THE PROCESS MANAGER

December 7, 1984

2600 Liberty Avenue
P. O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

Accent is a trademark of Carnegie-Mellon University.

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

The information in this document is subject to change without notice and should not be construed as a commitment by PERQ Systems Corporation. The company assumes no responsibility for any errors that may appear in this document.

PERQ Systems Corporation will make every effort to keep customers apprised of all documentation changes as quickly as possible. The Reader's Comments card is distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

PERQ, PERQ2, LINQ, and Qnix are trademarks of PERQ Systems Corporation.

Table of Contents Page

1. Theory

Accent supports the concurrent execution of multiple processes. The Accent Process Manager provides keyboard process control and inheritance for the user processes within the system. It supplements the process control functions available from the Accent Kernel, which are described in the document "Theory of Operations" in this manual.

A process must be registered with the Process Manager to make use of its operations. It is possible to run a process without registering it, but an unregistered process is harder to start up and impossible to control.

Registering the process supplies the Process Manager with:

- the kernel and data ports of the new process;
- a descriptive name for the process;
- the window, typescript, and environment manager connection;
- the parent of the new process.

Chapter 2 describes how the Accent Process Manager works, and Chapter 3 lists the definitions, routines, and asynchronous messages to a process.

2. Use

2.1. Process Trees

Processes are organized into trees. Each process in the tree has a parent and a set of children. The parent of a process is notified when one of its children dies, and the parent may control its children via Accent or Process Manager functions.

When a process is created, it is registered with the Process Manager. The process that created it (via Spawn) is registered as its parent.

A process is notified (via an emergency message) when one of its child processes terminates. The parent receives the reason for the termination (if the child gave one), and the run, elapsed, and load times for the process (if available).

A process may ask the Process Manager to terminate it, supplying a reason for termination. The reason is given to the parent process. If the process dies without asking the process manager, the reason is given as "process death." (In this case, some of the statistics are not available for the process.) Its parent is immediately notified.

A process may re-register itself with a new parent (or no parent). If this happens, its former parent is notified that the process has "died," with a reason code stating that the child has disowned its parent.

When a process dies, its surviving children are "orphans" (not children of any other process).

2.2. Process IDs

A process is usually identified by its KernelPort. However, the Process Manager also assigns each process a unique process ID. This process ID is used by some Process Manager routines (listed in Section 2.5) to identify a process. These routines accept the string form of the process ID. The process ID can be obtained by using the PMGetWaitID or PMGetStatus routines (described in Section 3.2).

Another use for the process ID is to identify a process after termination. After a process has terminated, its KernelPort has been deleted. For this reason, the process ID is used to identify the process in the ProcessDeath message sent to its parent.

2.3. Process (Control) Groups

Processes are organized into Process Groups, each one associated with a controlling window (or more than one window). A process control key typed to the controlling window affects all of the processes within its process group.

More than one window may be a control window for the same process group. In this case, a process control key has the same effect no matter which of the controlling windows it is typed to.

Each process group has a name. It is the name displayed in the icon for the window controlling the process group. If there is more than one control window, the name in the icon for any of those windows is an alias for the name of the process group.

A process may change its process group at any time. If it does, it becomes the head of a new process group and all of its children become members of the new group. It retains its parent, who will still be notified when the process dies (but not when any of the control signals are used). The process may also rejoin its parent's

process group.

The control window for a new process group must not already be a control window for another process group.

A process whose last control window is removed is "out of control" (there is no default control window for it). It can only be controlled by the shell process control commands.

2.4. Process Control Signals

The process manager provides a large set (64) of Process Control Signals. Of these, seven have defined roles:

Suspend
Resume
Status
Level 1 Abort
Level 2 Abort
Level 3 Abort
Debug

A process may set up one of three actions to take on a process control signal:

Send	Send the process an emergency message specifying which signal has occurred. If the process is suspended, resume it first.
Ignore	Ignore the signal. A Level 3 Abort cannot be ignored; the process is terminated anyway.
Default	Perform the process manager's default action for the signal.

The default actions are:

Suspend	Suspend the process
Resume	Resume the process
Status	Display process status in the process manager window
Level 1 Abort	Terminate the process

Level 2 Abort
: Terminate the process

Level 3 Abort
: Terminate the process

Debug
: Suspend the process and invoke a debugger on it

Others
: Ignore signal

The action for each signal (send, ignore, or default) may be specified independently.

There are process manager routines to send a signal to an entire process group or to a single process in a process group. These are emergency messages to the process manager.

The process that wants to handle signals can set a port (using PMSetSignalPort) to receive the signal emergency messages. The default for this port is the process's DataPort.

2.5. Process Control by Process Group Name or Process ID

The process manager provides routines to control a process group by name. These are invoked by corresponding shell commands. The available routines are:

PMSuspend
PMResume
PMDebug
PMKill
PMSetPriority
PMSignalByName

These take a string representing the process name or process ID. If the string is the name of a process group, the command affects all of the processes in that group. If the string is the ID of a process, or is a prefix of only one process name, the command affects only that process. (If the string matches more than one process name, the command fails and returns NameAmbiguous).

2.6. Statistics

For each process, the process manager keeps track of:

Load Time	microseconds
Elapsed (clock) time	(1 / 60)second
Run (CPU) time	microseconds

Note that Load Time for a process is not currently maintained by the Accent operating system, and is usually zero. However, Load Time may be used for different purposes by other subsystems.

The Process Manager can return the status for one or several processes. It does a pattern match on a supplied name and returns an array of status records for each process whose name matches. The status information includes the times for the process and several items available from the Kernel:

Process ID number

State: supervisor, privileged user, normal user

Priority

Process queue number: ready, pending, sleeping

RunName: process name given at registration

IconName: name in icon for window

2.7. Debugging Aids

A process can get the registered ports of another process. These ports are:

Window

Typescript

EnvironmentConnection

Spawn provides these ports explicitly; others can also be passed. These ports are needed to establish the running process's total environment.

The debugger may use these ports to re-establish the environment of a debugged process on a subsequent run, without having to set up the window a second time.

The process manager can register a debug port for a process. If the process gets an uncaught exception or an addressing error, or if a signal is raised and not caught, the process is suspended and an emergency message is sent to the debug port. (This is like AccInt.SetDebugPort in the kernel, but intercepts signals as well as program errors.) The port may be set up to catch ALL uncaught signals or only the Debug signal. A debugger can use this to keep control of a process and to intercept subsequent Debug keys (instead of starting up another debugger).

A process can force another process into the Debugger (used to load a program and invoke the debugger before the program has run).

2.8. Invoking a Debugger

The Process Manager knows how to start up a debugger for another process.

The global environment variable "DebuggerName" is the name of a debugger to run. To invoke a debugger, the Process Manager first tries to run the program named in "DebuggerName." If that is not found (or the variable is blank or undefined), it tries to run "Debugger.Run." If that, in turn, is missing, it runs the built-in Mace debugger.

If "DebuggerName" has the value "?", the Process Manager asks the user for the name of a run file to run as the debugger. It repeats the prompt until the file is found or until the user asks to run the built-in debugger.

The debugger starts up with the following environment:

InPorts^[0]
: Kernel Port of target process

InPorts^[1]
: Process Manager Port

EMPort
: Environment Manager connection for target process

UserWindow, UserTypescript
: Window and typescript for the debugger. Process control functions on this window affect the Debugger, not the target process (the debugger may, of course, intercept them)

2.9. Summary of Process Control Routines

These routines are defined in Section 3.2.

Registering and controlling signals for a process (usually done by a process or by its parent in loading it):

PMRegisterProcess
PMSetSignalPort
PMSetSignal
PMTerminate
PMSaveLoadTime

Actions usually taken to control or wait for another process:

PMSetDebugPort
PMGetProcPorts
PMGetTimes
PMGetWaitID
PMDebugProcess

Control windows for process group:

PMAddControlWindow
PMRemoveControlWindow
PMChangeGroup

Control by String Names:

PMSuspend
PMResume
PMDebug
PMKill
PMSetPriority
PMSignalByName

Control functions (on Window or Process):

PMGroupSignal

PMProcessSignal

Miscellaneous:

PMGetStatus
PMBroadcast

2.10. Functions Provided by User Code

Spawn forks a process or loads a program into a new process. It automatically registers the new process as a child of the old one. The process group for the new process is the one associated with the Window Manager Connection (Window) that is passed to the new process. If it is a new window, a new process group is created.

2.11. Emergency Messages

A process may receive three emergency messages from the process manager (the emergency messages are defined in Section 3.3):

1. Signal sent to this process
2. Action taken on a process for whom this is DebugPort
3. Process Termination

3. Definitions

The definitions throughout this document are given in Pascal. If you are programming in the C language, please refer also to the document "C System Interfaces" in the *Accent Languages Manual*. If you are programming in the Lisp language, see the document "Lisp Interaction with the Accent Operating System" in the *Accent Lisp Manual*. When FORTRAN becomes available under Accent, the definitions will be the same as in the C language.

3.1. Type Definitions

The type and constant definitions are found in module ProcMgrDefs in ProcMgrDefs.Pas in LibPascal.

```
Const
        ProcMgrBase     = 3600; { Process Manager        }
                                {    messages and errors }
        SignalBase      = 3800; { Signals and asynchronous}
                                {    returns             }

        { Error returns from ProcMan }

        UnknownProcess  = ProcMgrBase+1;
        UnknownSignal   = ProcMgrBase+2;
        UnknownAction   = ProcMgrBase+3;
        UnknownWindow   = ProcMgrBase+4;
        WindowInUse     = ProcMgrBase+5;
        NoChildren      = ProcMgrBase+6;
        UnknownPort     = ProcMgrBase+7; { NullPort given }
                                         {    to ProcMgr }
        NameAmbiguous   = ProcMgrBase+8;
        ProcessDisowned = ProcMgrBase+9;

        { Signals }

        MinSignal       = SignalBase;
        MaxSignal       = SignalBase+63;

        { signal numbers 1..32 are reserved for Qnix }

        SigSuspend      = MinSignal+33;
        SigResume       = MinSignal+34;
```

```
        SigStatus       = MinSignal+35;
        SigDebug        = MinSignal+36;
        SigLevel1Abort  = MinSignal+37;
        SigLevel2Abort  = MinSignal+38;
        SigLevel3Abort  = MinSignal+39;


type
        SignalName = integer;


        { Actions }
type
        SignalAction = (SigDefault,
                        SigIgnore,
                        SigSend);


        { Status returned to caller }

        StatRecord = record
                RunTime:        long; { microseconds }
                LoadTime:       long; { microseconds }
                ElapsedTime:    long; { ticks }
                KernelPort:     long; { process ID number,
                                      { called KernelPort
                                      { for historic reasons }
                Priority:       integer;
                QueueID:        integer;
                ProcName:       string; { Process name }
                IconName:       ProgStr;{ name in Icon
                                          - group name }
                State:          ProcState;
             end;

        StatArray = array[0..0] of StatRecord;
        StatList  = ^StatArray;
```

3.2. Routine Definitions

3.2.1. Returning version number

```
function ProcMgr_Version( ServPort : Port ): String;
```

Abstract:

Gets version number of the Process Manager.

Parameters:

ServPort Process Manager Service Port (PMPort)

Returns:

Version number of Process Manager: "Process Manager Vx.x of dd-mmm-yy"

3.2.2. Registering a process

```
function PMRegisterProcess(
                    ServPort       : Port;
                    HisKPort       : Port;
                    HisDPort       : Port;
                    ProgName       : string;
                    HisWindow      : Window;
                    HisTypescript  : Typescript;
                    EMConn         : port;
                    Parent         : port
        ): GeneralReturn;
```

Abstract:

Registers a process for process control, debugging, and inheritance.

If the process is not already registered, it is registered and added to its parent's set of children. HisWindow is made the control window for the process. If it is already registered, it becomes the child of the new Parent (and disowns its old parent). All of its children go with it.

PMRegisterProcess sets the load time for the process to zero.

Parameters:

ServPort Process Manager Service Port (PMPort)

HisKPort Process KernelPort

HisDPort Process DataPort

ProgName Name to register process under for control commands from Shell

HisWindow
Process Window

HisTypescript
Process Typescript

EMConn Process Environment Manager connection

Parent Kernel port for process parent

Returns:

Success

UnknownProcess
Parent process not registered

WindowInUse
Window already controls a process group

(For the above three, the process is registered.)

UnknownPort
HisKPort = NullPort

3.2.3. Setting port on which to receive signal messages

```
function PMSetSignalPort(
                ServPort   : Port;
                ProcPort   : Port;
                SignalPort : Port
        ): GeneralReturn;
```

Abstract:

Set the signal port (port to receive signal message for SigSend) for a process.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel Port of process to affect

SignalPort Port to receive SignalMsg

Returns:

Success

UnknownProcess

UnknownPort

3.2.4. Setting signal action for a process

```
function PMSetSignal(
                ServPort : Port;
                ProcPort : Port;
                Signal   : SignalName;
                Action   : SignalAction
        ): GeneralReturn;
```

Abstract:

Sets the signal actions for a process.

Parameters:

ServPort	Process Manager Service Port (PMPort)
ProcPort	Kernel Port of process to affect
Signal	Signal to change action for
Action	New action to set for signal. Actions are:
	SigSend — send a SignalMessage to the process's SignalPort, with the signal as the reason
	SigIgnore — completely ignore the signal
	SigDefault — take the default action for the signal

Returns:

Success

UnknownProcess

UnknownSignal

UnknownAction

3.2.5. Terminating a process

```
function PMTerminate(
                ServPort : Port;
                ProcPort : Port;
                Reason   : long
        ): GeneralReturn;
```

Abstract:

Terminate a process.

Parameters:

ServPort	Process Manager Service Port (PMPort)
ProcPort	Kernel port of process to terminate
Reason	Reason for termination

Returns:

Success

UnknownProcess

3.2.6. Saving load time for a process

```
Function PMSaveLoadTime(
                ServPort : Port;
                ProcPort : Port;
                LoadTime : long
        ): GeneralReturn;
```

Abstract:

Saves the load time for a process so it can be printed later.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel port of process

LoadTime Time to load process (microseconds)

Returns:

Success

UnknownProcess

3.2.7. Setting port for errors and default signals

```
function PMSetDebugPort(
                ServPort        : Port;
                ProcPort        : Port;
                DebugPort       : port;
                DebugSignalOnly : boolean
        ): GeneralReturn;
```

Abstract:

Sets up a port to intercept errors and default signals for a process. If the debug port exists, the default action for the debug signal (or all signals) will be to suspend the process and send a message to the debug port; in addition, the debug port will receive all DEBUG error messages for the process (from the kernel).

The "SetDebugPort" routine for the Kernel only intercepts exceptions and memory faults. This call intercepts Signals as well.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel Port of process to set the debug port for

DebugPort Port to receive emergency message when something happens to the process

DebugSignalOnly
One of:

True	only intercept uncaught Debug signals
False	intercept All uncaught signals

Returns:

Success

UnknownProcess

3.2.8. Getting registered ports of a process

```
function PMGetProcPorts(
                ServPort      : Port;
                ProcPort      : port;
            var hisWindow     : Window;
            var hisTypescript : Typescript;
            var hisEMConn     : Port
        ): GeneralReturn;
```

Abstract:

Gets the registered ports of a process.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel port for process

hisWindow
Returns window port for process

hisTypescript
Returns typescript port for process

hisEMConn

Returns Environment Manager connection for process

Returns:

Success

UnknownProcess

3.2.9. Returning run and elapsed times for a process

```
function PMGetTimes(
                ServPort    : Port;
                ProcPort    : Port;
            var LoadTime    : long;
            var RunTime     : long;
            var ElapsedTime : long
        ): GeneralReturn;
```

Abstract:

Returns the run and elapsed time for a process.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel port of process

LoadTime Returns the load time for the process (microseconds)

RunTime Returns the run time for the process (microseconds)

ElapsedTime
Returns the elapsed time for the process (1/60 second)

Returns:

Success

UnknownProcess

3.2.10. Getting process ID of a child process

```
function PMGetWaitID(
                ServPort : Port;
                ProcPort : Port;
            var WaitID   : long
        ): GeneralReturn;
```

Abstract:

Get the process ID ("Wait ID") of a child process. The process ID is a 32-bit number that is returned when the child process dies. It is used to identify the dead process (the dead process' KernelPort is no longer valid, since it was

deallocated when the process died). Note that the process ID has to be converted to a string by the user before it can be used as a parameter for another routine.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel port of child process

WaitID Returns the process ID for the child process

Returns:

Success

UnknownProcess

3.2.11. Invoking a debugger, with supplied reason

```
function PMDebugProcess(
                ServPort : Port;
                ProcPort : Port;
                Reason   : long
        ): GeneralReturn;
```

Abstract:

Invokes a debugger on a process.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel port of process to terminate

Reason Reason for debugging process

Returns:

Success

UnknownProcess

Failure Couldn't invoke debugger

3.2.12. Adding a control window for a process group

```
function PMAddCtlWindow(
                ServPort     : Port;
                CtlWindow    : Window;
                NewCtlWindow : Window
        ): GeneralReturn;
```

Abstract:

Adds a new window to the set of controlling windows for a process group.

Parameters:

ServPort Process Manager Service Port (PMPort)

CtlWindow Existing window for a control group

NewCtlWindow
New window to add as a control window

Returns:

Success

UnknownWindow
CtlWindow is not a control window for a process group

UnknownPort
NewCtlWindow is NullPort

3.2.13. Removing window from set of controlling windows

```
function PMRemoveCtlWindow(
                ServPort  : Port;
                CtlWindow : Window
        ): GeneralReturn;
```

Abstract:

Removes a window from the set of controlling windows for a process group.

Parameters:

ServPort Process Manager Service Port (PMPort)

CtlWindow
Window to remove

Returns:

Success

UnknownWindow
CtlWindow is not a control window for a process group

3.2.14. Changing process group of a process

```
function PMChangeGroup(
                ServPort   : Port;
                ProcPort   : Port;
                NewWindow  : Window
        ): GeneralReturn;
```

Abstract:

Changes a process (and its descendants) to a new process group. If there is already a process in the process group, it must be the parent of the affected process.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel port for process

NewWindow
Window to use for new group

Returns:

Success

UnknownProcess

UnknownPort
If window is NullPort

WindowInUse
Window already controls a process group.

3.2.15. Suspending a process

```
Function PMSuspend(
                ServPort : Port;
                ProcID   : string
        ): GeneralReturn;
```

Abstract:

Suspends a named (or numbered) process.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcID Name of group, or prefix of process name, or process ID as returned by PMGetStatus or PMGetWaitID (note that process ID must be converted to string for use here)

Returns:

Success

UnknownProcess

NameAmbiguous

Side Effects:

If the name designates a process group, it sends the Suspend signal to the process group.

If the name designates a single process, it suspends the process. If the process has a DebugPort set, it also sends an M_DebugMsg message to the debug port, with SigSuspend as the reason (unless the debug port is set to DebugSignalOnly).

3.2.16. Resuming a suspended process

```
Function PMResume(
                ServPort : Port;
                ProcID   : string
        ): GeneralReturn;
```

Abstract:

Resumes a named (or numbered) process.

Parameters:

ServPort — Process Manager Service Port (PMPort)

ProcID — Name of group, or prefix of process name, or process ID as returned by PMGetStatus or PMGetWaitID (note that process ID must be converted to string for use here)

Returns:

Success

UnknownProcess

NameAmbiguous

Side Effects:

If the name designates a process group, it sends the Resume signal to the process group.

If the name designates a single process, and the process does not have a DebugPort set, it resumes the process. If the process has a DebugPort set, it instead sends an M_DebugMsg message to the debug port, with SigResume as the reason (unless the debug port is set to DebugSignalOnly).

3.2.17. Invoking a debugger on a process

```
Function PMDebug(
                ServPort : Port;
                ProcID   : string
        ): GeneralReturn;
```

Abstract:

Invokes the debugger on a named (or numbered) process.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcID Name of group, or prefix of process name, or process ID as returned by PMGetStatus or PMGetWaitID (note that process ID must be converted to string for use here)

Returns:

Success

UnknownProcess

NameAmbiguous

Side Effects:

If the name designates a process group, it sends the Debug signal to the process group.

If the name designates a single process, and the process does not have a DebugPort set, it suspends the process and invokes a debugger on it. If the process has a DebugPort set, it instead sends an M_DebugMsg message to the debug port, with SigDebug as the reason.

3.2.18. Killing a process

```
Function PMKill(
                ServPort : Port;
                ProcID   : string
        ): GeneralReturn;
```

Abstract:

Kills a named (or numbered) process.

Parameters:

ServPort — Process Manager Service Port (PMPort)

ProcID — Name of group, or prefix of process name, or process ID as returned by PMGetStatus or PMGetWaitID (note that process ID must be converted to string for use here)

Returns:

Success

UnknownProcess

NameAmbiguous

Side Effects:

If the name designates a process group, it sends the SigLevel1Abort signal to the process group.

If the name designates a single process, and the process does not have a DebugPort set, it terminates the process with reason = SigLevel1Abort. If the process has a DebugPort set, it instead sends an M_DebugMsg message to the debug port, with SigLevel1Abort as the reason (unless the debug port is set to DebugSignalOnly).

3.2.19. Setting priority of a process

```
Function PMSetPriority(
                ServPort : Port;
                ProcID   : string;
                Priority : integer
        ): GeneralReturn;
```

Abstract:

Sets the priority of a named (or numbered) process, or of all of the processes in a process group.

Parameters:

ServPort — Process Manager Service Port (PMPort)

ProcID — Name of group, or prefix of process name, or process ID as returned by PMGetStatus or PMGetWaitID (note that process ID must be converted to string for use here)

Priority — Desired run priority for process:
0: lowest priority
15: highest priority

Returns:

Success

UnknownProcess

NameAmbiguous

BadPriority

Side Effects:

Changes the priority of the process or of all processes in the process group.

3.2.20. Sending signal to a process group

```
Procedure PMGroupSignal(
                ServPort   : Port;
                CtlWindow  : Window;
                Signal     : SignalName);
```

Abstract:

Sends a signal to a process group. This is an Emergency message to the Process Manager.

Parameters:

ServPort Process Manager Service Port (PMPort)

CtlWindow One of the windows controlling a process group

Signal Signal to send

3.2.21. Sending signal to a single process

```
Procedure PMProcessSignal(
                ServPort : Port;
                ProcPort : Port;
                Signal   : SignalName
        );
```

Abstract:

Sends a signal to a single process. This is an Emergency message to the Process Manager.

Parameters:

ServPort Process Manager Service Port (PMPort)

ProcPort Kernel port for process to signal

Signal Signal to send

3.2.22. Sending signal to process or group by string name

```
Function PMSignalByName(
                PMPort          : Port;
                ProcID          : String;
                ProcessOrGroup  : Boolean;
                Signal          : SignalName
        ): GeneralReturn;
```

Abstract:

Sends a signal to a process or group of processes designated by the string name of the process or group. The signal is sent to the process, or each process in the group. The action taken depends on what the process has set for that signal (Ignore, Send, or Default).

Parameters:

PMPort — Process Manager Service Port (PMPort)

ProcID — String name or process ID of process or group (note that if you are using process ID, it must be converted to string first)

ProcessOrGroup
One of:

True	control a single process
False	control an entire process group

Signal — Signal to send

ProcID is parsed as follows:

If ProcessOrGroup is True (signaling one process):

If ProcID is entirely numeric, it is interpreted as the process ID of a process;

Otherwise, ProcID must be a unique initial portion of a process name.

If ProcessOrGroup is False (signaling an entire group):

If ProcID is entirely numeric, it is interpreted as the process ID of a process. That process must be at the head of a process group;

Otherwise, ProcID must be the name of a control window for a process group.

Returns:

Success

UnknownProcess

NameAmbiguous

UnknownSignal

3.2.23. Getting status information

```
Function PMGetStatus(
                ServPort   : Port;
                ProcID     : String;
            var Stats      : StatList;
            var Stats_Cnt  : long
        ): GeneralReturn;
```

Abstract:

Returns status for one or more processes. Note that the process ID (the KernelPort field of Stats) must be converted to a string by the user before it can be used as a parameter in another routine.

Parameters:

ServPort — Process Manager Service Port (PMPort)

ProcID — One of:

Process group name
returns status for all processes in group

Pattern to match (as in file name patterns)
returns status for all matching process names

Process ID number for process
returns status for that process

Null string
returns status for all registered processes

Stats — Returns a pointer to the array of status records, one for each process whose name matches ProcID

Stats_Cnt
Returns number of records in Stats

Returns:

Success

UnknownProcess
ProcID didn't match any process

3.2.24. Printing message in the Process Manager window

```
Function PMBroadcast(
                ServPort : Port;
                S        : string
        ): GeneralReturn;
```

Abstract:

Prints a message in the Process Manager Window. This is used to display "system" messages.

Parameters:

ServPort Process Manager Service Port (PMPort)

S String to display

Returns:

Success

3.3. Asynchronous (Emergency) Messages to a Process

3.3.1. Report on action of another process

Abstract:

Message sent to DebugPort to report an action on another process (this is the same as the kernel's M_DebugMsg).

Parameters:

KPort Kernel port of process affected

Arg1 Always 0 (present for historical reasons)

Arg2 Reason for DebugMessage:

If the process was halted by an error, this is the GeneralReturn value describing the error (MemFault, UncaughtException, ...)

If the process has a Process Manager Debug Port set (by PMSetDebugPort), and a signal was raised on the process and not sent or ignored, this is the name of the Signal that was raised

```
type DebugMessage = record
      Head    : Msg;          { Accent message header }

      tKPort  : TypeType;     { (TypePT, 32) }
      KPort   : Port;         { Kernel port of process
                                    affected }

      tArg1   : TypeType;     { (TypeInt32, 32) }
      Arg1    : Long;         { What happened, field 1:
                                always 0! }
      tArg2   : TypeType;     { (TypeInt32, 32) }
      Arg2    : Long;         { What happened, field 2:
                                GeneralReturn value for
                                error or signal }
   end;
```

3.3.2. Process termination

Abstract:

Message sent on process termination.

Parameters:

WaitID Process ID of process that died, as returned by PMGetWaitID

Reason Reason for process death:

if killed by Terminate (in AccInt), PROCESSDEATH

if killed by PMTerminate(in ProcMgr), the Reason given to PMTerminate

if the process re-registered with a different parent, ProcessDisowned

if the process was killed by a SigLevel[1,2,3]Abort, the Signal

value

LoadTime CPU time to load process (microseconds)

RunTime CPU time to run process (microseconds)

ElapsedTime
Total time elapsed from PMRegisterProcess until termination (60Hz clock ticks).

```
const ProcessDeathMsgID = 3800;

type ProcessDeathMsg = record
      Head          : Msg;          { Accent Message Header }

      tWaitID       : TypeType;     { (TypeInt32, 32) }
      WaitID        : long;         { process ID of process,
                                      returned by
                                      PMGetWaitID }

      tReason       : TypeType;     { (TypeInt32, 32) }
      Reason        : long;         { reason for process death }

      tLoadTime     : TypeType;     { (TypeInt32, 32) }
      LoadTime      : long;         { process Load time
                                          (microseconds) }

      tRunTime      : TypeType;     { (TypeInt32, 32) }
      RunTime       : long;         { process Run time
                                          (microseconds) }

      tElapsedTime  : TypeType;     { (TypeInt32, 32) }
      ElapsedTime   : long;         { Elapsed time (ticks) }

   end;
```

3.3.3. Signal to a process

Abstract:

Message sent for Signal to a process: this is sent to a process's SignalPort, or to its DATAPORT if the SignalPort has not been set or is NullPort.

Parameters:

CtlWindow
: Window that the signal was sent from. If the signal was sent only to one process, it is the window that heads the control group for the process

Signal
: Signal sent to process

```
const SignalMsgID = 3801;

type SignalMsg = record
      Head        : Msg;          { Accent message header }

      tCtlWindow  : TypeType;     { (TypePt, 32) }
      CtlWindow   : Window;       { window that signal was
                                    received on }

      tSignal     : TypeType;     { (TypeInt16, 16) }
      Signal      : SignalName;   { signal received }
   end;
```