Power Management Subsystem Specification

Version 4.7
Copyright and publication information

This manual reflects version 4.7 of Microware OS-9. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.
Contents

Overview 5

Definitions ........................................................................................................................................................................ 6
Power Unaware ........................................................................................................................................................................... 6
Power Aware ............................................................................................................................................................................. 6
Support Mechanisms ........................................................................................................................................................... 6
A Power Managed System ...................................................................................................................................................... 6
Power Aware Applications ....................................................................................................................................................... 7
Power Management Subsystem ............................................................................................................................................... 8
Power Aware Device Drivers .................................................................................................................................................. 8
Power Management Subsystem Components .......................................................................................................................... 8
  PwrMan................................................................................................................................................................................... 8
  PwrPlcy................................................................................................................................................................................... 8
  PwrExt................................................................................................................................................................................... 9
  SysIF................................................................................................................................................................................... 9
Interaction in the Power Managed System ................................................................................................................................ 9

PwrMan 11

PwrMan Tables ..................................................................................................................................................................... 12
  Device Registry Table ............................................................................................................................................................ 12
  Power State Table ............................................................................................................................................................... 13
Power Structures and Definitions ............................................................................................................................................. 15
  Device Energy Condition ....................................................................................................................................................... 16
  Local Power Mode .............................................................................................................................................................. 17
  Power Event .......................................................................................................................................................................... 17
pwrman_globals Structure ....................................................................................................................................................... 18
PwrMan Library ....................................................................................................................................................................... 20
  _os_pwr_add() ................................................................................................................................................................. 24
  _os_pwr_callback() .......................................................................................................................................................... 25
  _os_pwr_change() .............................................................................................................................................................. 26
  _os_pwr_check() ................................................................................................................................................................. 27
  _os_pwr_copyglob() ......................................................................................................................................................... 28
  _os_pwr_debug() .............................................................................................................................................................. 29
  _os_pwr_ev_notify() ......................................................................................................................................................... 30
  _os_pwr_ev_request() ...................................................................................................................................................... 31
  _os_pwr_link_ext() ......................................................................................................................................................... 32
  _os_pwr_link_plcy() ...................................................................................................................................................... 33
  _os_pwr_reg() ................................................................................................................................................................. 34
  _os_pwr_remove() ........................................................................................................................................................... 35
  _os_pwr_unlink_ext() ...................................................................................................................................................... 36
  _os_pwr_unlink_plcy() .................................................................................................................................................. 37
  _os_pwr_unreg() ............................................................................................................................................................... 38
pwrstat Utility .......................................................................................................................................................................... 38
pwrstat ................................................................................................................................................................................... 39
Power Management Subsystem Specification

PwrPlcy 43
- Default Idle Loop ............................................................................................................................................................................. 44
- Customizing the Idle Function for OS-9 ........................................................................................................................................... 44
- Customizing the Idle Function for OS-9 for 68K .................................................................................................................................. 44
- PwrPlcy Example ................................................................................................................................................................................ 45
- pwrplcy.c .......................................................................................................................................................................................... 48

PwrExt 57
- pwrext.c .......................................................................................................................................................................................... 58

SysIF 63
- cpu.c ............................................................................................................................................................................................... 64
- rtclock.c .......................................................................................................................................................................................... 78

Programming Guidelines 85
- Boot Code ...................................................................................................................................................................................... 86
- Applications .................................................................................................................................................................................... 89
- Device Drivers ................................................................................................................................................................................ 89
- File Managers ................................................................................................................................................................................ 89

OS-9 for 68K 91
- 68328 (OS-9/68000) Hardware Interface ..................................................................................................................................... 92

PowerPC 93
- PPC821 (OS-9/ PPC) Hardware Interface .................................................................................................................................... 94

Assembly Interface for OS-9 for 68K 97
- FSPwrMan .................................................................................................................................................................................... 98
- Sub-Codes ........................................................................................................................................................................................ 98
- Internal Structures ........................................................................................................................................................................... 99
- Parameter Block Definitions .......................................................................................................................................................... 100

SuperH 103
- SH7709 (OS-9/ SH-3) Hardware Interface .................................................................................................................................... 104

Index
Overview

Power management is crucial to enabling long life from small and inexpensive batteries in personal communications devices. As an addition to the core OS-9® operating system, the Power Management Subsystem aids an embedded system, especially a portable battery-powered system, in conserving energy (and thus, extending battery life) via customizable power management policy.

As opposed to communications and human-interface (and most other interfaces an application deals with), the Power Management Subsystem provides a system-level interface to the kernel and a set of conventions used to write device drivers and applications.
Definitions

Power Management Subsystem terminology used in this manual is defined in this section.

Power Unaware

Power unaware device drivers and applications are those not managed by the Power Management Subsystem.

Power Aware

Power aware device drivers and applications are those managed by the Power Management Subsystem.

A power aware application may provide useful hints to a power management support mechanism (such as giving an initialized device a hint to power down due to anticipated non-use for a long interval).

Support Mechanisms

Low-level support mechanisms are used by the Power Management Subsystem to implement power management strategies. Support mechanisms include:

- Power aware device drivers
- Power aware file managers
- Power aware system modules

Different levels of power management may be implemented. For example, a liquid crystal display (LCD) screen driver may implement an internal timer to blank the screen upon a predefined period of non-use, or a hard disk driver may implement a similar internal timer to turn off the hard disk motor upon a predefined period of non-access to the disk. In both of these cases, power management is implemented totally in local device drivers. But in cases where power management crosses subsystem boundaries, such as turning off the system clock, the Power Management Subsystem, based upon entries in the Power Policy Table, implement this power management.

A Power Managed System

A Power Managed system commonly includes:

- Kernel and Power Management Subsystem
- Applications
- Device Drivers
- File Managers
- System Modules
The Power Management Subsystem is customizable. Documentation and example sources enabling development of power aware device drivers and applications and power management policy are provided in this manual.

Applications, device drivers, file managers, and system modules in a power managed system may be either power-aware or -unaware.

**Figure 1-1. A Power Managed System**

By implementing various levels of the distributed power management support mechanisms depicted in Figure 1-1, a system designer can be as aggressive as necessary in conserving power in a portable communications device.

**Power Aware Applications**

The highest power management level — power aware applications — implements power-down of devices consuming high power (or even medium- to low-power consumption devices if aggressively conserving power) when the device is not in use. For example, a serial port is powered-up when initialized since the serial driver must be ready to receive or transmit data. The application, however, knows the characteristics of data transmitted and should power down the serial driver when it determines that transmission is not expected for a long period of time.
Power Management Subsystem

The Power Management Subsystem supports power management at the operating system. Upon recognizing the absence of active processes (all processes are either sleeping or waiting for an external event) the Power Management Subsystem can place the system into a very low powered state by turning off the system clock and powering down unnecessary chips and devices.

Power Aware Device Drivers

The lowest power management level—power aware device drivers—enables software that interfaces with hardware (e.g., device driver) to be aware of the power consumption characteristics of its local subsystem. For example, an audio module subsystem can be initialized yet powered down until it is actually in use. In this example, the audio module subsystem would be powered up before playing the first note of a song and then powered back down after playing the last note.

Power Management Subsystem Components

The Power Management Subsystem components identified following are described in this section:

- Power Management Module (PwrMan™)
- Power Policy Module (PwrPlcy™)
- Power Extension Module (PwrExt™)
- System Interface Module (SysIF™)

![Figure 1-2. Power Management Subsystem Components](image)

PwrMan

PwrMan, a generic object code module available across all supported OS-9 processor platforms, provides a consistent system level interface including library functions, device and power information tables, and a status utility. PwrMan is designed to support a wide variety of power management policies by providing a generic, system-level interface for:

- Establishing, implementing, and maintaining policy
• Maintaining and transitioning between power states

For more information, see Chapter 2 PwrMan.

PwrPlcy

PwrPlcy is provided as generic source code that is customizable by original equipment manufacturers (OEMs). It is highly dependent on target system hardware configurations and the specific power management strategy chosen for that system. PwrPlcy is the high level mechanism responsible for:

• Decision making for power state changes
• Initializing PwrMan power states
• Instigating power state transitions, based upon past and current system states, via library functions

For more information, see Chapter 3 PwrPlcy.

PwrExt

PwrExt is a system-state subroutine module used to override the PwrMan system call. At initialization, PwrMan looks for both PwrExt and PwrPlcy and if either or both are in memory, PwrMan links first to PwrExt and then PwrPlcy.

For more information, see Chapter 4 PwrExt.

SysIF

SysIF is a special system module providing a system-specific interface for the microprocessor and other hardware components without device driver interfaces.

SysIF enables PwrPlcy to reference the power levels of the CPU (such as normal operation, throttled, and deep-sleep) just as it references other device-driver components (such as a serial or ethernet drivers).

For larger power managed systems, several SysIF type modules may be needed. For example, the system could have a CPU system module that controls all CPU-specific operations and a separate battery system module that controls a smart battery.

For more information, see Chapter 5 SysIF.

Interaction in the Power Managed System

Figure 1-3 depicts logical and direct links between components in a power managed system.
All power-aware and -unaware device drivers and applications communicate with the kernel. Power aware device drivers and applications may communicate directly with the Power Management Subsystem for power management requests. Power unaware device drivers may request power management via the kernel to the Power Management Subsystem if a **SysIF** exists in the system to power manage that device.

**Figure 1-3. Interaction in the Power Managed System**
PwrMan, a system module kernel extension, provides a consistent system level interface for power management. PwrMan includes library functions, device and power information tables, and a status utility, available across all supported processor platforms. PwrMan is designed to support a wide variety of power management policies by providing a generic, system-level interface for:

- Establishing, implementing, and maintaining policy
- Transitioning between power states

This chapter covers the following system level interface information:

- Device Registry Table
- Power State Table
- Power Structures
- PwrMan Globals Structure
- PwrMan Library
- pwrstat Utility

See Appendix C Assembly Interface for OS-9 for 68K.
PwrMan Tables

PwrMan comprises two table types:

- Device Registry
- Power State

The Device Registry Table identifies power aware devices to be power managed and device driver or SysIF callback functions and parameters.

See the Device Registry Table and Power State Table subsections in this chapter.

Callback functions provide hardware specific power management interfaces.

The Power State Table identifies device power control.

To power manage a device identified in the Device Registry Table, a corresponding entry must exist in the Power State Table. The common field between the two tables is the device name (or id). To create a unique entry (based upon ID) in a table and a corresponding entry in the other table, the _os_pwr_reg() and _os_pwr_add() library functions must be used.

Updates to entries (state additions, device or state deletions, or device or state changes), not creation of unique entries, to either of the tables causes PwrMan to dynamically update the other table.

The PwrPlcy module initializes and manages entries in the Power State Table. By using two tables, the order of PwrMan table initialization is irrelevant; devices may register with PwrMan (via the Device Registry Table) before the Power State Table is initialized, or vise-versa.

To view table entries, use the pwrstat utility described in this chapter.

Device Registry Table

An example Device Registry Table is shown in Table 2-1. An initial allocation of 16 Device Registry Table entries is made by PwrMan. If an attempt is made to register more than sixteen entries, PwrMan allocates additional entries in 16 block increments.

<table>
<thead>
<tr>
<th>id</th>
<th>func</th>
<th>funcparam</th>
<th>devpwrdef</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;t2&quot;</td>
<td>&lt;t2 function&gt;</td>
<td>&lt;t2 parameter&gt;</td>
<td>&lt;pointer&gt;</td>
</tr>
<tr>
<td>&quot;cpu&quot;</td>
<td>&lt;cpu function&gt;</td>
<td>&lt;cpu parameter&gt;</td>
<td>&lt;pointer&gt;</td>
</tr>
</tbody>
</table>
Device Registry Table data is formulated from device drivers and `SysIF` calls to `_os_pwr_reg()` and `_os_pwr_unreg()` library functions.

`_os_pwr_reg()` adds an entry to the Device Registry Table and checks the Power State Table to determine if the same `id` exists in both tables. If so, the function and parameter fields for the entry in the Power State Table are set to the callback function and parameters identified in the Device Registry Table. If not, the Power State Table is not updated.

`_os_pwr_unreg()` removes an entry from the Device Registry Table.

`id` is the name of the table entry. This is the common link between the Device Registry Table and the Power State Table. This field is a case-sensitive ASCII string.

`func` is a pointer to the callback power management registered by the driver or system module.

`funcparam` is a device-specific parameter, usually consisting of a device static pointer or device entry pointer.

`devpwrdef` is a pointer back to the local device energy condition structure.

### Power State Table

The main goal of `PwrMan` is to maintain the Power State Table. The initialization of this table is left to `PwrPlcy`.

An example Power State Table is shown in Figure 2-1. An initial allocation of 32 Power State Table entries is made by `PwrMan`. If more than 32 entries are added, `PwrMan` allocates additional entries in 32-block increments.
`_os_pwr_add()` adds an entry to the Power State Table and determines if the same ID is in the Device Registry Table. If not, the function and parameter fields for the entry in the Power State Table are set to NULL.

`_os_pwr_remove()` removes an entry from the Power State Table.

_id_ is the name of the entry. This is the common link between the Device Registry Table and the Power State Table. This field is a case-sensitive ASCII string.

_syslevel_ is the power state name.

_devlevel_ is a parameter for the callback function that specifies which part of the callback function is executed. For example, the same callback function may be used for several power states of a device. _devlevel_ can indicate which power state (syslevel) is used.
priority designates the order in which the callback function \texttt{devlevels} specified in the \texttt{syslevel} execute upon a state change. The lower the value, the higher the priority (e.g., 1 is highest priority).

\texttt{devpb} is an optional parameter block for the callback function. \texttt{devpb} may specify additional details concerning which part of the callback function is executed. For example, when using a PCMCIA card, this parameter may be used for whatever device is plugged into the card at that time. If \texttt{devpb} is not in use, \texttt{devpb} must be filled with NULL.

\texttt{func} is a pointer to the callback function registered by the driver or system module. \texttt{funcparam} is a device-specific parameter, usually consisting of a device static pointer or device entry pointer.

The \texttt{devlevel} and the \texttt{devpb} Power State Table entries are device specific. Power state entries for the same power state can (and probably will) have different power parameter values between them. This implies that PwrMan should be aware of all the devices that are in the system as well as their power characteristics (via the Power State Table and Device Registry Table).

\texttt{devpb} can be used for anything needed by the driver. For example, it could be used for a parameter block to change a system clock rate independent of the current system state.

\subsection*{Power Management Power State Table Flow}

In addition to the Power State Table, PwrMan maintains a pointer to the current power state, \texttt{currlevel}, in the \texttt{pwrman\_globals} structure. Requests to change power states are made by PwrPlcy to PwrMan. When changing states, PwrMan:

1. Changes the current state pointer to the new \texttt{syslevel}
2. Calls each power state entry callback function in the \texttt{syslevel} with \texttt{devpb} and \texttt{funcparam}

If errors are returned from these callback functions and the \texttt{ret\_on\_err} global flag is not set, PwrMan continues to process the remaining entries in that power state but returns the first error encountered to PwrPlcy. Upon notification of an error during state change, PwrPlcy determines whether to return to the previous power state or remain in the current one. If an error occurs while trying to change states and \texttt{ret\_on\_err} is set, PwrMan restores the previous state and immediately returns an error.

\section*{Power Structures and Definitions}

Power Management Subsystem power structures are defined in this section. Structures are:

- Device Energy Condition
Device Energy Condition

The `pwr_devcond` structure resides in the `<MWOS/SRC/DEFS/PWRMAN/pwrman.h>` header file. This structure may be used to describe the current energy/power condition of a device. The data represents the device’s current power condition. PwrExt may use the information in calculating the entire system’s energy condition or in determining if there is a need to generate a power event (`pwr_event`). A device driver or SysIF may update the `pwr_devcond` structure without generating a `pwr_event`.

Power events are unrelated to OS-9 or OS-9 for 68K events used in interprocess communication.

For a definition of `pwr_event`, reference the Power Event subsection in this chapter.

The `pwr_devcond` structure may be part of a Device Registry Table entry if used as the last parameter of the `_os_pwr_reg()` and `_os_pwr_unreg()` calls. The writer of PwrExt determines whether to use or ignore the information.

typedef struct pwr_devcond {
    pwr_localmode lpm;/* ptr to array of local power modes */
    u_int32 num_lpm;/* # of local power modes */
    pwr_devtype dev_type;/* device type */
    pwr_level pres_level;/* present power level */
    int32 pres_energy;/* present energy level (mW) */
    int32 pres_drain;/* present energy drain (mW/hr) */
    u_int8 rsv1[24];/* reserved */
} pwr_devcond, *Pwr_devcond;

`lpm` is a pointer to the local power mode of the device.
`num_lpm` is the number of local power modes for the device.
`dev_type` describes the type of device in terms of energy or power.
`pres_level` is the present local power mode level of the device.
`pres_energy` is the present energy level of the device.
`pres_drain` represents the device’s present energy drain or load on the system.

For information about local power mode, reference the Local Power Mode subsection in this chapter.
Local Power Mode

The `loc_pwr_mode` structure resides in the `<MWOS/SRC/DEFS/PWRMAN/pwrman.h>` header file. `loc_pwr_mode` is a structure used to represent local power modes of devices. This is a suggested way of abstracting how power or energy may be defined in a system, making the power management system more portable.

```c
typedef struct pwr_localmode {
    pwr_level level; /* Local power mode level */
    int32 maxload; /* max energy load or drain (mW) */
    int32 minload; /* min energy load or drain (mW) */
    u_int32 entrytime; /* worst-case device entry time (ms) */
    u_int32 exittime; /* worst-case device exit time (ms) */
    char lpm_name[PWR_IDLEN]; /* local power-mode name */
    u_int8 context_pres; /* device context preserved flag */
    u_int8 rsv1[11]; /* reserved */
} pwr_localmode, *Pwr_localmode;
```

`level` is the local power mode level.

`max_load` and `min_load` are the maximum and minimum energy loads the device may put on the system.

`entrytime` and `exittime` are the worst-case times the device takes to power-up and down respectively.

`lpm_name` is the local power mode name.

`context_pres` indicates whether the context of the device is preserved. An example where this may be used is to indicate whether a device is plugged into a PCMCIA card.

Power Event

The `pwr_event` structure resides in the `<MWOS/SRC/DEFS/PWRMAN/pwrman.h>` header file. `_os_pwr_ev_notify()` and `_os_pwr_ev_request()` are calls that use the `pwr_event` structure.

Power events are used to pass information from the device driver/SysIF level to PwrPlcy. For example, a new device is added to the system that may need a substantial amount of power or a device suddenly becomes active, such as a fax machine, and is going to consume more power as a result. Power events may be used to pass such vital information to PwrPlcy enabling PwrPlcy to make the appropriate changes in the system, if necessary.

Power events are also meant for situations that cause the system’s power condition to change. Thus, power events should be used when something that causes a substantial change in power within the system occurs. As the system’s power level hits lower and lower thresholds, the criteria for a substantial change of power in the system may change. In other words, as the energy left in a battery gets lower, more circumstances
may cause a substantial change of power in the system to occur, thus requiring a
power event to occur.

Since there is overhead involved with using power events, a system is not required to
use them. A simpler system may use interprocess communication mechanisms already
built into the operating system, such as signals and events, to indicate that something
has changed in the system.

Power events are unrelated to OS-9 or OS-9 for 68K events used in interprocess
communication.

typedef struct pwr_event {
    char* devid;/* ptr to device identifier */
    Pwr_devcond devcond;/* ptr to device condition struct */
    pwr_level old_lpm;/* ptr to old local power mode */
    pwr_level new_lpm;/* ptr to new local power mode */
    int32 old_energy;/* old energy level (mW) */
    int32 new_energy;/* new energy level (mW) */
    int32 old_drain;/* old drain[/supply] (mW/hr) */
    int32 new_drain;/* new drain[/supply] (mW/hr) */
    u_int8 rsv1[16];/* reserved */
} pwr_event, *Pwr_event;

dev_id is a pointer to the device identifier.
dev_cond is a pointer to the overall device power condition structure.
old_lpm is the local power mode before the power event takes place.
new_lpm is the local power mode that the device just entered (via _os_pwr_ev_notify())
or is requesting to enter (via _os_pwr_ev_request()).
old_energy and new_energy (used by suppliers only) represent the change in the
amount of energy stored. The associated unit is energy-based (e.g., milliwatt-hours
(mw-hour)).
old_drain and new_drain represent the change, in mw-hour, in the rate of
use/supply of energy. Based on the nature of the device being either a consumer or
supplier of energy, the old_drain and new_drain fields may represent positive or
negative values, respectively.

pwrman_globals Structure

pwrman_globals resides in <MWOS/SRC/DEFS/PWRMAN/pwrman.h>. The
structure comprises addresses of callback functions of fields in the Power State Table
and is known by all Power Management Subsystem modules: PwrPlyc, PwrMan,
PwrExt, and SysIF.

   /* Global Variables to PwrMan, pwrplcy, and pwrext */
   struct pwrman_globals {
Sysglobs sglobptr; /* system globals pointer */

Pwr_state phead; /* head of the Power State table */
Pwr_state pfree; /* power state table free pool */

Pwr_devreg dthead; /* head of the device registry table */
Pwr_devreg dtfree; /* device registry table free pool */

pwr_level currlevel; /* current system power level */

void* pwrplcy_mem; /* PwrPlcy-specific memory pointer */
error_code (*pwrplcy_func)(F_pwrman_pb pb, Pwrman_globals ldptr);
   /* PwrPlcy entry point function */
void* pwrext_mem; /* PwrExt-specific memory pointer */
error_code (*pwrext_func)(F_pwrman_pb pb, Pwrman_globals ldptr);
   /* PwrExt entry point function */

u_int8 from_usrstate; /* usr state flag */
#define FROM_SYSSTATE 0x00
#define FROM_USRSTATE 0x01

u_int8 from_super; /* super user flag */
#define FROM_NONSUPER 0x00
#define FROM_SUPER 0x01

u_int8 ret_on_err; /* return on state-change err flag */

u_int8 rsv1[1]; /* reserved */

error_code (*pwrman_func)(F_pwrman_pb pb, Pwrman_globals ldptr);
   /* PwrMan entry point function */

Mh_com pwrplcy_modhead; /* pointer to PwrPlcy module head */
Mh_com pwrext_modhead; /* pointer to PwrExt module head */

u_int8 rsv2[12]; /* reserved */

sysglob is a pointer to the system globals.
phead and dthead are pointers to the heads of the Power State and Device Registry tables respectively.
pfree and dtfree are pointers to the Power State and Device Registry tables free entry pools, respectively.
currlevel is the current power state.
pwrplcy_mem is a pointer to PwrPlcy allocated local memory.
pwrplcy_func is a pointer to the PwrPlcy entry point.
pwrext_mem is a pointer to PwrExt allocated local memory.
pwrext_func is a pointer to the PwrExt entry point.
from_usrstate is a flag indicating run a process from user state.
from_super is a flag indicating that super user is required to run the process.
ret_on_err is a flag error handling when one of the callback functions error during an _os_pwr_change(). If ret_on_err is set and a callback function errors, the previous state is restored and the error is returned immediately. If ret_on_err is not set and a callback function errors, _os_pwr_change() continues to call the remaining callback functions for that state and then returns the first error.
pwrman_func is the pointer to the default entry point of the F$PwrMan system call.
pwrexcy_modhead holds a pointer to a module head for unlinking.
pwrext_modhead holds a pointer to a module head for unlinking.

PwrMan Library

Power Management C library function bindings reside in MWOS/<OS>/<CPU>/LIB/pwrman.l.

<OS>=OS9
OS9000
<CPU>=68000
PPC
80386

This library provides an interface for all PwrMan operations. All power management macro, structure, and type definitions reside in MWOS/SRC/DEFS/pwrman.h.

PwrMan library information for assembly language users is provided in Appendix C Assembly Interface for OS-9 for 68K.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_code _os_pwr_add(char id[PWR_IDLEN], pwr_level syslevel, u_int32 priority, pwr_level devlevel, void *devpb)</td>
<td>Adds an entry in the Power State Table.</td>
</tr>
<tr>
<td>error_code _os_pwr_callback(char id[PWR_IDLEN], pwr_level devlevel, void *devpb)</td>
<td>Calls either the SysIF or driver callback function without changing states.</td>
</tr>
<tr>
<td>error_code _os_pwr_change(pwr_level syslevel)</td>
<td>Changes the current power state to the one indicated as syslevel.</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>error_code _os_pwr_check(char version[PWR_VERS_LEN])</td>
<td>Checks to see if PwrMan is active in the system.</td>
</tr>
<tr>
<td>error_code _os_pwr_copyglob(pwrman_globals *pwrglob)</td>
<td>Retrieves a copy of the internal PwrMan globals (for debugging use only).</td>
</tr>
<tr>
<td>error_code _os_pwr_debug(const pwrman_globals **pwrglob)</td>
<td>Retrieves a pointer to the internal PwrMan globals (for debugging use only).</td>
</tr>
<tr>
<td>error_code _os_pwr_ev_request(Pwr_event pwrevent)</td>
<td>Sends PwrPlcy a power event requesting a change in state.</td>
</tr>
<tr>
<td>error_code _os_pwr_link_ext(char id[PWR_IDLEN])</td>
<td>Link to a PwrExt module.</td>
</tr>
<tr>
<td>error_code _os_pwr_link_plcy(char id[PWR_IDLEN])</td>
<td>Link to a PwrPlcy module.</td>
</tr>
<tr>
<td>error_code _os_pwr_ev_notify(Pwr_event pwrevent)</td>
<td>Sends PwrPlcy a power event indicating that a change of state must occur.</td>
</tr>
<tr>
<td>error_code _os_pwr_reg(char id[PWR_IDLEN], error_code (*func)(void *funcparam, pwr_level devlevel, void *devpb), void *funcparam, Pwr_devcond devpwrdef)</td>
<td>Adds an entry in the Device Registry Table.</td>
</tr>
<tr>
<td>error_code _os_pwr_remove(char id[PWR_IDLEN], pwr_level syslevel, u_int32 priority, pwr_level devlevel, void *devpb)</td>
<td>Removes an entry from the Power State Table.</td>
</tr>
<tr>
<td>error_code _os_pwr_unlink_ext(void)</td>
<td>Unlink from the attached PwrExt module.</td>
</tr>
<tr>
<td>error_code _os_pwr_unlink_plcy(void)</td>
<td>Unlink from the attached PwrPlcy module.</td>
</tr>
<tr>
<td>error_code _os_pwr_unreg(char id[PWR_IDLEN], error_code (*func)(void *funcparam, pwr_level devlevel, void *devpb), void *funcparam, Pwr_devcond devpwrdef)</td>
<td>Removes an entry from the Device Registry Table.</td>
</tr>
</tbody>
</table>
Table 2-3. PwrMan Function Overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Processor State</th>
<th>Group Permissions *</th>
<th>Typical Calling Entity/Use</th>
<th>Device Driver/SysIF</th>
<th>Power Aware Application</th>
<th>Debug/Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_pwr_add()</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_os_pwr_callback()</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_os_pwr_change()</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_os_pwr_check()</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_os_pwr_copyglob()</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_os_pwr_debug()</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_os_pwr_ev_notify()</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_os_pwr_link_ext()</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_os_pwr_link_plcy()</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_os_pwr_reg()</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_os_pwr_remove()</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_os_pwr_ev_request()</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_os_pwr_unlink_ext()</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_os_pwr_unlink_plcy()</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_os_pwr_unreg()</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Permissions are not relevant to system state.
<table>
<thead>
<tr>
<th>Number</th>
<th>Error</th>
<th>Functions</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>208</td>
<td>EOS_UNKSVC</td>
<td>All</td>
<td>P2init PwrMan prior to using PwrMan library functions.</td>
</tr>
<tr>
<td>221</td>
<td>EOS_MNF</td>
<td>_os_pwr_link_ext() _os_pwr_link_plcy()</td>
<td>The module specified by id in the function syntax does not exist.</td>
</tr>
<tr>
<td>231</td>
<td>EOS_KWNMOD</td>
<td>_os_pwr_unlink_ext() _os_pwr_unlink_plcy()</td>
<td>Unlink from attached PwrExt module using <em>os_pwr_unlink</em>&lt;module&gt;() before linking to the named module.</td>
</tr>
</tbody>
</table>
_os_pwr_add()
Add an Entry in the Power State Table

Syntax

#include <PWRMAN/pwrman.h>

error_code _os_pwr_add(char id[PWR_IDLEN], pwr_level syslevel,
u_int32 priority, pwr_level devlevel, void *devpb);

Description

_os_pwr_add() creates an entry to the Power State Table for PwrMan and
determines if the same ID is in the Device Registry Table. If so, the function and
parameter fields for the entry in the Power State Table are set to the callback function
and parameters identified in the Device Registry Table. If not, the function and
parameter fields for the entry in the Power State Table are set to NULL.

id is the name of the entry. This is the common link between the Device Registry Table
and the Power State Table. This field is a case-sensitive ASCII string.

syslevel is the power state level used for defining the state in which the entry is to
be placed.

priority designates the order in which the callback function devlevels within the
syslevel execute upon a state change. The lower the value, the higher the priority
(e.g., 1 is highest priority).

devlevel is a parameter for the callback function that specifies which part of the
callback function is executed. For example, the same callback function may be used for
several power states of a device. devlevel can indicate which power state
(syslevel) is used.

devpb is an optional parameter block for the callback function. devpb may specify
additional details concerning which part of the callback function is executed. For
example, when using a PCMCIA card, this parameter may be used for whatever device
is plugged into the card at that time. If devpb is not in use, devpb must be filled with
NULL.

Library

pwrman.l

See Also

_os_pwr_remove()
Syntax

```c
#include <PWRMAN/pwrman.h>

error_code _os_pwr_callback(char id[PWR_IDLEN], pwr_level devlevel, void *devpb);
```

Description

_os_pwr_callback() calls a callback function without doing a power state change. The id is the name of the entry. This is the common link between the Device Registry Table and the Power State Table. This field is a case-sensitive ASCII string.

`devlevel` is a parameter for the callback function that specifies which part of the callback function is executed. For example, the same callback function may be used for several power states of a device. `devlevel` can indicate which power state (`syslevel`) is used.

`devpb` is an optional parameter block for the callback function. `devpb` may specify additional details concerning which part of the callback function is executed. For example, when using a PCMCIA card, this parameter may be used for whatever device is plugged into the card at that time. If `devpb` is not in use, `devpb` must be filled with NULL.

Library

pwrman.l
_os_pwr_change()
Change Power State

Syntax

#include <PWRMAN/pwrman.h>

error_code _os_pwr_change(pwr_level syslevel);

Description

_os_pwr_change() changes the current power state pointer, currlevel, to the
specified syslevel and calls the callback functions of each of the Power State Table
entries in that syslevel.

If the global flag, ret_on_err within the pwrman_globals structure is set, then
_os_pwr_change() returns upon the first error and restores the previous power
state. If ret_on_err is not set, os_pwr_change() returns the first error and
continues with the power state change.

syslevel is the name of power state to which to transition.

Library

pwrman.l
Chapter 2: PwrMan

__os_pwr_check__
Determine if PwrMan is Enabled

Syntax

```c
#include <PWRMAN/pwrman.h>

error_code __os_pwr_check(char version[PWR_VERS_LEN]);
```

Description

__os_pwr_check__ determines if PwrMan is installed in the system and, if so, returns the version number.

`version` is the version number of PwrMan upon returning SUCCESS. `version` is a character array buffer allocated by the caller and filled by PwrMan. If the `version` is NULL, then the buffer is not filled but the function returns successfully.

Library

pwrman.1
_os_pwr_copyglob()
Retrieves a Copy of the Internal PwrMan Globals

Syntax
#include <PWRMAN/pwrman.h>

error_code _os_pwr_copyglob(pwrman_globals *pwrglob);

Description
_os_pwr_copyglob() retrieves a copy of the internal PwrMan globals.

Library
pwrman.l

See Also
_os_pwr_debug()
Chapter 2: PwrMan

(os_pwr_debug())
Retrieves a Pointer to the Internal PwrMan Globals

Syntax

#include <PWRMAN/pwrman.h>

error_code _os_pwr_debug(const pwrman_globals **pwrglob);

Description

_os_pwr_debug() passes a pointer to the PwrMan globals. This call is useful for debugging.

- pwrglob is a read-only pointer to PwrMan globals.

\[\text{pwrglob} \text{ is constant structure. The fields in this structure should not be changed.}\]

Library

pwrman.l

See Also

_os_pwr_copyglob()
_os_pwr_ev_notify()
Send a Notification Power Event

Syntax
#include <PWRMAN/pwrman.h>

error_code _os_pwr_ev_notify(Pwr_event pwrevent);

Description
_os_pwr_ev_notify() enables a driver or SysIF to pass information to PwrPlcy about a power state change that occurred or that needs to occur. For example, if a new device is plugged into the system, the driver could use _os_pwr_ev_notify() to notify PwrPlcy of the change in the system.

pwrevent is a parameter block of type pwr_event that holds vital information (such as how much power a device needs) for passing to PwrPlcy. PwrPlcy uses pwrevent to determine if additional changes must occur in the system. For example, if a new device drains a lot of power from the system, PwrPlcy may need to turn off another device or put some devices in lower power states.

Library
pwrman.l

See Also
_os_pwr_ev_request()
_os_pwr_ev_request()
Send Request Power Event

Syntax

```c
#include <PWRMAN/pwrman.h>

error_code _os_pwr_ev_request(Pwr_event pwrevent);
```

Description

_os_pwr_ev_request() enables a driver or SysIF to pass information to PwrPlcy to request a change of power state. A state change occurs if PwrPlcy approves the request.

pwrevent is a parameter block of type pwr_event that holds vital information (such as how much power a device needs) for passing to PwrPlcy. PwrPlcy uses pwrevent to determine if additional changes must occur in the system. For example, if a new device drains a lot of power from the system, PwrPlcy may need to turn off another device or put some devices in lower power states.

Library

pwrman.l

See Also

_os_pwr_ev_notify()
_os_pwr_link_ext()

Link to a PwrExt Module

Syntax

#include <PWRMAN/pwrman.h>

error_code _os_pwr_link_ext(char id[PWR_IDLEN]);

Description

_os_pwr_link_ext() links to the PwrExt module specified by id.

id is the name of the module to which to link.

Library

pwrman.l

See Also

_os_pwr_unlink_ext()
_os_pwr_link_plcy()
Link to a PwrPlcy Module

Syntax

```c
#include <PWRMAN/pwrman.h>

error_code _os_pwr_link_plcy(char id[PWR_IDLEN]);
```

Description

_os_pwr_link_plcy() links to the PwrPlcy module specified by id.

id is the name of the module to which to link.

Library

pwrman.l

See Also

_os_pwr_unlink_plcy()
_os_pwr_reg()
Add an Entry in the Device Registry Table

Syntax

```c
#include <PWRMAN/pwrman.h>

error_code _os_pwr_reg( char id[PWR_IDLEN], error_code (*func)(void *funcparam, pwr_level devlevel, void *devpb), void *funcparam, Pwr_devcond devpwrdef);
```

Description

_os_pwr_reg() creates an entry in the Power State Table for PwrMan and determines if the same ID is in the Device Registry Table. If so, function and parameter fields for the entry in the Power State Table are set to the callback function and parameters identified in the Device Registry Table. If not, the function and parameter fields for the entry in the Power State Table are set to NULL.

*id* is the name of the entry. This is the common link between the Device Registry Table and the Power State Table. This field is a case-sensitive ASCII string.

```c
error_code (*func)(void *funcparam, pwr_level devlevel, void *devpb)
```

*func* is the callback function that PwrMan calls when changing states.

*funcparam* is the parameter (or parameter block) for *func*.

*devlevel* is a parameter for the callback function that specifies which part of the callback function is executed. For example, the same callback function may be used for several power states of a device. *devlevel* can indicate which power state (*syslevel*) is used.

*devpb* is an optional parameter block for the callback function. *devpb* may specify additional details concerning which part of the callback function is executed. For example, when using a PCMCIA card, this parameter may be used for whatever device is plugged into the card at that time. If *devpb* is not in use, *devpb* must be filled with NULL.

*funcparam* is used for any device specific parameters that may be needed in the callback function, such as a device table entry pointer or a device static storage pointer.

*devpwrdef* is the pointer to a structure which defines the present energy condition of a device.

Library

pwrman.l

See Also

_os_pwr_ev_unreg(), pwr_devcond structure
Chapter 2: PwrMan

_os_pwr_remove()
Delete an Entry in the Power State Table

Syntax

#include <PWRMAN/pwrman.h>

error_code _os_pwr_remove(char id[PWR_IDLEN], pwr_level syslevel,
u_int32 priority, pwr_level devlevel, void *devpb);

Description

_os_pwr_remove() removes an entry from the Power State Table for PwrMan and
determines if the same ID is in the Device Registry Table. If so the function and
parameter fields for the entry in the Device Registry Table are set to NULL.

id is the name of the entry. This is the common link between the Device Registry Table
and the Power State Table. This field is a case-sensitive ASCII string.

syslevel is the name of power state from which to remove the entry.

priority of the entry as specified in _os_pwr_add().

devlevel of the entry as specified in _os_pwr_add().

devpb of the entry as specified in _os_pwr_add().

Library

pwrman.l

See Also

_os_pwr_add()
_os_pwr_unlink_ext()
Unlink from the Attached PwrExt Module

Syntax
#include <PWRMAN/pwrman.h>

error_code _os_pwr_unlink_ext(void);

Description
_os_pwr_unlink_ext() unlinks from the attached PwrExt module.

Library
pwrman.l

See Also
_os_pwr_link_ext()

Errors
EOS_MNF There is not a PwrExt module attached to unlink.
**Syntax**

```c
#include <PWRMAN/pwrman.h>

error_code _os_pwr_unlink_plcy(void);
```

**Description**

`_os_pwr_unlink_plcy()` unlinks from the attached PwrPlcy module.

**Library**

pwrman.l

**See Also**

`_os_pwr_ev_reg()`

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_MNF</td>
<td>There is not a PwrPlcy module attached to unlink.</td>
</tr>
</tbody>
</table>
_os_pwr_unreg()
Delete an Entry in the Device Registry Table

Syntax
#include <PWRMAN/pwrman.h>

```c
error_code _os_pwr_unreg(char id[PWR_IDLEN], error_code (*func)(void *funcparam, pwr_level devlevel, void *devpb), void *funcparam, Pwr_devcond devpwrdef);
```

Description
_os_pwr_unreg() removes an entry from the Device Registry Table for PwrMan and checks to determine if the same ID is in the Power State Table. If so, the function and parameter fields for the entry in the Power State Table are set to NULL.

id is the name of the entry. This is the common link between the Device Registry Table and the Power State Table. This field is a case-sensitive ASCII string.

```c
error_code (*func)(void *funcparam, pwr_level devlevel, void *devpb)
```

func of the entry as specified in _os_pwr_reg().
funcparam of the entry as specified in _os_pwr_reg().
devlevel of the entry as specified in _os_pwr_reg().
devpb of the entry as specified in _os_pwr_reg().

funcparam is used for any device specific parameters that may be needed in the callback function, such as a Device Registry Table entry pointer or a device static storage pointer.

devpwrdef is a pointer to a structure which defines the present energy condition of a device.

Library
pwrman.l

See Also
_os_pwr_ev_reg(), pwr_devcond structure

pwrstat Utility
The pwrstat utility enables viewing of some internal PwrMan structures, such as version label, Device Registry Table entries, and Power State Table entries. A description of the pwrstat utility and example dumps from the utility are provided in this section.
Chapter 2: PwrMan

**pwrstat**

Enables Command Line, Pseudo Power Policy Control

**Syntax**

pwrstat [<opts>]

**Description**

*pwrstat* displays the Device Registry Table and Power State Table. *pwrstat* can also add or remove states from the Power State Table.

**Options**

- `?` Display the options, function, and command syntax of *pwrstat*.
- `-a=<id> <syslevel> <devlevel> <priority>` Add a power state entry.
- `-c=<syslevel>` Change to system level.
- `-d` Display the Device Registry Table.
- `-i` P2init the PwrMan module.
- `-le[=<module>]` Link to the PwrExt module.
- `-lp[=<module>]` Link to PwrPlcy module.
- `-p` Display the Power State Table.
- `-r=<id> <syslevel> <devlevel> <priority>` Remove a power state entry.
- `-v` Print out PwrMan version.
- `-ue` Unlink from PwrExt module.
- `-up` Unlink from PwrPlcy module.

The -a and -r dynamically add or remove respectively a state from the Power State Table. Parameters to the -a and -r options are the same.

The -c option enables change of power state.

The -d option displays the Device Registry Table entries for devices registered with PwrMan.

The -i initializes the PwrMan module.

The -le[=<module>] option links to the PwrExt module specified by <module>. If only one PwrExt module exists in the Power Management Subsystem, the module name need not be specified. If a PwrExt module is linked in the Power Management Subsystem, it must be unlinked using the -ue option prior to linking a different PwrExt module.

The -lp[=<module>] option links to the PwrPlcy module specified by <module>. If only one PwrPlcy module exists in the Power Management Subsystem, the module name need not be specified. If a PwrPlcy module is linked
in the Power Management Subsystem, it must be unlinked using the -up option prior to linking a different PwrPlcy module.

The -p option displays the Power State Table entries for power states of the power aware system.

⚠️ pwrstat does not display information for unregistered devices.

The -r option removes, from the Power State Table, the power state entry specified by <id>, <syslevel>, <devlevel>, and <priority> in the option.

The -v option displays the PwrMan version number.

**Option Variables**

id is the name of the entry. This is the common link between the Device Registry Table and the Power State Table. This field is a case-sensitive ASCII string.

syslevel indicates the current power state level.

devlevel is a parameter for the callback function that specifies which part of the callback function is executed. For example, the same callback function may be used for several power states of a device. devlevel can indicate which power state (syslevel) is used.

priority designates the order in which the callback function devlevels within the syslevel execute upon a state change. The lower the value, the higher the priority (e.g., 1 is highest priority).

The pwrstat display header lists the system name and the OS-9 version number.

**Examples**

This first example displays the Device Registry Table entries.

```bash
$ pwrstat -d
```

Motorola VME147 OS-9/68K V3.0

<table>
<thead>
<tr>
<th>Id</th>
<th>C/B Ptr</th>
<th>Data Addr</th>
<th>Device Param</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev1</td>
<td>$004f048e</td>
<td>$004ed9c0</td>
<td>$000000000</td>
</tr>
<tr>
<td>dev2</td>
<td>$004f04d8</td>
<td>$004ed9c0</td>
<td>$000000000</td>
</tr>
<tr>
<td>dev3</td>
<td>$004f0520</td>
<td>$004ed9c0</td>
<td>$000000000</td>
</tr>
<tr>
<td>dev4</td>
<td>$004f056a</td>
<td>$004ed9c0</td>
<td>$000000000</td>
</tr>
</tbody>
</table>

The following example displays Power State Table entries.
$ pwrstat -p

Motorola VME147 OS-9/68K V3.0

<table>
<thead>
<tr>
<th>Id</th>
<th>Call Back Ptr</th>
<th>Call Back Param</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu</td>
<td>$00f203b0</td>
<td>0x000000010</td>
</tr>
</tbody>
</table>

The following example adds an entry to the Power State Table.

$ pwrstat -a=cpu 0x10 0x10 1 or pwrstat -acpu 0x10 0x10 1

The following example removes an entry from the Power State Table.

$ pwrstat -r=cpu 0x10 0x10 1 or pwrstat -rcpu 0x10 0x10 1
PwrPlcy is provided as generic source code that is customizable by OEMs as it is highly dependent on the target system hardware configuration as well as the specific type of power management strategy chosen for that system or project. As such, PwrPlcy, after establishing specific rules for state changing, is relatively static. Power aware applications communicate with PwrPlcy to interactively modify power management policy.

PwrPlcy is the high level mechanism responsible for:

- Power state change decision making
- Initialization of PwrMan power states
- Information in this chapter includes:
  - Default Idle Loop
  - Customizing the Idle Loop (OS-9)
  - Customizing the Idle Loop (OS-9 for 68K)
  - PwrPlcy Example

All PwrPlcy definitions, types, and function prototypes reside in the MNOS/SRC/SYMS/SYSMODS/PWRMAN/PWRPLCY/<port> directory.
Default Idle Loop

To support low-overhead power management, OS-9 for 68K and OS-9 kernels have two system globals data areas: a context switch count area and an idle callout function area.

The context switch count is initialized to zero during kernel boot-up and incremented (to its maximum allowable value) during every context switch (in the kernel currproc() routine). A power policy provider may reset this field to zero, wait for an interval, then check it again to determine if context switches occurred (and thus the system wasn’t entirely idle) during that interval.

The idle callout function area consists of a function pointer and function parameter. Both the idle callout function pointer and parameter are required. The idle callout function may consist of just a return and enter into a busy-wait loop. A power policy provider may use this idle callout function to prompt power state changes.

Customizing the Idle Function for OS-9

PwrMan may replace the kernel default idle function. Requirements of the idle loop are:

- Idle function called with interrupts masked
- Idle function called as a subroutine
- The following registers passed to the idle function:
  - (a6) = system global data pointer
  - (sr) = interrupts are masked
  - (a7) = active system stack
- Upon returning to the kernel, the following register settings are required:
  - (a7) = intact
  - (sr) = interrupts are unmasked
- All other registers must be intact. The active system stack must not be switched if the system has an MSP/ISP stack set.
  - Idle function must
  - Honor the B_NoStop flag
  - If a STOP/LPSTOP is performed, ensure that an accidental stack switch does not occur if the system is using the MSP/ISP stack set

Customizing the Idle Function for OS-9 for 68K

The following are requirements for an OS-9 for 68K idle loop.

- Idle function called with interrupts masked
- Idle function must return with interrupts enabled
PwrPlcy Example

Our example system, shown in Figure 3-1 consists of an MC68341, ROM, and RAM. This system comprises the following:

- **SysIF** module to interface with the MC68341 microprocessor core and real-time clock
- **SCFDRVR** SCF device driver to interface with the MC68341 serial I/O subsystem
- **SPFDRVR** SPF device driver to interface with the MC68341 SPI communications subsystem
- **RAMDISK** NRF device driver to provide a RAM disk

After determining the hardware configuration and device interface configuration for the target system, the power policy writer must gather all device power management characteristics information, Figure 3-2. These device power management characteristics help determine the possible types of power management policies and are used to determine PwrMan Power State Table initialization.
The overall power management strategy chosen for this example is a simple, three-state policy, Figure 3-3. This policy consists of three power states: on, idle, and suspend as identified in Table 3-1.

<table>
<thead>
<tr>
<th>State</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>All devices are powered up and the CPU (and all associated clock signals) is running.</td>
</tr>
<tr>
<td>idle</td>
<td>All devices are powered up but the CPU is in a low-powered stop (LPSTOP) state with clock signals activated.</td>
</tr>
<tr>
<td>suspend</td>
<td>All devices are powered down and the CPU is in LPSTOP with clock signals deactivated.</td>
</tr>
</tbody>
</table>

The RAM-disk driver is not specified anywhere in the power management state diagram; Since the RAM-disk is a software-only device, it does not need a power management interface in this system.
From the derived power state diagram and device power management characteristics, a designer can begin to implement the PwrPlcy module. First, PwrPlcy initializes PwrMan with a Power State Table which implements the desired power management policy, Figure 3-3. This table specifies the order and the power option to pass to registered devices whenever the corresponding power state is entered.

PwrPlcy then places an idle intercept routine into the system globals to determine (by the kernel calling this idle routine) when the system is idle (when no active processes are running). Finally, PwrPlcy calls PwrMan to change to the current (initial) power state, ON.

![Figure 3-3. Example Power State Table](image)

From this point on, the power management policy is being applied to this system. The kernel calls the idle intercept routine when it detects that no applications are running in the system. PwrPlcy registers a timed alarm, say one minute, with the OS. If the alarm goes off and applications were not activated in the interim, a transition to the SUSPEND state occurs and PwrPlcy calls PwrMan to change the current power state to IDLE. PwrMan then calls the CPU registered callback routine with “0x20” which indicates that the CPU should be placed into a SLEEP state. The CPU module must also set a real-time clock interrupt prior to the next OS timed event (as found in the d_elapse field in the OS-9 system globals) and disable the ticker interrupt.

Finally, the CPU performs an LPSTOP with system clocks still running. At this point, the CPU blocks until the next interrupt. When the CPU comes out of the LPSTOP, it checks the system active process queue to determine if any active processes are pending. If there are no active processes pending, the CPU simply reenters LPSTOP, updating the real-time clock interrupt if needed. Otherwise, the CPU returns control back to PwrMan and then back to PwrPlcy. PwrPlcy calls PwrMan to change the current power state to ON and then returns control back to the kernel.
If the SUSPEND timed alarm goes off while still in the IDLE state, PwrPlcy changes the current power state to SUSPEND. This causes PwrMan to call SCFDRVR with 0x20 indicating that the SCF driver should power down, then SPFDRVR with 0x00 indicating that the SPF driver should power down, then CPU with 0x30 indicating that the CPU should be turned off. The CPU module acts similarly to the IDLE state except that it performs an LPSTOP with the system clocks turned off. Also similar to the IDLE state, any activated process causes PwrPlcy to change the current power state back to ON and returns control to the kernel.

In contrast to the previous simple example, a more complicated power management policy may require additional states, transitional states, or even dynamically changing states depending upon the power management strategy for the target system and the implementation of the PwrPlcy module. In a very complicated system, the PwrPlcy implementor must choose between a straightforward, but large, Power State Table encompassing every possible power state of the target system and a smaller, more complex, Power State Table dynamically updated by PwrPlcy.

pwrplcy.c

Source code for pwrplcy.c follows.

```
/******************************************************************************
**  ****************************************************************************
**  ** Power Manager Policy subroutine module guts
**  **
**  ****************************************************************************
**  ** Copyright 1995 by Microwave Systems Corporation
**  **
**  ** Copyright 2001 by RadiSys Corporation
**  **
**  ** Reproduced Under License
**  **
**  **
**  **
**  ** This source code is the proprietary confidential property of
**  **
**  ** Microwave Systems Corporation, and is provided to licensee
**  **
**  ** solely for documentation and educational purposes. Reproduction,
**  **
**  ** publication, or distribution in any form to any party other than
**  **
**  ** the licensee is strictly prohibited.
**  **
**  ****************************************************************************
**  ****************************************************************************
**
** Edition History:
**```
** # Date     Comments                                          By
** --- -------- ------------------------------------------------- --
** ------ **
**  1 06/14/95 creation                                           bat
**
**  5 12/21/95 ported to PwrMan (for 68328 board)                 rmm
**
**  6 01/05/96 added in the subroutine module stuff               rmm
**
**  7 01/10/96 changed subroutine module interface to P2           rmm
**
**  8 01/15/96 split into 3 files & modules                        rmm
**
**  9 05/30/96 modified for PwrMan v1.0.Beta structures            bat
**
** 10 09/11/96 updated cast’s for new sysglob structure           bat
**
** 11 01/12/97 modified INIT and added DEINIT to pwrplcy          ecm
**
** 17/01/97 <--------------- PwrMan v2.1 ------------------------>
**
********************************************************************
********/

```
#include <defs.h>

#define _SYSEDIT

const devreg_init pwrstates[] = {
    { PWRPLCY_STATE_ON, "gfx", MPC821_PWR_LCD_ON, 1 },
    { PWRPLCY_STATE_0, "cpu", MPC821_PWR_CPU_NRMLHIGH, 0xffffffff },
};
```
{ PWRPLCY_STATE_IDLE,"cpu",MPC821_PWR_CPU_NRMLLOW,0xffffffff},

/* SUSPEND state */
{ PWRPLCY_STATE_SUSPEND,"gfx",MPC821_PWR_LCD_OFF,1},
{ PWRPLCY_STATE_SUSPEND,"cpu",MPC821_PWR_CPU_DOZELOW,0xffffffff },

/* end-of-list */
{0,0,0,0},
};

/*
** System-Call Entry Point
**
** Trying to do the following setup:
**      fpb   = parameter block containing subcode and version number
**      ldptr = Pwrman’s global structure
*/
error_code PwrPlcy_entry(F_pwrman_pb pb, Pwrman_globals ldptr)
{
    error_code err;
    Localdata plcyptr = ldptr->pwrplcy_mem;

    switch(pb->subcode) {

    /* Initialize the power state table, set up system globals,
and
    ** go through power states
    */
    case PWRPLCY_INIT:
    {
        status_reg oldirq;
        Devreg_init pwrstates_ptr;

        /* get & initialize power policy globals */
        {
            u_int32 size = sizeof(localdata);
            u_int32 color = 0;

            /* allocate some memory for power policy use */
            if ((err=_os_srqmem(&size,(void**)&plcyptr,color)) != SUCCESS) {
                return(err);
            }

            /* initialize the newly acquired memory */
        }
    }
ldptr->pwrplcy_mem = plcyptr;
}

/* load up the power state table */
{
    int i;
    for (i=0; (pwrstates[i].id[0] != 0); i++) {
        if ((err = pwr_add(ldptr,
                         (char*)pwrstates[i].id,
                         pwrstates[i].syslevel,
                         pwrstates[i].priority,
                         pwrstates[i].devlevel,
                         NULL)) != SUCCESS) {
            return(err);
        }
    }
}

/* change to our initial power state */
if ((err = pwr_change(ldptr,START_STATE)) != SUCCESS) {
    return(err);
}

/* setup ourselves to be the kernel’s idle function */
oldirq = irq_save(); irq_disable();
{
    /* save old, and set d_switches to 0 so we can see if
       ** it increments indicating there’s active processes
       ** in the system
    */
    plcyptr->old_d_switches = ldptr->sglobptr->d_switches;
    ldptr->sglobptr->d_switches = 0;

    /* get the current tick count to see if we can go idle */
    plcyptr->d_ticks = ldptr->sglobptr->d_ticks;

    /* save old, replace kernel’s idle function parameters
       with our own */
    plcyptr->old_d_idledata = ldptr->sglobptr->d_idledata;
    ldptr->sglobptr->d_idledata = ldptr;

    /* save old, and replace kernel’s idle function with our
       own */
    plcyptr->old_d_idle = ldptr->sglobptr->d_idle;
    ldptr->sglobptr->d_idle = idle;
}
irq_restore(oldirq);

return(SUCCESS);

} /* PWRPLCY_INIT */

case PWRPLCY_DEINIT:
{
    status_reg oldirq;
    Devreg_init pwrstates_ptr;

    u_int32 size;

    /* do our best to remove the pwrplcy-loaded state table */
    {
        int i;
        for (i=0; (pwrstates[i].id[0] != 0); i++) {
            if ((err = pwr_remove(ldptr,
                                    (char*)pwrstates[i].id,
                                    pwrstates[i].syslevel,
                                    pwrstates[i].priority,
                                    pwrstates[i].devlevel,
                                    NULL)) != SUCCESS) {
                /* throw away errors */
                return(SUCCESS);
            }
        }
    }

    /* change to the startup power state */
    if ((err = pwr_change(ldptr, 0x00)) != SUCCESS) {
        return(err);
    }

    /* put the kernel’s idle function back */
    oldirq = irq_save(); irq_disable();
    {
        /* put d_switches back */
        ldptr->globptr->d_switches = plcyptr->old_d_switches;

        /* put kernels idle function parameters back */
        ldptr->globptr->d_idledata = plcyptr->old_d_idledata;

        /* put kernel’s idle function back */
        ldptr->globptr->d_idle = plcyptr->old_d_idle;


```c
} irq_restore(oldirq);

{
    /* get size of memory */
    size = sizeof(localdata);

    /* deinitialize the pwrplcy memory */
    ldptr->pwrplcy_mem = NULL;

    /* deallocate memory from power policy use */
    if ((err=_os_srtmem(size,(void*)plcyptr)) != SUCCESS) {
        return(err);
    }
}
return(SUCCESS);

} /* PWRPLCY_DEINIT */

/* go to the ON state */
case PWRPLCY_STATE_ON:
{
    /* change to ON state */
    return(pwr_change(ldptr,PWRPLCY_STATE_ON));
}
/* PWRPLCY_STATE_ON */

/* go to the IDLE state */
case PWRPLCY_STATE_IDLE:
{
    /* change to IDLE state */
    return(pwr_change(ldptr,PWRPLCY_STATE_IDLE));
}
/* PWRPLCY_STATE_IDLE */

/* go to the SUSPEND state */
case PWRPLCY_STATE_SUSPEND:
{
    /* change to SUSPEND state */
    return(pwr_change(ldptr,PWRPLCY_STATE_SUSPEND));
}
/* PWRPLCY_STATE_SUSPEND */

} /* switch */
```
/ * if the call is not “switched” out, return unknown service code */
return(EOS_UNKSVC);

} /* PwrPlcy_entry */

/* Own idle routine that will replace the kernel’s idle routine.
Assume that
** the kernel has masked interrupts.
*/
void idle(void)
{
    Sysglobs sglobptr;
Pwrman_globals ldptr;
Localdata plcyptr;

    /* get global pointers */
    sglobptr = get_static();
    ldptr = (void*)(sglobptr->d_idledata);
    plcyptr = ldptr->pwrplcy_mem;

    /* check if the correct amount of time has gone by before going idle */
    if (sglobptr->d_ticks >= (plcyptr->d_ticks+IDLE_DELAY)) {
        /* if d_switches is 0, we’re going idle */
        if ((sglobptr->d_switches) == 0) {
            /* change to SUSPEND state
** This change will place the CPU in a “freeze” mode; An
** interrupt or real-time clock alarm will wake it up
** and finally return from this call; Thus, we need to
** change back to ON after this call returns.
*/
            (void)pwr_change(ldptr,PWRPLCY_STATE_SUSPEND);
            (void)pwr_change(ldptr,PWRPLCY_STATE_ON);
        }

        /* set d_switches to 0 so we can see if it increments
** indicating there’s active processes in the system
*/
        ldptr->sglobptr->d_switches = 0;

        /* get the current tick count to see if we can go idle */
plcyptr->d_ticks = ldptr->globptr->d_ticks;
}

/* re-enable interrupts before leaving back to kernel */
irq_enable();

return;
}

/*
** Local "os_pwr_add()" binding w/ direct access to PwrMan
** (eg, no system call)
*/
error_code pwr_add(Pwrman_globals ldptr, char id[PWR_IDLEN],
pwr_level syslevel, u_int32 priority, pwr_level devlevel, void*
devpb)
{
    f_pwrman_add_pb pb;

    /* set-up parameter block */
    pb.pwrcom.subcode = PWRMAN_STATEADD;
    pb.pwrcom.edition = PWR_PB_EDITION;
    pb.id = id;
    pb.syslevel = syslevel;
    pb.priority = priority;
    pb.devlevel = devlevel;
    pb.devpb = devpb;

    /* call PwrMan directly to perform "system call" */
    return((*ldptr->pwrman_func)((F_pwrman_pb)(&pb),ldptr));
}

/*
** Local "os_pwr_remove()" binding w/ direct access to PwrMan
** (eg, no system call)
*/
error_code pwr_remove(Pwrman_globals ldptr, char id[PWR_IDLEN],
pwr_level syslevel, u_int32 priority, pwr_level devlevel, void*
devpb)
{
    f_pwrman_remove_pb pb;

    /* set-up parameter block */
    pb.pwrcom.subcode = PWRMAN_STATEREMOVE;
    pb.pwrcom.edition = PWR_PB_EDITION;
    pb.id = id;
pb.syslevel = syslevel;
pb.priority = priority;
pb.devlevel = devlevel;
pb.devpb = devpb;

/* call PwrMan directly to perform “system call” */
return((*ldptr->pwrman_func)((F_pwrman_pb)(&pb),ldptr));
}

/*
** Local “os_pwr_change()” binding w/ direct access to PwrMan
** (eg, no system call)
*/
error_code pwr_change(Pwrman_globals ldptr, pwr_level syslev)
{
    f_pwrman_change_pb pb;

    /* set-up parameter block */
    pb.pwrcom.subcode = PWRMAN_CHANGE;
    pb.pwrcom.edition = PWR_PB_EDITION;
    pb.syslevel = syslev;

    /* call PwrMan directly to perform “system call” */
    return((*ldptr->pwrman_func)((F_pwrman_pb)(&pb),ldptr));
}
PwrExt is a system-state subroutine that may override PwrMan. During initialization, PwrMan looks for both PwrExt and PwrPlcy. If either or both are in memory during initialization, PwrMan links to PwrExt first and PwrPlcy second.

For more information, see Chapter 2 PwrMan.

PWREXT_INIT and PWRPLCY_INIT run at initialization. These should contain an initialization code to run. If an initialization code is not necessary, return (SUCCESS).

Furthermore, if PwrExt is found in the system upon initializing PwrMan, pwrext_func in the pwrman_globals structure is initialized to the address of the PwrExt entry point. If a PwrExt module is not in the system upon initializing PwrMan, pwrext_func is null. When an $F$PwrMan call occurs, the function held in pwrman_globals pwrman_func is called. Thus, if PwrExt is in the system and its entry point is in pwrman_func, any PwrMan call can be replaced by a PwrExt call.

Figure 4-1 and Figure 4-2 illustrate flow diagrams with and without PwrExt in the system, respectively.

In the PwrExt initialization routine (PWREXT_INIT), the following occurs:

- The PwrMan entry point (pwrman_entry) is saved in PwrExt static storage (pwrext_mem) and is replaced by pwrext_entry.

To customize the _os_pwr_change() system call, the PWRMAN_CHANGE subcode in PwrExt is used to override the PwrMan PWRMAN_CHANGE subcode. PwrExt should save the PwrMan entry point and call it as a default case for calls that are not replaced within PwrExt.

```c
error_code Pwrext_entry(F_pwrman_pb pb) {
    switch(pb->subcode) {
        case PWREXT_INIT:
            SavePwrmanEntry(pb);
            ReplacePwrmanEntry(pb);
            break;
        case PWRMAN_CHANGE:
            MyCustomPwrChange(pb);
            break;
        default:
            CallPwrmanEntry(pb);
    }
}
```

For more information, see Chapter 2 PwrMan.
pwrext.c

/***************************************************************************/
/**********
** Power Manager Extensions entry points
**
*****************************************************************************/
/**********
** Copyright 1995 by Microware Systems Corporation
**
** Copyright 2001 by RadiSys Corporation
**
** Reproduced Under License
**
**
**
** This source code is the proprietary confidential property of  
**
** Microware Systems Corporation, and is provided to licensee  
**
** solely for documentation and educational purposes. Reproduction,  
**
** publication, or distribution in any form to any party other than  
**
** the licensee is strictly prohibited.
**
********************************************************************
**********
** Edition History:
**
** # Date Comments By
** --- --------  ------------------------------------------------- --
------- **
** 1 06/14/95 creation bat
**
** 5 12/21/95 ported to PwrMan (for 68328 board) rmm
**
** 6 01/05/96 added in the subroutine module stuff rmm
**
** 7 01/10/96 changed subroutine module interface to P2 rmm
**
** 8 01/15/96 split into 3 files & modules rmm
**
** 9 05/31/96 updated for PwrMan v1.0.Beta structures bat
**
** 10 09/11/96 updated cast’s for new sysglob structure bat
**
** 11 01/12/97 added DEINIT routine for pwrman detach ecm
**
** 01/17/97 <-------------- PwrMan v2.1 --------------->
**
********************************************************************
*********/

/*
** Header Files
*/

#define _SYSEDIT
#include <defs.h>
/* Trying to do the following setup:
**    fpb   = parameter block containing subcode and version number
**    ldptr = Pwrman’s global structure
*/

/*
** System-Call Entry Point
*/
error_code PwrExt_entry(F_pwrman_pb pb, Pwrman_globals ldptr)
{
    error_code err;
    Localdata extptr = ldptr->pwrext_mem;

    switch(pb->subcode) {

    /* Initialize pwrext module - srqmem() some memory for globals
    ** to be stored in. Store Pwrman’s entry point in this area to
    ** be called as a default case.
    */
    case PWREXT_INIT:
    {
        u_int32 size = sizeof(localdata);
        u_int32 color = 0;

        /* if we need more than 4 bytes for local “global”
        variables,
        ** then we’ll need to srqmem some memory for PwrExt use
        */

        /* assume that the current value in entrycb is PwrMan’s
        ** entry point. srqmem() some memory and grab it to use
        ** as a default case for any calls not customized by PwrExt
        */
        if ((err=_os_srqmem(&size,(void**)&extptr,color)) != SUCCESS) {
            return(err);
        }

        /* initialize the newly acquired memory & install ourselves
        ** as the default PwrMan entry point, saving the previous
        ** default in our globals
        */
        extptr->pwrman_func = ldptr->pwrman_func;
        ldptr->pwrext_mem = extptr;
        ldptr->pwrman_func = PwrExt_entry;
return(SUCCESS);
} /* PWREXT_INIT */

case PWREXT_DEINIT:
{
  u_int32 size = sizeof(localdata);

  /* reinstall the saved pointer as the default PwrMan */
  /* entry point, and null the pwrext_mem pointer out */
  ldptr->pwrman_func = extptr->pwrman_func;
  ldptr->pwrext_mem = NULL;

  /* return memory to system */
  if ((err=_os_srtmem(size,(void*)extptr)) != SUCCESS) {
    return(err);
  }

  return(SUCCESS);
} /* PWREXT_DEINIT */

#if 0
  /* This is where the customized change function will go. I’m just
   * passing back a bogus error to make sure I hit the function. */
  case PWRMAN_CHANGE:
  {
    return(999);
  }

  } /* PWREXT_DEINIT */
#endif

/* if the call is not “switched” out, call PwrMan’s entry point, */
/* and have him deal with it */

return((*extptr->pwrman_func)(pb, ldptr));

} /* PwrExt_entry */
SysIF is provided as generic source code by processor family. SysIF can be customized by OEM's as it is a system-specific interface for the microprocessor and other hardware components without device driver interfaces.

SysIF implements power management functions and provides feedback to PwrPlcy about a device’s tactical power management issues (e.g., it can’t shut down right now) in the absence of a power aware device driver. SysIF enables PwrPlcy to reference the power levels of the CPU (such as normal operation, throttled, and deep-sleep) just as it references other device-driver components (such as a serial driver or ethernet driver).

During SysIF initialization, SysIF registers a power callback routine with PwrMan which PwrPlcy may then reference in its Power State Table initialization. SysIF does not install any system calls by default. However, a user-defined system call may be used if system components other than PwrMan or PwrPlcy require access to data in SysIF.

For larger power managed systems, several SysIF types may be needed. For example, the system could have a CPU P2 module that controls all the CPU-specific operations and another SysIF module that controls a smart battery.

Custom header files may be defined. Examples reside in MWOS/SRC/SYSMODS/SYSIF/<board>.

CPU and real time clock functions of a sample SysIF, based on an MPC821, follows.
** CPU.C **

 /**************************************************************************
 **********
 ** ADS821 System Module
 **
 **********
 ** Copyright 1995-1997 by Microware Systems Corporation
 **
 ** Copyright 2001 by RadiSys Corporation
 **
 ** Reproduced Under License
 **
 **
 ** This source code is the proprietary confidential property of
 ** Microware Systems Corporation, and is provided to licensee
 ** solely for documentation and educational purposes. Reproduction,
 ** publication, or distribution in any form to any party other than
 ** the licensee is strictly prohibited.
 **
 ** Edition History:
 **
 ** #   Date     Comments                                          By
 ** --- -------- ------------------------------------------------- --
 **
 **    1 02/01/96 Creation                                          jgm
 **    2 04/02/96 Ported to use with the ADS821 PPC board          jgm
 **    3 06/04/96 Cleaned up to ship out with the Beta release     jgm
 **    4 01/16/97 Modified to work with MPC821 Rev A chip          ecm
 **    01/17/97 <------------------ PwrMan v2.1 ------------------>
 **
 ****************************************************************************
 ***/

 /*
** Header Files
*/

#include <defs.h>

/**
** Definitions
*/

#define CSRCBIT (1<<10) /* it is actually bit 21, but going in reverse logic */
/* it is the CSRC bit in the PLPCRC register */

/**
** Initialize the CPU
*/
error_code cpu_init(Localdata ldptr)
{
    char device[PWR_IDLEN] = "cpu"; /* name of the device to register */
    error_code err; /* error code temp variable */
    Pwr_devconddevpwrdef=NULL; /* Device Energy Condition Definition, NULL for this Example */

    /* initialize the hardware */
    ldptr->regs = REGBASE; /* Set the actual port address */

    /* Tell PwrMan to call us on power-down */
    err = _os_pwr_reg(device,dr_pwrman,ldptr,devpwrdef);
    if (err == SUCCESS){
        /* PwrMan in system and successful -- continue */
    }
    else if (err == EOS_UNKSVC) {
        /* no PwrMan in system -- continue */
    } else {
        /* PwrMan in system and unsuccessful -- return */
            return(err);
    }
    return(SUCCESS);
}

/**
** De-Initialize the CPU
*/
error_code cpu_term(Localdata ldptr)
{ char device[PWR_IDLEN] = "cpu"; /* name of the device to register */

Pwr_devconddevpwrdef=NULL; /* Device Energy Condition Definition, NULL for this Example */

/* Tell PwrMan to forget our previous registration */
(void)_os_pwr_unreg(device,dr_pwrman,ldptr,devpwrdef);
return(SUCCESS);
}

/*
 ** Function to switch the CPU to the ON state(Normal High)
 */
error_code cpu_nrmlhigh(Localdata ldptr)
{
    status_reg oldirq;
    u_int16 state = MPC821_PWR_CPU_NRMLHIGH; /* normal high mode */

    oldirq = irq_maskget(); /* mask irqs */

    {(ldptr->regs->irq_regs.simask &= ~SIMASK_RTC); /* disable the
RTC interrupt */
     LOW(ldptr->regs->pll_regs.plprcr,CSRCBIT); /* clear the CSRC bit */
     }         irq_restore(oldirq);/* unmask irqs */

    return(SUCCESS); /* Return a success */
}

/*
 ** Function to switch the CPU to the ON(LOW) state(Normal Low)
 */
error_code cpu_nrmllow(Localdata ldptr)
{
    status_reg oldirq;
    u_int16 state = MPC821_PWR_CPU_NRMLOW; /* normal low mode */

    oldirq = irq_maskget(); /* mask irqs */

    {ldptr->regs->irq_regs.simask |= SIMASK_RTC; /* unmask the RTC
bit to enable the RTC interrupt */
     ldptr->regs->pll_regs.sccr |= (DPNLSET | POW_ENABLE); /* Set
to divide by 4 */
    /* call common function */    /* and set the PRQEN bit to
    (void)common(ldptr,state);    /* the high frequency(as defined
    by the */     /* DFNH bits) after an interrupt */
    /* DFNH bits) after an interrupt */
}
irq_restore(oldirq);    /* unmask irqs */

LOW(ldptr->regs->pll_regs.plprcr,CSRCBIT);    /* clear the CSRC bit,
go back to DFNH */
    ldptr->regs->pl1_regs.sccr &= -DFNLSET;    /* Return to the default
    ldptr->regs->irq_regs.simask &= -SIMASK_RTC;    /* disable the RTC
interrupt */

    return(SUCCESS);    /* Return a success */
}

/*
** Function to switch the CPU to the DOZE HIGH state
*/
error_code cpu_dozehigh(Localdata ldptr)
{
    status_regoldirq;
    u_int16state = MPC821_PWR_CPU_DOZEHIGH;    /* doze high mode */

    oldirq = irq_maskget();    /* mask irqs */
{
    ldptr->regs->irq_regs.simask |= SIMASK_RTC;    /* unmask the RTC
    bit to enable the RTC interrupt */
    ldptr->regs->pl1_regs.sccr |= (DFNHSET | POW_ENABLE);    /* Set
to divide by 2 */
    /* call common function */    /* and
    set the PRQEN bit to switch to */
    (void)common(ldptr,state);    /* the high frequency(as defined
    by the */     /* DFNH bits) after an interrupt */
}
    irq_restore(oldirq);    /* unmask irqs */

    ldptr->regs->pl1_regs.sccr &= -DFNHSET;    /* Return to the default
    ldptr->regs->irq_regs.simask &= -SIMASK_RTC;    /* disable the RTC
interrupt *//
return(SUCCESS); /* Return a success */
}

/**
 * ** Function to switch the CPU to the DOZE LOW state
 */
error_code cpu_dozelow(Localdata ldptr)
{
    status_reg oldirq;
    u_int16 state = MPC821_PWR_CPU_DOZELOW; /* doze low mode */

    oldirq = irq_maskget(); /* mask irqs */
    {
        ldptr->regs->irq_regs.simask |= SIMASK_RTC; /* unmask the RTC
bit to enable the RTC interrupt */
        ldptr->regs->pll_regs.sccr |= (DFNLSET | POW_ENABLE); /* Set
to divide by 4 */
        /* call common function */ /* and set the PRQEN bit to switch
to */
        (void)common(ldptr,state); /* the high frequency(as defined
by the */
        /* DFNH bits) after an interrupt */
    }
    irq_restore(oldirq); /* unmask irqs */

    LOW(ldptr->regs->pll_regs.plprcr,CSRCBIT); /* clear the CSRC bit,
go back to DFNH */
    ldptr->regs->PLL_regs.sccr &= ~DFNLSET; /* Return to the default */
    ldptr->regs->irq_regs.simask &= ~SIMASK_RTC; /* disable the RTC
interrupt */

    return(SUCCESS); /* Return a success */
}

/* WARNING!!!!!!!!!! WARNING!!!!!!!!!! WARNING!!!!!!!!!!

Only use the sleep mode if you have SRAM or you can refresh DRAM when
you
power down because sleep, deep sleep, and power down modes shut off
the memory
controller, which means you will lose what you have in memory if you
don’t
refresh it somehow. These next three functions are optional only if
you have
a way to keep memory alive and go to one of these states.
*/
Chapter 5: SysIF

*/

/*
** Function to switch the CPU to the SLEEP state
** This is only here if you have some method of preserving memory!
*/
error_code cpu_sleep(Localdata ldptr)
#if 0
{
    status_reg oldirq;
    u_int16 state = MPC821_PWR_CPU_SLEEP; /* sleep mode */

    oldirq = irq_maskget(); /* mask irqs */
    {
        ldptr->regs->irq_regs.simask |= SIMASK_RTC; /* unmask the RTC
bit to enable the RTC interrupt */

        /* Add your code here! */

        /* call common routine */
        (void)common(ldptr, state);
    }
    irq_restore(oldirq); /* unmask irqs */

    ldptr->regs->irq_regs.simask &= ~SIMASK_RTC; /* disable the RTC
interrupt */

    return(SUCCESS);
}
#else
{
    return(EOS_UNKSVC);
}
#endif

/*
** Function to switch the CPU to the DEEP SLEEP state
** This is only here if you have some method of preserving memory!
*/
error_code cpu_deep_sleep(Localdata ldptr)
#if 0
{
    status_reg oldirq;
    u_int16 state = MPC821_PWR_CPU_DEEP_SLEEP; /* deep sleep mode */
oldirq = irq_maskget(); /* mask irqs */
{
    ldptr->regs->irq_regs.simask |= SIMASK_RTC; /* unmask the RTC bit to enable the RTC interrupt */

    /* Add your code here! */

    /* call common routine */
    (void)common(ldptr,state);
}
irq_restore(oldirq); /* unmask irqs */

ldptr->regs->irq_regs.simask &= ~SIMASK_RTC; /* disable the RTC interrupt */

return(SUCCESS);
}
#else
{
    return(EOS UnKsVC);
}
#endif

/*
** Function to switch the CPU to the POWER DOWN state
** This is only here if you have some method of preserving memory!
*/
error_code cpu_pwrdwn(Localdata ldptr)
#if 0
{
    status_reg  oldirq;
    u_int16 state = MPC821_PWR_CPU_PWRDWN; /* power down mode */

    oldirq = irq_maskget(); /* mask irqs */
{
    ldptr->regs->irq_regs.simask |= SIMASK_RTC; /* unmask the RTC bit to enable the RTC interrupt */

    /* Add your code here! */

    /* call common routine */
    (void)common(ldptr,state);
}
irq_restore(oldirq); /* unmask irqs */

ldptr->regs->irq_regs.simask &= ~SIMASK_RTC; /* disable the RTC interrupt */

return(SUCCESS);
}
#else
{
    return(EOS_UNKSVC);
}
#endif

/*
** Entry point for the call back function
*/
error_code dr_pwrman(void *ldptr, pwr_level pwrlvl, void *devpb)
{
    switch (pwrlvl){ /* switch to the level requested */
    /* cpu to normal high state */
    case MPC821_PWR_CPU_NRMLHIGH:
    {
        return(cpu_nrmlhigh((Localdata)ldptr));
    }
    /* cpu to normal low state */
    case MPC821_PWR_CPU_NRMLLOW:
    {
        return(cpu_nrmllow((Localdata)ldptr));
    }
    /* cpu to doze high state */
    case MPC821_PWR_CPU_DOZEHIGH:
    {
        return(cpu_dozehigh((Localdata)ldptr));
    }
    /* cpu to doze low state */
    case MPC821_PWR_CPU_DOZELOW:
    {
        return(cpu_dozelow((Localdata)ldptr));
    }
    /* cpu to sleep state */
    case MPC821_PWR_CPU_SLEEP:
Power Management Subsystem Specification

```c
{    return(cpu_sleep((Localdata)ldptr));
}

/* cpu to deep sleep state */
case MPC821_PWR_CPU_DEEP_SLEEP:
{    return(cpu_deep_sleep((Localdata)ldptr));
}

/* cpu to power down state */
case MPC821_PWR_CPU_PWRDWN:
{    return(cpu_pwrdwn((Localdata)ldptr));
}

/* cpu driver to terminate */
case MPC821_PWR_CPU_TERM:
{    return(cpu_term((Localdata)ldptr));
}
}    return(EOS_UNKSVC);
}

/
** CPU-Management common routine

**
** ASSUME: interrupts are masked up to maximum level before calling
common()!
*/
void common(Localdata ldptr, u_int16 state)
{
    u_int32 sec = ldptr->sysglob->d_elapse / ldptr->sysglob->d_tcksec; /* compute the # */
    u_int32 active;/* of seconds to sleep for.  d_elapase */
    u_int32 statereg;/* is the shortest timed sleep in the */
    status_reg svirq;/* sleep queue & is stored as ticks */

    /* check for minimum sleep period */
    if ((ldptr->sysglob->d_elapse == 0) || (sec > MINSLEEP)) {

        /* set real-time clock alarm interrupt for any timed sleeps */
        if (ldptr->sysglob->d_elapse > 0) {
```
/* compute real-time clock wake-up time */
sec -= (MINSLEEP-1);

(void)currentime(ldptr); /* get the current time from the hardware */

/* Sleep for a maximum of one day */
if (sec >= MAXSEC) {
    sec = ((MAXSEC-1) + ldptr->sec);

    /* start real-time clock alarm interrupt */
    if (rtc_alarm_set(ldptr,sec) != SUCCESS) {
        /* can’t start alarm interrupt */
        return;
    }
} else {
    sec += ldptr->sec;

    /* start real-time clock alarm interrupt */
    if (rtc_alarm_set(ldptr,sec) != SUCCESS) {
        /* can’t start alarm interrupt */
        return;
    }
}

/* turn off ticker(TB and DEC) and PIT module if it is running */
ldptr->regs->tbr_regs.tbscr &~ TBR_ENABLE; /* this will be running because this is our ticker module */
ldptr->regs->pit_regs.piscr &~ PIT_ENABLE; /* need to only shut off if it is running, OPTIONAL */

/* only power down for a maximum of 24 hours */
if (ldptr->sysglob->d_elapse == 0){
    /* Get the current time and save it in static storage */
    (void)currentime(ldptr);

    /* power down for a Maximum of 24 hours */
    sec = ((MAXSEC-1) + ldptr->sec);
    /* wake up after 24 hours */
    if (rtc_alarm_set(ldptr,sec) != SUCCESS){
        /* can’t start alarm interrupt */
        return;
    }
}
/* call the check routine to see if there are any active processes */
if ((active = check(ldptr)) == 0){
    /* Check to see if the status bits are set or not and clear them if they are */
    if (((ldptr->regs->pll_regs.plprcr & PLL_TEXPS) != 0) && ((ldptr->regs->pll_regs.plprcr & PLL_TMIST) != 0)) {
        ldptr->regs->pll_regs.plprcr |= (PLL_TEXPS | PLL_TMIST);
    }
    /* save the mask level and then enable interrupts */
    svirq = irq_save(); irq_enable();
}

/* When using the DFNL, there are several issues which need addressed. One is when you divide by 2 through divide by 32, the cpu acts as normal. When you go to one of the LOW modes, the serial port (keyboard) interrupt will wake you up. When you divide by 64, you cannot wake up from the serial port interrupt (can only assume some hardware problem). When you use the divide by 256, it will not wake up from a serial port interrupt and it will also give a 244 error, which means a read error from the serial port. So there seems to be a problem with trying to use the 64 or 256 values. Example, use only 2, 4, 8, 16 & 32. */

    if (state == MPC821_PWR_CPU_NRMLLOW){
        HIGH(ldptr->regs->pll_regs.plprcr,CSRBIT); /* set the CSRC bit */
    }
    if (state == MPC821_PWR_CPU_DOZEHIGH){
        ldptr->regs->pll_regs.plprcr |= PLL_HIDOZE; /* go to a high doze mode */
    }
    if (state == MPC821_PWR_CPU_DOZELOW){
        ldptr->regs->pll_regs.plprcr |= (PLL_LODOZE | PLL_CSRC); /* go to a low doze mode */
    }
/******* OPTIONAL for the sleep, deep sleep, power down modes
********/

if (state == MPC821_PWR_CPU_SLEEP){
    ldptr->regs->pll_regs.plprcr |= PLL_SLEEP; /* go to a
    sleep mode */
}

if (state == MPC821_PWR_CPU_DEEP_SLEEP){
    ldptr->regs->pll_regs.plprcr |= PLL_DEEP_SLEEP; /* go
to a deep sleep mode */
}

if (state == MPC821_PWR_CPU_PWRDWN){
    ldptr->regs->pll_regs.plprcr |= PLL_PWRDWN; /* Power
down */
}

/********* OPTIONAL for the sleep, deep sleep, power down modes
***********/

} irq_restore(svirq); /* restore mask */
}

} else {
    /* call the check routine to see if there are any active
    processes */
    if ((active = check(ldptr)) == 0){

        /* Check to see if the status bits are set or not and
        clear them if they are */
        if (((ldptr->regs->pll_regs.plprcr & PLL_TEXPS) != 0) &&
            ((ldptr->regs->pll_regs.plprcr & PLL_TMIST) != 0)) {
            ldptr->regs->pll_regs.plprcr |= (PLL_TEXPS |
            PLL_TMIST);
        }

        /* save the mask level and then enable interrupts */
        svirq = irq_save(); irq_enable();
    }

    /* When using the DFNL, there are several issues which need
    addressed.  One is
    when you divide by 2 through divide by 32, the cpu acts as normal.
    When you go
    to one of the LOW modes, the serial port(keyboard) interrupt will
    wake you up.  When
    you divide by 64, you cannot wakeup from the serial port
    interrupt(can only assume some
    hardware problem).  When you use the divide by 256, it will not wake
    up from a serial
port interrupt and it will also give a 244 error, which means a read error from the serial port. So there seems to be a problem with trying to use the 64 or 256 values. Example, use only 2, 4, 8, 16 & 32.

```c
if (state == MPC821_PWR_CPU_NRMLLOW) {
    HIGH(ldptr->regs->pll_regs.plprcr, CSRCBIT); /* set the CSRC bit */
}
if (state == MPC821_PWR_CPU_DOZEHIGH) {
    ldptr->regs->pll_regs.plprcr |= PLL_HIDOZE; /* go to high doze mode */
}
if (state == MPC821_PWR_CPU_DOZelow) {
    ldptr->regs->pll_regs.plprcr |= (PLL_LODOZE | PLL_CSRC); /* go to low doze mode */
}

/*** OPTIONAL for the sleep, deep sleep, power down modes
**********/  
if (state == MPC821_PWR_CPU_SLEEP) {
    ldptr->regs->pll_regs.plprcr |= PLL_SLEEP; /* go to sleep mode */
}
if (state == MPC821_PWR_CPU_DEEP_SLEEP) {
    ldptr->regs->pll_regs.plprcr |= PLL_DEEP_SLEEP; /* go to deep sleep mode */
}
if (state == MPC821_PWR_CPU_PWRDWN) {
    ldptr->regs->pll_regs.plprcr |= PLL_PWRDWN; /* go to power down mode */
}

/*** OPTIONAL for the sleep, deep sleep, power down modes
**********/

  irq_restore(svirq); /* restore mask */
}

/*/ turn off real-time clock alarm interrupt */
(void)rtc_alarm_stop(ldptr);

/* calculate the time we were not at full power */
(void)downtime(ldptr);
/* turn on ticker and PIT(if needed) */
ldptr->regs->tbr_regs.tbscr |= TBR_ENABLE; /* Turn on ticker(DEC) */
ldptr->regs->pit_regs.piscr |= PIT_ENABLE; /* OPTIONAL, depending on if you use it or not */
}
return;
}

/* This section sets up the current time. It also calculates the time we */
/* were down!!!!!! */
void downtime(Localdata ldptr)
{
    /* update system globals */
    u_int32 sc;
    u_int32 ticks;

    /* This next section is to calculate the time we were not at full power */
    /* and to generate the updated time(current time). We get the secs */
    /* from the RTC hardware. This will give me the current time. */
    /* We then take the time we kept in the globals, which is the time we */
    /* stored before we went to a lower power state, and calculate the time */
    /* we were not at full power and update it to the current time. */

    /* get current time */
    sc = ldptr->regs->rtc_regs rtc; /* get the seconds value */

    /* compute # of ticks we were asleep */
ticks = (sc - ldptr->sec);
ticks *= ldptr->sysglob->d_tcksec;

    /* update system global fields */
    ldptr->sysglob->d_ticks += ticks;
    ldptr->sysglob->d_slice = 0;

    /* force OS sleep/alarm recomputation */
    ldptr->sysglob->d_elapse = 1;

    /* reset clock from real-time clock */
(void)_os_setime(sc);
return;
}

/* This function just stores the time before we power down to a state other than */
/* full power!!!!!!!! * /
void currentime(Localdata ldptr)
{
    /* get current time and save the time in static storage */
    ldptr->sec = ldptr->regs->rtc_regs.rtc;
    return;
}

/* check to see if there are any active processes pending */
uint32 check(Localdata ldptr)
{
    Sysglobs sysglob = ldptr->sysglob;
    pr_desc* aproc = FAKEHD(pr_desc*,sysglob->d_activq[0],p_queuen);
    status_reg oldirq;
    oldirq = irq_maskget();      /* mask irqs */
    {
        /* check for empty active process queue (head pointing to itself) */
        if (aproc->p_queuen == aproc) {
            irq_restore(oldirq);
            return(0); /* Go ahead and power down because there are no active process */
        }
    }
    irq_restore(oldirq);   /* unmask irqs */
    return(1); /* Can’t power down because there is an active process */
}

rtclock.c

/*******************************************************************
**********
** ADS821 System Module
**
# include <regs.h>
# include <types.h>
# include <defs.h>

 Bret W. Baker

Michael Long

Chapter 5: SysIF
** Initialize Real-Time Clock Sub-System */

error_code rtc_init(Localdata ldptr) {
    error_code err;
    u_int32 old_rtcsck;
    /* set up RTC isr */
    if ((err = _os_irq(RTC_IRQ_VECTOR,RTC_IRQ_PRIORITY,rtc_isr,ldptr)) != SUCCESS) {
        return(err);
    }
    /* save the old, and set the RTCSCK register */
    old_rtcsck = ldptr->regs->key_regs.rtcsck;
    ldptr->regs->key_regs.rtcsck = 0x55ccaa33;
    /* make sure the RTC is enabled */
    ldptr->regs->rtc_regs.rtcsc |= RTC_ENABLE;
    /* restore the RTCSCK register with the old value */
    /* ldptr->regs->key_regs.rtcsck = old_rtcsck; */
    /* make sure everything is cleared */
    (void)rtc_alarm_stop(ldptr);
    return(SUCCESS);
}

/*
** De-Initialize Real-Time Clock Sub-System */

error_code rtc_term(Localdata ldptr) {
    /* make sure everything is cleared */
    (void)rtc_alarm_stop(ldptr);
    /* remove RTC isr */
    (void)_os_irq(RTC_IRQ_VECTOR,RTC_IRQ_PRIORITY,NULL,ldptr);
    return(SUCCESS);
}

/*
** Set Real-Time Clock Alarm Interrupt
error_code rtc_alarm_set(Localdata ldptr, u_int32 sec)
{
    u_int32 old_rtcsck;
    u_int32 old_rtcalk;

    /* make sure the interrupts are shut off */
    (void)rtc_alarm_stop(ldptr);

    /* save the old, and set the RTCALK register */
    old_rtcalk = ldptr->regs->key_regs.rtcalk;
    ldptr->regs->key_regs.rtcalk = 0x55ccaa33;
    {
        /* set up the alarm register to wake up at this time */
        ldptr->regs->rtc_regs.rtcal = sec; /* set the RTC alarm time
        register */
    }
    /* restore the RTCALK register with the old value */
    /* ldptr->regs->key_regs.rtcalk = old_rtcalk; */

    /* save the old, and set the RTCSCK register */
    old_rtcsck = ldptr->regs->key_regs.rtcsck;
    ldptr->regs->key_regs.rtcsck = 0x55ccaa33;
    {
        /* turn on rtclock alarm interrupts */
        ldptr->regs->rtc_regs.rtcsck |= RTCIRQ_LVL; /* set the RTC IRQ
        level */
        ldptr->regs->rtc_regs.rtcsck |= ALM_ENABLE; /* enable the RTC
        alarm interrupt */
    }
    /* restore the RTCSCK register with the old value */
    /* ldptr->regs->key_regs.rtcsck = old_rtcsck; */

    return(SUCCESS);
}

/*
** Stop Real-Time Clock Alarm Interrupt
*/
error_code rtc_alarm_stop(Localdata ldptr)
{
    status_reg oldsr;
    u_int32 old_rtcsck;
    u_int32 old_rtcalk;

    /* turn off interrupts */
oldsr = irq_maskget();
{
    /* save the old, and set the RTCSCK register */
    old_rtcsck = ldptr->regs->key_regs.rtcsck;
    ldptr->regs->key_regs.rtcsck = 0x55ccaa33;
    {
        /* turn off rtclock interrupts */
        ldptr->regs->rtc_regs.rtcsck &= ~(ALM_ENABLE); /* disable
        alarm interrupt */
        ldptr->regs->rtc_regs.rtcsck &= ~(SEC_ENABLE); /* disable
        seconds interrupt */

        /* clear any pending alarms */
        ldptr->regs->rtc_regs.rtcsck |= (ALM_STATUS | SEC_STATUS);
    }
    /* restore the RTCSCK register with the old value */
    /* ldptr->regs->key_regs.rtcsck = old_rtcsck; */
/* save the old, and set the RTCALK register */
old_rtcalk = ldptr->regs->key_regs.rtcalk;
ldptr->regs->key_regs.rtcalk = 0x55ccaa33;
{
    /* clear alarm register */
    ldptr->regs->rtc_regs.rtcal = 0x00000000; /* set the time
    to 0 */
}
    /* restore the RTCALK register with the old value */
    /* ldptr->regs->key_regs.rtcalk = old_rtcalk;*/
}

irq_restore(oldsr); /* enable interrupts */

return(SUCCESS);
}

/*
**         Real-Time Clock Interrupt Service Routine
*/
error_code rtc_isr(Localdata ldptr)
{
    status_reg oldsr;

    /* is this us? */
    if ((ldptr->regs->rtc_regs.rtcsck & ALM_STATUS) == 0) {
        return(EOS_NOTME); /* this is not a RTC alarm interrupt */
    }
/* woke up from the alarm interrupt */
(void) rtc_alarm_stop(ldptr);

return(SUCCESS);
}
Programming Guidelines

This chapter provides power management programming guidelines for the various OS-9 for 68K and OS-9 sub-systems. These guidelines must be followed to produce a working power managed system.

General compatibility guidelines for development of components in a power managed system involve:

- Boot Code
- Applications
- Device Drivers
- File Managers
**Boot Code**

- The boot-code should leave hardware subsystems (except the CPU) in a powered-down state when control is passed to the kernel. The appropriate high-level device driver (or other hardware-specific module) is then responsible for powering-up the corresponding hardware subsystem. The idea here is to keep idle subsystems powered down whenever possible.

- For consistency, use the "PWR_AWARE" #ifdef macro in sources (when needed) to indicate code sections used only in power aware systems. If the power aware code also works in non-power aware system, then #ifdef is not required.

- PwrMan should be in the M$PreIO list of the init module within systype.d since it should be initialized prior to IO. SysIF should go in the M$Extens list. Following is the section of systype.d that configures the initialization module for OS-9.

- Reference the *OS-9 for 68K Technical Manual* for information on the initialization module and the PreIO and Extens lists. Reference the *OS-9 Technical Manual* for information on the initialization module and the m_preio and m_extens list.

```assembly
CONFIG macro

* specific defs for the Eval Board
MainFram dc.b "Motorola M68328ADS Board",0

ifdef SYSGO

* name of initial module to execute
SysStart dc.b "sysgo",0
* parameters to pass to initial module
SysParam dc.b ",",0

else

* name of initial module to execute
SysStart dc.b "shell",0
* parameters to pass to initial module
SysParam dc.b ",",0

endc

SysDev dc.b "/dd",0 * default disk
ConsolNm dc.b "/term",0 * console terminal pathlist
ClockNm dc.b "tk68328",0 * clock module name

PreIO dc.b "OS9PreIO "
ifdef PWRMAN
```
dc.b "pwrman "
endc

dc.b 0

Extens dc.b "OS9P2 "
dc.b "fpu "
ifdef SYSMBUF
    dc.b "SysMbuf "
endc
ifdef PWRMAN
    dc.b "sysif "
endc
    dc.b 0

- Similarly, for OS-9, PwrMan should be in the PREIOS list of the initialization module for OS-9. SysIF should be in the EXTENSIONS list. Following is an example OS-9 initialization module definition within systype.h.

    /*
    * Init Module variable definitions
    */
#define INSTALNAME "MPC821ADS" /* installation name string */
#define OS9K_REVSTR "OS-9000 for the PowerPC(tm)" /* revision string */

ifndef INIT_DD
    /* name of initial module to execute */
    define SYS_START "shell"

    /* params to pass to initial module */
    define SYS_PARAMS "chd /dd; mbinstall; ex shell"
#endif

 ifndef INIT_DD
     /* initial system disk pathlist */
    define SYS_DEVICE "/dd"
#endif /* INIT_DD */
#ifdef INIT_VCONS
/* name of initial module to execute */
#define SYS_START "shell"

/* params to pass to initial module */
#define SYS_PARAMS "mbinstall; ex shell"
/* #define SYS_PARAMS "mbinstall; undpd -s <>>>/nil* ex shell" */

/* initial system disk pathlist */
#define SYS_DEVICE ""
#endif /* INIT_VCONS */

#ifdef INIT_NODISK
/* name of initial module to execute */
#define SYS_START "shell"

/* params to pass to initial module */
#define SYS_PARAMS "mbinstall; ex shell"
/* #define SYS_PARAMS "mbinstall; undpd -s <>>>/nil* ex shell" */

/* initial system disk pathlist */
#define SYS_DEVICE ""
#endif /* INIT_NODISK */

#ifdef INIT_VCONS
#define CONS_NAME "/vcons" /* console terminal pathlist */
#else
#define CONS_NAME "/term" /* console terminal pathlist */
#endif
#define TICK_NAME "tkdec" /* clock ticker module name */
#define RTC_NAME "rtc821" /* real time clock module name */

/* The order of the following list is important. Please see release notes. */
#define PREIOS "siuirq cpicirq ssm cache pwrman" /* pre-I/O extension module list */
#define IOMAN_NAME "Ioman" /* I/O manager name */
Applications

Guidelines for application development are necessary to ensure integration of power management services.

- Applications should be interrupt driven; polling loops should be avoided. This will cause idle applications to sleep or wait outside the active process queue, providing `PwrPlcy` with an easy indication when the entire system is idle.
- For consistency, use the `PWR_AWARE` #ifdef macro in sources (when needed) to indicate code sections used only in power aware systems. If the power aware code also works in non-power aware system, then the #ifdef macro is not needed.

Device Drivers

- To remain compatible with systems not power managed by the Power Management Subsystem, a device driver must default to a powered-up state during initialization.
- Device drivers should keep hardware subsystems powered down when possible to conserve power (e.g., during deiniz or when the hardware is not in use even though it has been initialized). Care must be taken to share information when powering down a shared device (such as a 2-port 68681 UART).
- Pending operations (I/O or other) must be completed before committing to a power state change (via the `PwrMan` callback routine). For example, an SPI-bus transfer must be completed before a driver can successfully be placed into a low power state.
- For consistency, use the “PWR_AWARE” #ifdef macro in sources (when needed) to indicate code sections used only in power aware systems. If the power aware code also works in non-power aware system, then the #ifdef macro is not needed.

File Managers

- Pending I/O operations must be completed before committing to a power state change request operation. For example, the RBF file manager must install a `PwrMan` power management callback routine to postpone any power state changes if RBF file activities are pending but not completed. This ensures disk data integrity.
- For consistency, use the “PWR_AWARE” #ifdef macro in sources (when needed) to indicate code sections used only in power aware systems. If the power aware
code also works in non-power aware system, then the #ifdef macro is not needed.
## 68328 (OS-9/68000) Hardware Interface

### Table A-1. 68328 State/Characteristics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Characteristics</th>
</tr>
</thead>
</table>
| Normal (default) | Phase lock loop (PLL) enabled  
                      CPU clock enabled (100% duty cycle)  
                      LCD clock enabled  
                      System clock enabled  
                      Interrupts wake up CPU immediately and disable power controller  
                      (interrupt service routines (ISRs) must reset power-controller if lower duty cycle is desired) |
| Gear          | PLL enabled  
                      CPU clock at 3%-97% [3% increment] duty cycle  
                      LCD clock enabled  
                      System clock enabled  
                      Interrupts wake up CPU immediately and disable power controller  
                      (ISRs must reset power controller if lower duty cycle is desired) |
| Doze          | PLL enabled  
                      CPU clock disabled (0% duty cycle)  
                      LCD clock enabled  
                      System clock enabled  
                      Interrupts wake up CPU immediately |
| Sleep         | PLL disabled  
                      CPU clock disabled  
                      LCD clock disabled  
                      System clock disabled  
                      Interrupts wake up CPU within 2ms (maximum PLL synch latency) |
PowerPC
## PPC821 (OS-9/ PPC) Hardware Interface

Table B-1. PPC821 State/Characteristics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal (default)</td>
<td>Phase Lock Loop (PLL) enabled</td>
</tr>
<tr>
<td></td>
<td>CPU clock enabled (100% duty cycle)</td>
</tr>
<tr>
<td></td>
<td>System clock enabled</td>
</tr>
<tr>
<td></td>
<td>LCD enabled</td>
</tr>
<tr>
<td></td>
<td>Interrupts wake up CPU immediately (within 4 maximum system clocks)</td>
</tr>
<tr>
<td>Gear</td>
<td>PLL enabled</td>
</tr>
<tr>
<td></td>
<td>CPU clock at ([\text{Full}/(2^{\text{DivisionFactor}})]%) duty cycle</td>
</tr>
<tr>
<td></td>
<td>System clock enabled</td>
</tr>
<tr>
<td></td>
<td>LCD enabled</td>
</tr>
<tr>
<td></td>
<td>Interrupts wake up CPU immediately (within 4 maximum system clocks)</td>
</tr>
<tr>
<td>Doze</td>
<td>PLL enabled</td>
</tr>
<tr>
<td></td>
<td>CPU clock disabled (0% duty cycle)</td>
</tr>
<tr>
<td></td>
<td>System clock enabled</td>
</tr>
<tr>
<td></td>
<td>LCD enabled</td>
</tr>
<tr>
<td></td>
<td>Interrupts wake up CPU immediately (within 4 maximum system clocks)</td>
</tr>
<tr>
<td>Sleep</td>
<td>PLL enabled</td>
</tr>
<tr>
<td></td>
<td>CPU clock disabled</td>
</tr>
<tr>
<td></td>
<td>System clock disabled</td>
</tr>
<tr>
<td></td>
<td>LCD disabled</td>
</tr>
<tr>
<td></td>
<td>DRAM refresh disabled</td>
</tr>
<tr>
<td></td>
<td>Interrupts wake up CPU immediately (within 4 maximum system clocks)</td>
</tr>
</tbody>
</table>
Table B-1. PPC821 State/Characteristics (Continued)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep Sleep</td>
<td>PLL disabled</td>
</tr>
<tr>
<td></td>
<td>CPU clock disabled</td>
</tr>
<tr>
<td></td>
<td>System clock disabled</td>
</tr>
<tr>
<td></td>
<td>LCD disabled</td>
</tr>
<tr>
<td></td>
<td>DRAM refresh disabled</td>
</tr>
<tr>
<td></td>
<td>Interrupts wake up CPU within 500 PLL input frequency clock (15.6 ms @ 32KHz / 125 us @ 4MHz)</td>
</tr>
<tr>
<td>Power Down</td>
<td>PLL disabled</td>
</tr>
<tr>
<td></td>
<td>CPU clock disabled</td>
</tr>
<tr>
<td></td>
<td>System clock disabled</td>
</tr>
<tr>
<td></td>
<td>LCD disabled</td>
</tr>
<tr>
<td></td>
<td>DRAM refresh disabled</td>
</tr>
<tr>
<td></td>
<td>Reset wakes up CPU within 500 PLL input frequency clock + power supply wake-up</td>
</tr>
</tbody>
</table>
Assembly Interface for OS-9 for 68K

This appendix defines the `F$PwrMan` assembly interface for OS-9 for 68K.
F$PwrMan
Invoke Power Management

ASM Call
OS9 F$PwrMan

Input
(a0).l = Subcode parameter
(a3).l = Private static storage
(a4).l = Process descriptor pointer
(a5).l = Caller's registers
(a6).l = System global data pointer

Output
(cc).w = Carry clear

Error Output
(d1).w = Error code
(cc).w = Carry set

Description
F$PwrMan performs various operations depending on the subcode passed in.
Operations include registering and unregistering a device in the Device Registry Table,
adding and removing an entry from the Power State Table, and changing power states.
Subcode values are defined in the following section of this document.

Sub-Codes
PwrMan sub-codes defined in assembly language follow.

* PwrMan Sub-Codes *

PWRMAN_MIN: equ 0x00  * minimum PwrMan subcode *
PWRMAN_CHECK: equ 0x00  * check if PwrMan is going *
PWRMAN_REGISTER: equ 0x01  * register pwr-routine *
PWRMAN_UNREGISTER: equ 0x02  * unregister pwr-routine *
PWRMAN_STATEADD: equ 0x03  * add power state *
PWRMAN_STATEREMOVE: equ 0x04  * remove power state *
PWRMAN_DEBUG: equ 0x05  * get pwrman globals for testing *
PWRMAN_CHANGE: equ 0x06  * change power states *
PWRMAN_CALLBACK: equ 0x07  * call call-back function directly *
PWRMAN_LINK_PLCY equ 0x08  * link to pwrplcy module *
PWRMAN_UNLINK_PLCY equ 0x09  * unlink from pwrplcy module *
PWRMAN_COPYGLOB equ 0x0a  * get a copy of pwrman globals *
PWRMAN_LINK_EXT equ 0x0b  * link to pwrext module *
PWRMAN_UNLINK_EXT equ 0x0c  * unlink from pwrext module *
Appendix C: Assembly Interface for OS-9 for 68K

Internal Structures

PwrMan internal structures defined in assembly language follow.

* type of device (with respect to power consumption/generation) *

pwr_devtype_consumer: equ 0
pwr_devtype_supplier: equ 1
pwr_devtype_neutral: equ 2
pwr_devtype_other: equ 3

* local power-mode structure for a power-aware device *

org 0

pwr_localmode_level: do.l 1 * Local power mode level *
pwr_localmode_maxload: do.l 1 * max energy load or drain (mW) *
pwr_localmode_minload: do.l 1 * min energy load or drain (mW) *
pwr_localmode_entrytime: do.l 1 * worst-case device entry time (ms) *
pwr_localmode_exittime: do.l 1 * worst-case device exit time (ms) *
pwr_localmode_lpm_name: do.b PWR_IDLEN * local power-mode name *
pwr_localmode_context_pres: do.b 1 * device context preserved flag *
pwr_localmode_rsv1: do.b 11 * reserved *
pwr_localmode_size: equ .

* energy condition structure for a power-aware device *

org 0

pwr_devcond_localpwrmd: do.l 1 * ptr to local power mode array *
pwr_devcond_localpwrmd_num: do.l 1 * # of local power modes *
Parameter Block Definitions

Parameter block definitions for calls into PwrMan follow.

* F_PWRMAN/PWRMAN_CHECK *

f_pwrman_check_pb_version:  do.l 1 * version string buffer ptr *
f_pwrman_check_pb_size:    equ .

* F_PWRMAN/PWRMAN_ADD *

f_pwrman_add_pb_syslevel:  do.l 1 * state level *
f_pwrman_add_pb_priority:  do.l 1 * priority of entry *
f_pwrman_add_pb_devlevel:  do.l 1 * device level *
f_pwrman_add_pb_d:         do.l 1 * additional device param block *
f_pwrman_add_pb_id:        do.l 1 * entry id ptr *
f_pwrman_add_pb_size:      equ .

* F_PWRMAN/PWRMAN_REMOVE *

f_pwrman_remove_pb_syslevel:  do.l 1 * state level *
f_pwrman_remove_pb_priority:  do.l 1 * priority of entry *
f_pwrman_remove_pb_devlevel:  do.l 1 * device level *
f_pwrman_remove_pb_devpb:    do.l 1 * additional device param block *
f_pwrman_remove_pb_id:      do.l 1 * entry id ptr *
f_pwrman_remove_pb_size:    equ .
Appendix C: Assembly Interface for OS-9 for 68K

* F_PWRMAN

F_PWRMAN_COPYLOB *
org f_pwrman_pb_size
f_pwrman_copyglob_pb_pwrglob: do.l 1 * copy of powerman globals *
f_pwrman_copyglob_pb_size: equ .

* F_PWRMAN_DEBUG *
org f_pwrman_pb_size
f_pwrman_debug_pb_pwrglob: do.l 1 * pointer to powerman globals *
f_pwrman_debug_pb_size: equ .

* F_PWRMAN_CHANGE *
org f_pwrman_pb_size
f_pwrman_change_pb_syslevel: do.l 1 * PwrMan’s state level to check *
f_pwrman_change_pb_size: equ .

* F_PWRMAN_CALLBACK *
org f_pwrman_pb_size
f_pwrman_callback_pb_devlevel: do.l 1 * device level *
f_pwrman_callback_pb_devpb: do.l 1 * additional device param block *
f_pwrman_callback_pb_id: do.l 1 * entry id ptr *
f_pwrman_callback_pb_size: equ .

* F_PWRMAN_LINK_PLCY *
org f_pwrman_pb_size
f_pwrman_link_plcy_mname: do.l 1 * *
f_pwrman_link_plcy_pb_size: equ .

* F_PWRMAN_UNLINK_PLCY *
org f_pwrman_pb_size
f_pwrman_unlink_plcy_pb_size: equ .

* F_PWRMAN_LINK_EXT *
org f_pwrman_pb_size
f_pwrman_link_ext_mname: do.l 1 * *
f_pwrman_link_ext_pb_size: equ .

* F_PWRMAN_UNLINK_EXT *
org f_pwrman_pb_size
f_pwrman_unlink_ext_pb_size: equ .

* F_PWRMAN_REG *
or g f_pwrman_pb_size
f_pwrman_reg_pb_func: do.l 1 * call-back function *
f_pwrman_reg_pb_funcparam: do.l 1 * call-back parameter *
f_pwrman_reg_pb_devwrdef: do.l 1 * device power definition *
f_pwrman_reg_pb_id: do.l 1 * entry id ptr *
f_pwrman_reg_pb_size: equ .

* F_PWRMAN_PWRMAN_UNREG *
org f_pwrman_pb_size
f_pwrman_unreg_pb_func do.l 1 * call-back function *
f_pwrman_unreg_pb_funcparam: do.l 1 * call-back parameter *
f_pwrman_unreg_pb_devpwrdef: do.l 1 * device power definition *
f_pwrman_unreg_pb_id: do.l 1 * entry id ptr *
f_pwrman_unreg_pb_size: equ .

* F_PWRMAN_PWRPLCY_INIT *
org f_pwrman_pb_size
f_pwrplcy_init_pb_size: equ .

* F_PWRMAN_PWRPLCY_TERM *
org f_pwrman_pb_size
f_pwrplcy_term_pb_size: equ .

* F_PWRMAN_PWRPLCY_EV_REQUEST *
org f_pwrman_pb_size
f_pwrplcy_ev_request_pb_pwrevent: do.l 1 * Ptr to power-event structure *
f_pwrplcy_ev_request_pb_size: equ .

* F_PWRMAN_PWRPLCY_EV_NOTIFY *
org f_pwrman_pb_size
f_pwrplcy_ev_notify_pb_pwrevent: do.l 1 * Ptr to pwrevent structure *
f_pwrplcy_ev_notify_pb_size: equ .

* F_PWRMAN_PWREXT_INIT *
org f_pwrman_pb_size
f_pwrext_init_pb_size: equ .

* F_PWRMAN_PWREXT_TERM *
org f_pwrman_pb_size
f_pwrext_term_pb_size: equ .
### Table 6-1. SH7709 State/Characteristics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Characteristics</th>
</tr>
</thead>
</table>
| Normal (default) | Clock Pulse Generator (CPG) is operating  
                      CPU, bus and peripheral clocks are enabled  
                      On-Chip supporting modules are operating  
                      External memory refreshing is on  
                      Graphics and serial (SCI, SCIF) devices are on  
                      Interrupts are served immediately |
| Sleep            | Clock Pulse Generator (CPG) is operating  
                      CPU is halted (registers held)  
                      Bus and peripheral clocks are enabled  
                      On-Chip supporting modules are operating  
                      External memory refreshing is on  
                      Graphics device is off  
                      Serial (SCI, SCIF) devices are on  
                      System wakes up by interrupt or reset |
| Standby          | Clock Pulse Generator (CPG) is halted  
                      CPU is halted (registers held)  
                      Bus and peripheral clocks are halted  
                      Most on-chip modules (except RTC) are halted  
                      External memory is in self-refresh mode and needs external clock source to hold content  
                      Graphics device are off  
                      Serial (SCI, SCIF) devices are off  
                      System wakes up by NMI, IRL or RTC interrupt or by reset and executes some initialization code in ROM to put external memory in auto-refresh mode before serving interrupts |
Index

Symbols
_os_pwr_add() 14, 22
_os_pwr_callback() 22
_os_pwr_change() 22
    customizing function example 57
_os_pwr_check() 22
_os_pwr_copyglob() 22, 29
_os_pwr_debug() 22, 28
_os_pwr_ev_notify() 17, 22
_os_pwr_ev_request() 17, 18, 22
_os_pwr_link_ext() 22, 23
_os_pwr_link_plcy() 22, 23, 37
_os_pwr_reg() 22, 36, 38
_os_pwr_remove() 22
_os_pwr_unlink() 22
_os_pwr_unlink_ext() 22, 23
_os_pwr_unlink_plcy() 22, 23, 33
_os_pwr_unreg() 22, 34

A
application
    power aware 10
    power unaware 10
assembly programming
    FSPwrMan 97
    structures 98

B
B_NoStop 44

C
callback function
    call 25
    devlevel 14, 15, 24, 25, 34, 35, 38, 40
    devpb 15, 24, 25, 34, 35, 38
    func 13, 15, 34, 38
    priority 40
context_pres 17
    current power state 26
currlevel 26

debug
    _os_pwr_debug() 29
dev_cond 18
dev_id 18

dev_type
    energy 16
device
    context 17
display 39
drain 18
driver 6
energy 18
    devpwrdef 34, 38
load 16
    structure 16
identifier 18
power
    aware 39
    condition 18
    max_load 17
    min_load 17
mode
    structure 17
registry
    table
        id 13
registry table 12
display 39
    entry
        add 34
devpwrdef 34, 38
delete 38
        free entry pool pointer 19
        id 12, 14, 24, 25, 34, 35, 38, 40
os_pwr_reg() 16
os_pwr_unreg() 16
    pointer
pwr_devcond structure 16
device registry table 12
display 39
    entry
        add 34
delete 38
        free pool pointer 19
        id 12, 13, 14, 24, 25, 34, 35, 38, 40
os_pwr_reg() 16
os_pwr_unreg() 16
    pointer 19
pwr_devcond structure 16
devlevel 24, 25, 34, 35, 38, 40
devpb 15, 24, 25, 34, 35, 38
devpwrdef
  pointer
device energy 13, 34, 38
dfree 19
dhead 19
  device registry table
  pointer 19

e
energy
device
  level 16
  load 16
  type 16
entry point
  pwrplcy
  pointer 19
entrytime 17
error
  ret_on_err 15, 26
  ret_on_error 20
event
  generate 16
  power
  structure 17
example
  pwrplcy 43
exitime 17

F
F$PwrMan
  assembly programming 97
file manager 6
func
  callback function
  state change 13, 15, 34, 38
funcparam
  parameter block 13, 15, 34, 38
function binding 20

G
globals
  pointer 29
  pwrman 11
guidelines
  programming 85

H
header files
  custom 63

I
id 13, 14, 24, 25, 34, 35, 38, 40
  module 32, 33
idle
  function
    B_NoStop 44
    interrupt 44
    STOP/LPSTOP 44
  loop 43
idle loop
customize
  OS-9 43, 44
  OS-9000 43, 44
  interrupts 44
initialize
  power state 43
  pwrext_init 57
  pwrplcy_init 57
  sysif 63
interrupt
  idle function 44
  interrupts
    idle loop 44

L
level 17
library
  assembly language 20
load 17
  maximum 17
  minimum 17
loc_pwr_mode 17
  context_pres 17
  entrytime 17
  exitime 17
  level 17
  lpm_name 17
  max_load 17
  min_load 17
local
  memory
    pwrext
      pointer 20
    pwrplcy
      pointer 19
    power mode 18
  local power mode 18
  lpm 16
  lpm_name 17

M
macro 20
Index

A
  max_load 17
  module 17
  module id 32, 33

B
  min_load 17

C
  microprocessor interface
    sysif 63

D
  new_drain 18
  new_energy 18
  new_lpm 18
  newlink pwrext 57
  num_lpm 16

E
  old_drain 18
  old_energy 18
  old_lpm 18
  OS-9
    idle loop
      customize 43, 44
  OS-9000
    idle loop
      customize 43, 44

F
  phead
    power state table
      pointer 19
  pointer
    globals 29
    power
      state 26
  power
    aware 6
      application 10
      device driver 8
    definition 15
    device
      maximum energy load 17
      minimum energy load 17
    down 17
    event 16, 17, 20
    extension module 8
    management subsystem
      components 8
      interaction in 9
    policy module 8
    state 26
    change 30, 31, 43
    initialize 43
    level 24, 26, 35, 40
    pointer 26
    state table 12
      add
        entry 24
        display 39
        entry
          add 39
          delete 35
          free pool pointer 19
          remove 39
        free entry pool pointer 19
        id 12, 24, 25, 34, 35, 38, 40
        pointer
          phead 19
        structure 15
        unaware 6
          application 10
          device driver 10
          up 17
        power management module 8
        power policy module 9
        PowerPC
        PPC821
          hardware interface 94
        PPC821
          hardware interface 94
        pres_drain 16
        pres_energy 16
        prespwrlv 16
        print
          version 39
          priority 40
          programming
            guidelines 85
        pwr_devcond 16
          dev_type 16
          lpm 16
          num_lpm 16
          pres_drain 16
          pres_energy 16
          prespwrlv 16
          structure 34, 38
        pwr_event 17, 20, 30, 31
        _os_pwr_ev_notify() 17
        _os_pwr_ev_request() 17
        dev_cond 18
        dev_id 18
        generate 16
        local power mode 18
        new_drain 18
new_energy 18
new_lpm 18
old_drain 18
old_energy 18
old_lpm 18
pwrevent 30, 31
pwrext 8
initialize 57
pwrman_change 57
pwrext_func
entry point
pwrext
pointer 20
pwrext_mem 20
pwrglob 29
pwrman 8
event
generate 16
globals
structure 11
install 27
overriding 57
table
device registry 12
power state 12
pwrman_change 57
pwrman_globals 15
pwrplcy 8, 9
definition 43
example 43
function prototype 43
initialize 57
pwrplcy.c 48
type 43
pwrplcy.c
pwrplcy 48
pwrplcy_func 19
pwrplcy_mem 19

STOP/LPSTOP 44
structure 20
device
energy 16
power mode 17
loc_pwr_mode 17
power
event 17
pwr_devcond 34, 38
pwr_event 17
structures
assembly programming 98
support mechanism
device driver 6
file manager 6
system module 6
sysglob 19
SysIf 9
sysif 8, 9, 63
initialize 63
syslevel 24, 26, 35, 39, 40
system
global
data pointer 44
pointer
sysglob 19
interface module 8, 9
level 39
module 6

T
type
definition 20

V
version 27
pwrman 39

R
ret_on_err 26
ret_on_error
error 20

S
SH-3
hardware interface 104
SH7709
hardware interface 104
state
change 30
func 13, 15, 34, 38