	Passes activation information. The value 3 specifies that the parent process and all child processes have permission to reactivate the process.
RIN	Passes a local RIN (3) that is unlocked when the process is suspended.

In addition, your program can suspend itself using:

- The *susp* parameter of the ACTIVATE intrinsic
- Item number 10 of the CREATEPROCESS intrinsic

When a non-zero value is specified for either parameter, your program is suspended when the specified process is activated. The value also indicates what processes have permission to reactivate your program after it had been suspended (in the same manner as the *susp* parameter of SUSPEND).

Delaying a Process

Your program can call the PAUSE intrinsic to delay its own execution. The PAUSE intrinsic places the process in a suspended state. and keeps it there for the number of seconds you specify in the *interval* parameter. At the end of the specified interval, the process re-enters an active state. The maximum interval allowed is approximately 2,147,484 seconds (almost 25 days).

The following is an example of a PAUSE intrinsic call:

```
PAUSE (INTERVAL);
```

INTERVAL specifies the length of time, in seconds, that you want the process to remain suspended. MPE/iX resumes execution with the statement following the PAUSE intrinsic call.

Requesting a Process Break

During a session, your program can initiate a break programmatically with the CAUSEBREAK intrinsic. The CAUSEBREAK intrinsic performs the programmatic equivalent of pressing BREAK in a session. MPE/iX suspends the entire user process structure (below the Root CI process) and activates the Root CI.

The following is an example of a CAUSEBREAK intrinsic call:

```
CAUSEBREAK;
```

While you are in BREAK mode, MPE/iX permits you to use a limited number of CI commands to perform functions such as creating a file or transmitting a message. (Refer to the *MPE/iX Commands Reference Manual* (32650-90003) for a discussion of commands available in BREAK mode.) When you execute the :RESUME command, MPE/iX returns all user processes to the state they were in before the CAUSEBREAK intrinsic call executed.

Note You are not permitted to use the CAUSEBREAK intrinsic in a job. Also, a program containing the CAUSEBREAK intrinsic will not break if it is executed from a UDC in which the NOBREAK option is specified.

Deleting Processes

MPE/iX provides intrinsics that enable your program to delete itself and/or other processes. You may wish to delete a process because:

- It has accomplished its task (use the TERMINATE or KILL intrinsics).
- It has reached an error condition (use the QUIT or QUITPROG intrinsics).

When your program directs MPE/iX to delete a process, MPE/iX uses as algorithm to delete a process. When your program deletes a process, MPE/iX also searches for and deletes all of the process's descendants (children, grandchildren, and so forth). In addition, the operating system accomplishes the following tasks:

- All files opened by the process are closed and assigned the same disposition they had when opened (as if the FCLOSE intrinsic had been called with a disposition of zero).
- All system resources reserved for that process (including the data area, process control block, and PIN) are returned to MPE/iX and made available for use by other processes.

The following sections describe how your program can delete itself and/or another process using these intrinsics:

- TERMINATE
- KILL
- QUIT
- QUITPROG

Deleting a Process with TERMINATE

Your program can call the TERMINATE intrinsic to delete itself. The following is an example of a TERMINATE intrinsic call:

```
TERMINATE;
```

MPE/iX deletes the process by following the algorithm described in "Deleting Processes".

Deleting a Child Process with KILL (PH Capability Required)

If your program has PH capability, it can call the KILL intrinsic to delete a child process. An example of a KILL intrinsic call is:

```
KILL (PIN);
```

PIN specifies the Process Identification Number of the child process to be deleted from the system. MPE/iX deletes the process by following the algorithm described in "Deleting Processes".

Aborting a Process

When your program has reached what you (the programmer) have defined as an error condition, you may wish to delete it from the system. In addition, you want to be notified that MPE/iX deleted the calling process in an error condition (aborted).

The QUIT intrinsic accomplishes this task by:

1. Transmitting an abort message to $STDLIST
2. Setting your job/session Job Control Word (JCW) to an error state
3. Directing MPE/iX to delete the calling process, following the algorithm described in "Deleting Processes"

The following is an example of a QUIT intrinsic call:

```
QUIT (NUM);
```

The NUM parameter is transmitted to $STDLIST as part of the abort message. You can use this number to assist you in determining the state of the calling process when the abort occurred.

In a session, your Root CI remains active even after the entire program finishes. In a batch job, the job terminates as soon as the entire program finishes, unless the :CONTINUE command is in effect.

Aborting the Entire User Process Structure

Your program can call the QUITPROG intrinsic to abort your entire user process structure (all processes below the Root CI). The QUITPROG intrinsic functions similarly to the QUIT intrinsic. The only difference is that when your program calls QUITPROG, MPE/iX deletes the entire user process structure (not just the calling process, as is the case with QUIT).

The following is an example of a QUITPROG intrinsic call:

```
QUITPROG (NUM);
```

The NUM parameter is transmitted as part of the abort message to the calling process's $STDLIST. You can use this number to assist you in determining the state of the calling process when the abort occurred.

In a session, your Root CI remains active even after the entire program is aborted. In a batch job, the job terminates, unless the :CONTINUE command is in effect.

Interrogating Processes

MPE/iX provides intrinsics that enable your program to interrogate MPE/iX about the status of other processes. The process information you acquire can assist you in controlling your process structure. Process interrogation intrinsics are:

- FATHER
- GETPROCID
- GETORIGIN
- GETPROCINFO
- PROCTIME
- PROCINFO

Determining the Parent Process (PH Capability Required)

If your program has PH capability, it can call the FATHER intrinsic to determine the Process Identification Number (PIN) of its parent process. Following is an example of a FATHER intrinsic call:

```
PARENTPIN := FATHER;
```

FATHER acts as a function to return to PARENTPIN the PIN of the parent process.

Determining a Child Process (PH Capability Required)

If your program has PH capability, it can call the GETPROCID intrinsic to determine the PIN of a child process. The following is an example of a GETPROCID intrinsic call:

```
CHILDPIN := GETPROCID (NUMSON);
```

GETPROCID acts as a function to return the PIN of the child process specified by NUMSON. If NUMSON specifies a value greater than the number of current child processes, GETPROCID returns the PIN of the current child process that was created first.

- If NUMSON = 0, GETPROCID returns the PIN of the first child created by the calling process.
- If NUMSON = n ($n > 0$), GETPROCID returns the PIN of the *nth* child created by the calling process. For example, if NUMSON = 3, GETPROCID returns to CHILDPIN the PIN of the third child process created by the calling process.

Determining Source of Activation (PH Capability Required)

If your program has PH capability, it can call the GETORIGIN intrinsic to determine who activated it, the parent process or a child process. An example of a GETORIGIN intrinsic call is:

```
ACTIVATIONSOURCE := GETORIGIN;
```

GETORIGIN returns one of the following codes to ACTIVATIONSOURCE:

0=	Neither the parent process nor a child process activated the calling process with an ACTIVATE intrinsic call
1=	The parent process activated the calling process
2=	A child process activated the calling process

Determining Process Priority and State (PH Capability Required)

If your program has PH capability, it can call the GETPROCINFO intrinsic to determine process management information about its parent process or a child process. GETPROCINFO acts as a function to return a value indicating:

- Current state of the specified process (active or suspended).
- Processes permitted to activate the specified process.
- The type of process (parent or child) that called ACTIVATE to activate the specified process.
- The priority class of the specified process.
- The priority number in the master queue of the specified process.

This is an example of a GETPROCINFO intrinsic call:

```
STATINFO := GETPROCINFO (PIN);
```

When PIN specifies 0, GETPROCINFO returns to STATINFO process management information about the parent of the calling process. For example, if the value $F0482 is returned to STATINFO, it is interpreted in the following manner:

Bits	Setting	Meaning
(31:1)	0	Process is currently suspended.
(29:2)	01	Only the parent process can activate the process.
(25:4)	0000	Not used.
(23:2)	01	The parent process last activated the process.
(20:3)	010	Process is in CS priority class.
(16:4)	0000	Not used.
(8:8)	00001111	Process has priority 15 in master queue.
(0:8)	00000000	Not used.

Determining Process Information

Your program can call the PROCINFO intrinsic to determine process management information. PROCINFO returns the same information as the GETPROCINFO intrinsic, as well as:

- The PIN of the calling process
- A value indicating the success or failure of the PROCINFO call, and, if unsuccessful, the nature of the error

Some of the information available through PROCINFO requires that your program have PH capability. For a list of capability requirements associated with each type of information available through PROCINFO, refer to the PROCINFO description in the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Determining Accumulated CPU Time for a Process

Your program can call the PROCTIME intrinsic to determine the total amount of CPU time it has used since it was created. The following is an example of a PROCTIME intrinsic call:

```
TIME := PROCTIME;
```

PROCTIME acts as a function to return to TIME the number of milliseconds your program has been in an EXECUTING state.

Passing Information to a Process

When your program creates a process, it can programmatically pass information to the child process with:

- The *param* parameter of the CREATE intrinsic
- Item numbers 2 and/or 11 of the CREATEPROCESS intrinsic
- The *parm* and/or *info* parameters of the :RUN command

The GETINFO intrinsic enables your program to retrieve this information.

This is an example of a GETINFO intrinsic call:

```
GETINFO (INFO,MAXLENGTH,PARAMETER);
```

The parameters specified in the example are described below.

INFO	Returns the contents of the *info* parameter from the :RUN command or Item Number 11 from CREATEPROCESS.
MAXLENGTH	Passes the maximum number of characters that MPE/iX is allowed to move to INFO. MAXLENGTH returns the number of characters actually moved to INFO by MPE/iX. MPE/iX will not move a number of characters greater than the original value of MAXLENGTH.
PARAMETER	Returns the contents of the *parm* parameter from the :RUN command, item number 2 from CREATEPROCESS, or the *param* parameter from CREATE.

Rescheduling a Process (PH Capability Required)

If your program has PH capability, it can call the GETPRIORITY intrinsic to change its priority class or the priority class of a child process. A process is scheduled on the basis of a particular priority class when it is created.

Generally, MPE/iX schedules processes in linear or circular subqueues:

- AS subqueue, a linear subqueue containing system processes only
- BS subqueue, a linear subqueue containing processes of very high priority
- CS subqueue, a circular subqueue recommended for interactive processes
- DS subqueue, a circular subqueue available for general use at a lower priority than the CS subqueue and recommended for batch jobs
- ES subqueue, a circular subqueue operating at a very low priority (background)

The following is an example of a GETPRIORITY intrinsic call:

```
GETPRIORITY (PIN,PRIORITYCLASS);
```

The parameters specified in the above example are described below.

PIN	Passes the Process Identification Number of the process whose priority class is to change. For example, a value of zero indicates the calling process.
PRIORITY-CLASS	Passes a value indicating the priority class in which the specified process is rescheduled. For example, the value 17,235 indicates that the process is rescheduled in the CS priority class.

Providing Communication Between Processes

Different processes can pass information among themselves using a special feature of the operating system, referred to as Interprocess Communication (IPC). Large tasks that have been broken into independent processes can use IPC to synchronize their actions and exchange data with other processes. There are several ways you can implement IPC on MPE/iX:

- Using file system intrinsics and message files
- Using session-level variables and Job Control Words (JCWs)
- Using process management mail intrinsics

The file system intrinsics HPFOPEN,FOPEN,FREAD,and FWRITE provide the most powerful method of performing IPC. These intrinsics can be used to communicate between any user process; the processes do not need to be in the same process tree, or running in the same job or session.

Processes executing in the same job or session can use a session-level variable or JCW to pass smaller amounts of data more efficiently than using message files.

Some older applications use the "mail" facility to communicate between processes in the same process tree (same job or session). Each process in the process tree can use this facility to pass information between itself and either its parent or child process.

For more information about using IPC features to provide communication between processes, refer to *Interprocess Communication Programmer's Guide* (32650-90019).

3

Deadlock Considerations

Simultaneous use of mail transmission, process suspension, and RIN-locking intrinsics throughout a process structure could result in a deadlock if the intrinsic calls are not synchronized properly. MPE/iX ensures that deadlock between different jobs/sessions cannot occur as long as none of the processes have Multiple RIN (MR) capability. Be aware of the following:

- In a multiprocess job/session, whenever a process is suspended (through the SUSPEND intrinsic, or when locking a RIN or receiving mail), MPE/iX does not determine whether or not all other processes in the tree are suspended. Avoid this situation.
- An attempt by a process to lock a global RIN succeeds only if both of the following conditions are met:
  - No other process within the job/session currently has locked this RIN. A global RIN cannot be used as a local RIN, because deadlock within the same job/session can occur.
  - The calling process currently has no other global RIN locked for itself. This could otherwise result in deadlock between two jobs/sessions.

For more information concerning the use of RIN-locking intrinsics refer to *Resource Management Programmer's Guide* (32650-90024).

4

Managing a Session Programmatically

This chapter describes how your program can exercise a limited form of control over interactive sessions on an MPE/iX-based computer system. Because an interactive session (or a job) is a process created and managed by the system, a user is restricted from being able to use process management intrinsics to create, manage, and delete sessions.

MPE/iX provides you with two intrinsics that enable you to direct the system to create a session on a specified terminal, then abort the session when desired:

- "Creating a Session Programmatically", describes how your program can use the `STARTSESS` intrinsic to create a session for yourself, or for other users, on an available (not currently assigned to a session) terminal without having to be at the specified terminal.
- "Aborting a Session Programmatically", describes how your program can use the ABORTSESS intrinsic to remove (abort) an interactive session from your system.

Creating a Session Programmatically

Normally, if you wish to create a session for yourself, you must be at a terminal and log on to the system by:

1. Entering the `:HELLO` command correctly.
2. Pressing RETURN
3. Waiting for the logon banner and prompt to appear on your terminal screen before continuing.

With Programmatic Sessions (PS) Capability, you can create a session for yourself, or for other users, on an available (not currently assigned to a session) terminal without having to be at the specified terminal(s). The operating system facilities for performing the PS task are available to you through the `STARTSESS` intrinsic and the `:STARTSESS` command.

Assigning the Programmatic Sessions Capability

Your System Manager (SM) or Account Manager (AM) must assign the PS capability to any user who plans to:

- Run a program that calls the `STARTSESS` intrinsic.
- Execute the `:STARTSESS` command from the MPE/iX CI, from a UDC, from the Startup State Configurator, or programmatically through the `MYCOMMAND` or `HPCICOMMAND` intrinsic.

When the operating system executes the code of either `STARTSESS` or `:STARTSESS`, it examines the user's capability list for the PS capability:

- If the PS capability is not found when invoking `STARTSESS`, the intrinsic fails and the program is aborted.
- If the PS capability is not found when invoking `:STARTSESS`, the command fails and control returns to the CI.

STARTSESS Description

The syntax for the `STARTSESS` intrinsic is:

`STARTSESS (`*ldev,logonstr,jsid,jsnum,errorstat*`);`

The syntax for the `:STARTSESS` command is:

`:STARTSESS` *ldev,logonstr*

The *ldev* parameter of `STARTSESS` and the *ldev* parameter of `:STARTSESS` are functionally equivalent, as are logonstr and logonstr, respectively. The intrinsic has additional parameters that return the job/session number (jsid, jsnum) and possible error conditions (errorstat).

Specifying a Terminal

You must specify, with the *ldev* parameter, the logical device number of the terminal on which the session is to be created (the target terminal). The target terminal must be:

- A real physical device
- An available terminal
- Job accepting
- Hard wired
- Set to the configured baud rate

Specifying the Logon

You must specify with the *logonstr* parameter a character array containing the logon. The logon is a superset of the `:HELLO` command syntax and includes an additional optional parameter `;NOWAIT`. The `:HELLO` command options are described in the *MPE/iX Commands Reference Manual* (32650-90003). The `;NOWAIT` parameter is described below.

If you specify the `;NOWAIT` keyword parameter in *logonstr* and the session is successfully created, MPE/iX continues with session startup as if it had actually received a carriage return from the target terminal. There are two additional requirements for the `;NOWAIT` keyword parameter:

- The target terminal must be turned on and set to the configured baud rate. Otherwise, `STARTSESS/:STARTSESS` fails and the session is not created.
- You must have SM capability if the target terminal is the System Console.

You must supply all required password(s) in the *logonstr* parameter. If required passwords are not supplied in logonstr, `STARTSESS/:STARTSESS` fails because MPE/iX does not prompt for passwords on the target terminal. You must place a carriage return character (%15) in the array element following the last valid character of *logonstr* to indicate the end of valid data.

Session Creation

If STARTSESS/:STARTSESS is successful, a session is created at the terminal specified in the *ldev*. MPE/iX returns a successful status to STARTSESS/:STARTSESS, and the session or process that called STARTSESS/:STARTSESS can continue execution without having to wait for a carriage return from the target terminal. If you execute a :SHOWJOB command from another terminal prior to (RETURN) being pressed on the target terminal, MPE/iX displays the newly created session in EXEC* state.

If ;NOWAIT is not specified in *logonstr*, MPE/iX waits for a carriage return from the target terminal before continuing session startup and printing I/O to the terminal. If ;NOWAIT is specified, MPE/iX continues with session startup as if it had actually received a carriage return from the target terminal.

After MPE/iX recognizes that you pressed (RETURN) on the target terminal:

1. The logon banner appears.
2. The line "***PROGRAMMATIC LOGON***" appears.
3. MPE/iX finishes the normal logon sequence (initializing or executing logon UDCs).

At this point, MPE/iX no longer distinguishes between a session started with STARTSESS/:STARTSESS and a session started with the :HELLO command.

Note If the target terminal is not set to the configured baud rate, MPE/iX cannot recognize a carriage return (or any input) from that terminal because there is no baud rate sensing by the intrinsic or the command. If this situation arises, you must change the baud rate setting on the target terminal to the configured rate and again press (RETURN).

If you put a :STARTSESS command into a Startup State Configurator file, you can also start a session on the System Console. MPE/iX does not log OPERATOR.SYS on to the Console, but instead treats the Console as an available terminal requiring a session logon.

Using STARTSESS in a Program

Example 4-1 is a simple Pascal XL program that demonstrates the use of the STARTSESS intrinsic. The program can facilitate the logon process for novice users by creating the desired session on the specified logical device. Because the logonstr parameter specifies ;NOWAIT, MPE/iX accomplishes the logon sequence and executes all logon UDCs. In this case, the novice user does not need any knowledge of the :HELLO command.

```
program Create_Session (input, output);

{This program demonstrates the use of the MPE/iX STARTSESS intrinsic to}
{programmatically create sessions on specified terminals.  This program}
{prompts the user for the ldev, then creates a session for NOVICE.ACCT }
{specifying the :NOWAIT option.  When the user inputs a zero, the      }
{program ends.                                                         }

const
   logon_string_size = 20;
   blank = ' ';

var
   ldev       :shortint;
   logonstring:packed array [1..logon_string_size] of char;
   jsid       :shortint;
   jsnum      :integer
   error      :shortint;

procedure STARTSESS; intrinsic;

begin
   logonstring := 'NOVICE.ACCT;NOWAIT   ';

   logonstring[17] := ch(13);            {Required delimiter.          }

   write ('Please input ldevs where you wish sessions started');
   writeln (' (0 = end of list)');
   readlin (ldev);
   while ldev <> 0 do
   begin                                 {Create session on ldev.   }
       error := 0;
       STARTSESS (ldev,logonstring,jsid,jsnum,error);
       if error <> 0 then
       begin                             {Return error status.      }
           write ('STARTSESS error = ',error);
           writeln (' on ldev = ' ldev);
       end;
       readln (ldev);
    end;                                 {End while loop.           }
end.
```

Example 4-1. Program Using STARTSESS Intrinsic to Create Sessions

If the following UDC is located in the UDC file of the specified session(s), the novice user is further relieved of having to deal with the MPE/iX CI:

```
LOGON
OPTION LOGON,NOBREAK,NOHELP,NOLIST
```

```
RUN HPMENU.PUB.SYS
BYE
***
```

This UDC is executed at each session logon when the `STARTSESS` program is run. It automatically places the user into the application specified in the `:RUN` command (in this case, `HPMENU.PUB.SYS`).

Using :STARTSESS in the Startup State Configurator

Example 4-1 illustrates how you can programmatically create sessions on specified terminals. However, the system must be up and running, and you must execute the program from a session you have already logged onto.

You can use `:STARTSESS` from within the System Startup State Configurator to create sessions on specified terminals every time you start the system. If you have SM Capability as well as PS capability, you can use the System Startup State Configurator facility to automatically:

- Reset any of the system-defined defaults
- Stream special jobs
- Open DS lines
- Create user sessions on specified terminals when the system comes up

Using the full functionality of the System Startup State Configurator, it is possible for you to describe beforehand exactly what the desired System Startup State should be.

In Example 4-2, the `STARTUP` command defines actions that the system will perform regardless of the type of system startup used by the operator. WARMSTART (corresponds to the system startup command `START RECOVERY`) and COOLSTART (corresponds to the system startup command `START NORECOVERY`), as defined below, will cause sessions to be created for `MGR.ACCT` on logical device 20 (the Console) and for `NOVICE.ACCT` on logical devices 21, 22, and 23.

```
STARTUP
STREAMS 10
ALLOW @.@;COMMANDS=REPLY
ALLOCATE COBOLII.PUB.SYS
LIMIT 4,16
JOBFENCE 4
OUTFENCE 5
***
WARMSTART
STARTSESS 20;MGR.ACCT;HIPRI;NOWAIT
STARTSESS 21;NOVICE.ACCT;NOWAIT
STARTSESS 22;NOVICE.ACCT.NOWAIT
***
COOLSTART
STARTSESS 20;MGR.ACCT;HIPRI;NOWAIT
STARTSESS 21;NOVICE.ACCT;NOWAIT
STARTSESS 22;NOVICE.ACCT;NOWAIT
STARTSESS 23;NOVICE.ACCT;NOWAIT
***
```

Example 4-2. :STARTSESS Command Used in Startup State Configurator

If the following UDC is located in the UDC file of the specified sessions, the user is relieved of having to deal with the MPE/iX CI:

```
LOGON
OPTION LOGON,NOBREAK,NOHELP,NOLIST
RUN HPMENU.PUB.SYS
BYE
***
```

This UDC is executed at the time of session logon. It automatically places the user into the application specified in the `:RUN` command (in this case, `HPMENU.PUB.SYS`).

For more information about the MPE/iX System Startup State Configurator, refer to *Managing Jobs and Sessions* (32650-90035).

Aborting a Session Programmatically

Once your program has successfully created a session using the STARTSESS intrinsic, your program no longer has any control over that session, except to abort it using the ABORTSESS intrinsic. The newly created sessions runs as an independent system-managed process and is not a child process of the process that invoked STARTSESS.

The ABORTSESS intrinsic (the programmatic equivalent to the :ABORTJOB command) is provided to enable your program to remove (abort) a selected session from your system. ABORTSESS is commonly used in conjunction with the STARTSESS intrinsic to allow your program a limited form of control over interactive sessions located on your system.

Requirements for Using ABORTSESS

While the ABORTSESS intrinsic does not require either the PS or the PH capabilities, your program can successfully call ABORTSESS only if the program is executing in a session (or job) that satisfies one or more of the following conditions:

- Your session is on the System Console, or has been allowed the :ABORTJOB command by the Console.
- :JOBSECURITY has been set to LOW and one of the following three conditions is true:
  1. The user and account names of your session are the same as the user and account names of the session to be aborted.
  2. Your session has AM capability, and the account name of your session is the same as the account name of the session to be aborted.
  3. Your session has SM capability.

Note Aborting a session or job is a last resort measure. If you must do so, make sure you abort the right one, and make every effort to warn the user first.

Identifying the Job/Session You Want Aborted

The ABORTSESS takes as a required parameter the job/session number the system has associated with the session you want to abort. The STARTSESS intrinsic returns this job/session number to your program when you create the session.

The JOBINFO intrinsic can also return a job/session number if you pass it the user and account names of a selected session, but if there are multiple sessions associated with the same user and account, you cannot be guaranteed that JOBINFO will return the correct job/session number. In this case, the value returned by STARTSESS is the safest value to use.

An example of an ABORTSESS call is:

```
ABORTSESS(JSID,JSNUM,JSSTATUS)
```

The parameters specified in the example are described below:

JSID — Passes an integer value indicating whether the process to be aborted is a session (JSID=1) or a job (JSID=2). This parameter is necessary because session numbers and job numbers can be identical.

JSNUM Passes the system-defined session number or job number of the process that is to be aborted. This value was returned by the STARTSESS intrinsic when you created the session you are now planning to abort.

JSSTATUS Returns status information about the success or failure of the intrinsic call.

ABORTSESS deletes the specified session or job from the system and sends a message to the standard list device ($STDLIST) of the aborted session or job:

```
SESSION ABORTED BY SYSTEM MANAGEMENT
```

or

```
JOB ABORTED BY SYSTEM MANAGEMENT
```

No abort message is sent to the System Console. The session or job is deleted from the system in a fashion similar to that described in "Deleting Processes" in Chapter 2 "Process Management Tasks".

For more information about managing jobs and sessions on MPE/iX, refer to *Managing Jobs and Sessions* (32650-90035).