

**F\$Alarm****Set Alarm Clock**

**ASM CALL:** OS9 F\$Alarm

**INPUT:** d0.l = Alarm ID (or zero)  
 d1.w = Function code  
 d2.l = Reserved, must be zero  
 d3.l = Time interval (or time)  
 d4.l = Date (when using absolute time)  
 (a0) = Register image

**OUTPUT:** d0.l = Alarm ID

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** When called from system state, F\$Alarm causes the execution of a system-state subroutine at a specified time. It is provided for such functions as turning off a disk drive motor if the disk is not accessed for a period of time.

The register image pointed to by register (a0) contains an image of the registers to be passed to the alarm subroutine. The subroutine entry point must be placed in R\$pc(a0). The register image is copied by the F\$Alarm request into another buffer area and may re-used immediately for other purposes.

The alarm ID returned may be used to delete an alarm request.

The time interval is the number of system clock ticks (or 256ths of a second) to wait before the alarm subroutine is executed. If the high order bit is set, the low 31 bits are interpreted as 256ths of a second. **NOTE:** All times are rounded up to the nearest clock tick.

The system automatically deletes a process's pending alarms when the process dies.

The alarm function code is used to select one of the related alarm functions. Not all input parameters are always needed; each function is described in the following pages.

The following function codes are supported:

A\$Delete	Remove a pending alarm request
A\$Set	Execute a subroutine after a specified time interval
A\$Cycle	Execute a subroutine at specified time intervals
A\$AtDate	Execute a subroutine at a Gregorian date/time
A\$AtJul	Execute a subroutine at Julian date/time

System-state alarm subroutines must conform to the following conventions:

**INPUT:**    **d0-d7** = caller's registers (R\$d0-R\$d7(a5))  
              **(a0)-(a3)** = caller's registers (R\$a0-R\$a3(a5))  
              **(a4)** = system process descriptor pointer\*  
              **(a5)** = ptr to register image  
              **(a6)** = system global storage pointer

**OUTPUT:**    **cc** = carry set  
              **d1.w** = error code if error

\* **NOTE:** The user number in the system process descriptor will have been temporarily changed to the user number of original F\$Alarm request. The registers d0-d7 and (a0)-(a3) do not have to be preserved.

**CAVEATS:**    System-state alarms are executed by the system process at priority 65535. They may never perform any function that can result in any kind of queuing, such as F\$Sleep, F\$Wait, F\$Load, F\$Event (Ev\$Wait), F\$IOQu, or F\$Fork. When such functions are required, the caller must provide a separate process to perform the function, rather than an alarm.

**WARNING:**    If an alarm execution routine suffers any kind of bus trap, address trap, or other hardware-related error, the system will crash.

**SEE ALSO:**    F\$Alarm User-State System Call

**POSSIBLE**

**ERRORS:**    E\$UnkSvc, E\$Param, E\$MemFul, E\$NoRAM, and E\$BPAAddr.

*F\$Alarm FUNCTION CODES:***A\$Delete****Remove a Pending Alarm Request**

**INPUT:** d0.l = Alarm ID (or zero)  
d1.w = A\$Delete function code

**OUTPUT:** None

**FUNCTION:** A\$Delete removes a cyclic alarm or any alarm that has not expired. If zero is passed as the alarm ID, all pending alarm requests for the current process are removed.

**A\$Set****Execute a System-State Subroutine after a Specified Time Interval**

**INPUT:** d0.l = Reserved, must be zero  
d1.w = A\$Set function code  
d2.w = Reserved, must be zero  
d3.l = Time Interval  
(a0) = Register image

**OUTPUT:** d0.l = Alarm ID

**ERROR** cc = carry bit set to one

**OUPTUT:** 1.w = Error code

**FUNCTION:** A\$Set executes a system-state subroutine after the specified time interval has elapsed. The time interval may be specified in system clock ticks, or 256ths of a second. The minimum time interval allowed is two system clock ticks.

**A\$Cycle****Execute a System-State Subroutine Every N Ticks/Seconds**

**INPUT:** d0.l = reserved, must be zero  
d1.w = A\$Cycle function code  
d2.l = signal code  
d3.l = time interval

**OUTPUT:** d0.l = alarm ID

**ERROR** cc = carry bit set

**OUTPUT:** d1.w = appropriate error code

**FUNCTION:** The cycle function is similar to the set function, except that the alarm is reset after it is sent. This causes periodic execution of a system-state subroutine.

**CAVEAT:** Keep cyclic system-state alarms as fast as possible and schedule them with as long a cycle as possible to avoid consuming a large portion of available CPU time.

**A\$AtDate****Execute a System-State Subroutine at Gregorian Date/Time**

**INPUT:** d0.l = Reserved, must be zero  
d1.w = A\$AtDate function code  
d2.l = Reserved, must be zero  
d3.l = Time (00hhmmss)  
d4.l = Date (YYYYMMDD)  
(a0) = Register image

**OUTPUT:** d0.l = alarm ID

**ERROR** cc = carry bit set

**OUTPUT:** d1.w = appropriate error code

**FUNCTION:** A\$AtDate executes a system-state subroutine at a specific date and time. **NOTE:** A\$AtDate only allows you to specify time to the nearest second. However, it does adjust if the system's date and time have changed (via F\$\$Time). The alarm subroutine executes anytime the system date/time becomes greater than or equal to the alarm time.

**A\$AtJul****Execute a System-State Subroutine at Julian Date/Time**

**INPUT:** d0.l = Reserved, must be zero  
d1.w = A\$AtDate or A\$AtJul function code  
d2.l = Reserved, must be zero  
d3.l = Time (seconds after midnight)  
d4.l = Date (Julian day number)  
(a0) = Register image

**OUTPUT:** d0.l = alarm ID

**ERROR** cc = carry bit set

**OUTPUT:** d1.w = appropriate error code

**FUNCTION:** A\$AtJul executes a system-state subroutine at a specific Julian date and time. **NOTE:** A\$AtJul function only allows time to be specified to the nearest second. However, it does adjust if the system's date and time have changed (via F\$STime). The alarm subroutine is executed anytime the system date/time becomes greater than or equal to the alarm time.

**F\$AIIPD** **Allocate Process/Path Descriptor**

**ASM CALL:** OS9 F\$AIIPD

**INPUT:** (a0) = process/path table pointer

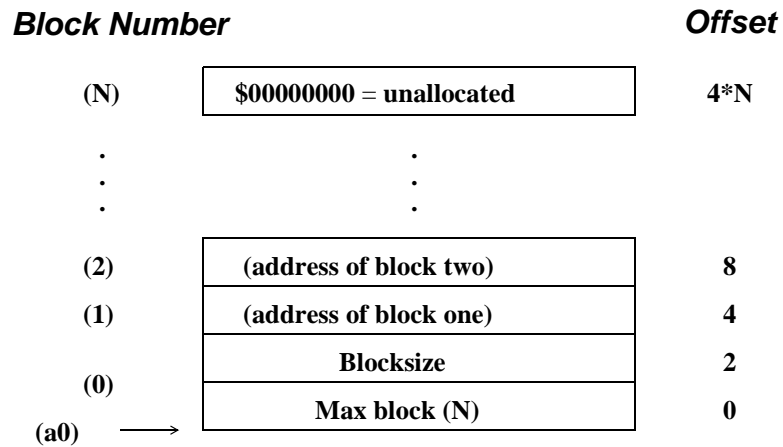
**OUTPUT:** d0.w = process/path number  
(a1) = pointer to process/path descriptor

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = error code if error

**FUNCTION:** F\$AIIPD allocates fixed-length blocks of system memory. It allocates and initializes (to zeros) a block of storage and returns its address.

It can be used with F\$FindPD and F\$RetPD to perform simple memory management. The system uses these routines to keep track of memory blocks used for process and path descriptors. They can be used generally for similar purposes by creating a map table for the data allocations. The table must be initialized as follows:



**SEE ALSO:** F\$FindPD and F\$RetPD.

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$AIIProc****Allocate Process Descriptor**

**ASM CALL:** OS9 F\$AIIProc

**INPUT:** None

**OUTPUT:** (a2) = Process Descriptor pointer

**ERROR** cc = Carry bit set.

**OUTPUT:** d1.w = Appropriate error code.

**FUNCTION:** F\$AIIProc allocates and initializes a process descriptor. The address of the descriptor is kept in the process descriptor table. Initialization consists of clearing the descriptor, setting up the state as system-state, and marking as unallocated as much of the MMU image as the system allows.

On systems without memory management/protection, this is a direct call to F\$AIIPD.

**SEE ALSO:** F\$AIIPD

**POSSIBLE**

**ERRORS:** E\$PrcFul

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST.**

**F\$AProc****Enter Process in Active Process Queue**

**ASM CALL:** OS9 F\$AProc

**INPUT:** (a0) = Address of process descriptor

**OUTPUT:** None

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** F\$AProc inserts a process into the active process queue so that it may be scheduled for execution. All processes already in the active process queue are aged. The age of the specified process is set to its priority. The process is then inserted according to its relative age. If the new process has a higher priority than the currently active process, the active process gives up the remainder of its time-slice and the new process runs immediately.

**CAVEATS:** OS-9 does not pre-empt a process that is in system state (that is, in the middle of a system call). However, OS-9 does set a bit in the process descriptor that cause it to give up its time slice when it re-enters user state.

**SEE ALSO:** F\$NProc; Chapter 2 of the *OS-9 Technical Overview*, the section on Process Scheduling

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$DelPrc****De-Allocate Process Descriptor Service Request**

**ASM CALL:** OS9 F\$DelPrc

**INPUT:** d0.w = process ID to de-allocate

**OUTPUT:** none

**ERROR** cc = carry set

**OUTPUT:** d1.w = appropriate error code

**FUNCTION:** F\$DelPrc de-allocates a process descriptor previously allocated by F\$AllPD. It is the caller's responsibility to ensure that any system resources used by the process are returned prior to calling F\$DelPrc.

Currently, the F\$DelPrc request is simply a convenient interface to the F\$RetPD service request. It is preferred to F\$RetPD to ensure compatibility with future releases of the operating system that may need to perform process specific de-allocations.

**SEE ALSO:** F\$AllPrc, F\$AllPD, F\$FindPD, and F\$RetPD.

**POSSIBLE**

**ERRORS:** E\$BNam and E\$KwnMod.

**NOTE: THIS IS A SYSTEM-STATE SERVICE REQUEST**

**F\$FindPD****Find Process/Path Descriptor**

**ASM CALL:** OS9 F\$FindPD

**INPUT:** d0.w = process/path number  
(a0) = process/path table pointer

**OUTPUT:** (a1) = pointer to process/path descriptor

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = error code if error

**FUNCTION:** F\$FindPD converts a process or path number to the absolute address of its descriptor data structure. You can use it for simple memory management of fixed length blocks. See F\$AllPD for a description of the data structure used.

**SEE ALSO:** F\$AllPd and F\$RetPd.

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$IOQu**

Enter I/O Queue

**ASM CALL:** OS9 F\$IOQu

**INPUT:** d0.w = Process Number

**OUTPUT:** None

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** F\$IOQu links the calling process into the I/O queue of the specified process and performs an untimed sleep. It is assumed that routines associated with the specified process send a wakeup signal to the calling process. IOQu is used primarily and extensively by the I/O system.

For example, if a process needs to do I/O on a particular device that is busy servicing another request, the calling process performs an F\$IOQu call to the process in control of the device. When the first process returns from the file manager, the kernel automatically wakes up the IOQu-ed process.

**SEE ALSO:** F\$FindPd, F\$Send, and F\$Sleep.

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$IRQ****Add or Remove Device from IRQ Table**

**ASM CALL:** OS9 F\$IRQ

**INPUT:** d0.b = vector number  
           25-31 for autovectors  
           57-63 for 68070 on-chip autovectors  
           64-255 for vectored IRQs  
 d1.b = priority (0 = polled first, 255 = last)  
 (a0) = IRQ service routine entry point (0 = delete)  
 (a2) = device static storage  
 (a3) = port address

**OUTPUT:** None

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** F\$IRQ installs an IRQ service routine into the system polling table. If (a0) equals zero, the call deletes the IRQ service routine, and only (d0/a0/a2) are used.

The port is sorted by priority onto a list of devices for the specified vector. If the priority is zero, only this device is allowed to use the vector. Otherwise, any vector may support multiple devices. OS-9 does not poll the I/O port prior to calling the interrupt service routine and makes no use of (a3). Device drivers are required to determine if their device caused the interrupt. Service routines conform to the following register conventions:

**INPUT:** (a2) = global static pointer  
 (a3) = port address  
 (a6) = system global data pointer (D\_'s)  
 (a7) = system stack (in active proc's descriptor)

**OUTPUT:** None

**ERROR** Carry bit set if the device did not cause the  
**OUTPUT:** interrupt.

**WARNING:** Interrupt service routines may destroy the following registers: d0, d1, a0, a2, a3, and/or a6. You must preserve all other registers used.

**SEE ALSO:** The **OS-9 Technical I/O Manual** contains more information on RBF and SCF device drivers.

**CAVEAT:** You may not put zero priority multiple auto-vectored devices on the polling list.

***POSSIBLE***

***ERRORS:*** E\$POLL is returned if the polling table is full.

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$Move****Move Data (Low Bound First)**

**ASM CALL:** OS9 F\$Move

**INPUT:** d2.1 = Byte count to copy  
(a0) = Source pointer  
(a2) = Destination pointer

**OUTPUT:** None

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** F\$Move is a fast “block-move” subroutine capable of copying data bytes from one address space to another (usually from system to user or vice versa).

The data movement subroutine is optimized to make use of long moves whenever possible. If the source and destination buffers overlap, an appropriate move (left to right or right to left) is used to avoid loss of data due to incorrect propagation.

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$NProc****Start Next Process**

**ASM CALL:** OS9 F\$NProc

**INPUT:** None

**OUTPUT:** Control does not return to caller.

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** F\$NProc takes the next process out of the Active Process Queue and initiates its execution. If there is no process in the queue, OS-9 waits for an interrupt, and then checks the active process queue again.

**CAVEATS:** The process calling NProc should already be in one of the system's process queues. If it is not, the calling process becomes unknown to the system even though the process descriptor still exists and is printed out by a `procs` command.

**SEE ALSO:** F\$AProc

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$Panic****System Catastrophic Occurrence**

**ASMCALL:** OS9 F\$Panic

**INPUT:** d0.l = panic code

**OUTPUT:** None. F\$Panic generally does not return.

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** The OS-9 kernel makes a F\$Panic request when it detects a disastrous, but not necessarily fatal, system condition. Ordinarily, F\$Panic is undefined and the system dies.

The system administrator may install a service routine for F\$Panic as part of an OS9P2 startup module. The function of such a routine might be to fork a warmstart Sysgo process or to cause the system to re-boot.

Two panic codes are defined:

**K\$Idle** The system has no processes to execute.

**K\$PFail** Power failure has been detected.

F\$Panic is called only when the kernel believes there are no processes remaining to be executed. Although it is likely the system is dead at this point, it may not be. Interrupt service routines or system-state alarms could cause the system to become active.

**NOTE:** The OS-9 kernel does not detect power failure. However, some machines are equipped with hardware capable of detecting power failure. For these machines, an OS9P2 routine could be installed to call F\$Panic when power failure occurs.

**SEE ALSO:** F\$SSvc; Chapter 2 of the **OS-9 Technical Overview**, the section on installing system-state routines.

**F\$RetPD****Return Process/Path Descriptor**

**ASM CALL:** OS9 F\$RetPD

**INPUT:** d0.w = process/path number  
(a0) = process/path table pointer

**OUTPUT:** None

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** F\$RetPD de-allocates a process or path descriptor. It can be used in conjunction with F\$AllPD and F\$FindPD to perform simple memory management of other fixed length objects.

**SEE ALSO:** F\$AllPD and F\$FindPD.

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$SSvc****Service Request Table Initialization**

**ASM CALL:** OS9 F\$SSvc

**INPUT:** (a1) = pointer to service request initialization table  
(a3) = user defined

**OUTPUT:** None

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** F\$SSvc adds or replaces function requests in OS-9's user and privileged system service request tables.

(a3) is intended to point to global static storage. This allows a global data pointer to be associated with each installed system call. Whenever the system call is invoked, the data pointer is automatically passed. Whatever (a3) points to is passed to the system call; (a3) may point to anything.

An example initialization table might look like this:

**SvcTbl**

dc.w F\$Service	<i>OS-9 service request code</i>
dc.w Routine-*-2	<i>offset of routine to process request</i>
:	
dc.w F\$Service+SysTrap	<i>redefine system level request</i>
dc.w SysRoutn-*-4	<i>offset of routine to handle system request</i>
:	
dc.w -1 end of table	

Valid service request codes range from (0-255).

If the sign bit of the function code word is set, only the system table is updated. Otherwise, both the system and user tables are updated.

You can only call privileged system service requests from routines executing in System (supervisor) state. The example above shows how a service call that must behave differently in system state than it does in user state is installed.

System service routines are executed in supervisor state, and are not subject to time-sliced task-switching. They are written to conform to register conventions shown in the following table:

**INPUT:**    **d0-d4** = user's values  
              **(a0)-(a2)** = user's values  
              **(a4)** = current process descriptor pointer  
              **(a5)** = user's registers image pointer  
              **(a6)** = system global data pointer

**OUTPUT:**    **cc** = carry set  
              **d1.w** = error code if error

The service request routine should process its request and return from subroutine with a RTS instruction. Any of the registers d0-d7 and (a0)-(a6) may be destroyed by the routine, although for convenience, (a4)-(a6) are generally left intact.

The user's register stack frame pointed to by (a5) is defined in the library sys.l and follows the natural hardware stacking order. If the carry bit is returned set, the service dispatcher sets R\$cc and R\$d1.w in the user's register stack. Any other values to be returned to the user must be changed in their stack by the service routine.

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**F\$VModul****Validate Module**

**ASM CALL:** OS9 F\$VModul

**INPUT:** d0.l = beginning of module group (ID)  
d1.l = module size  
(a0) = module pointer

**OUTPUT:** (a2) = Directory entry pointer

**ERROR** cc = Carry bit set

**OUTPUT:** d1.w = Appropriate error code

**FUNCTION:** F\$VModul checks the module header parity and CRC bytes of an OS-9 module.

If the header values are valid, the module is entered into the module directory, and a pointer to the directory entry is returned.

The module directory is first searched for another module with the same name. If a module with the same name and type exists, the one with the highest revision level is retained in the module directory. Ties are broken in favor of the established module.

**SEE ALSO:** F\$CRC and F\$Load.

**POSSIBLE**

**ERRORS:** E\$KwnMod, E\$DirFul, E\$BMID, E\$BMCRC, and E\$BMHP.

**NOTE: THIS IS A PRIVILEGED SYSTEM-STATE SERVICE REQUEST**

**End of Chapter 3**