man pages section 9: DDI and DKI Overview

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806-0637-10
February 2000

Copyright 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227–14(g)(2)(6/87) and FAR 52.227–19(6/87), or DFAR 252.227–7015(b)(6/95) and DFAR 227.7202–3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, Californie 94303-4900 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Contents

Preface

Both novice users and those familar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 6 contains available games and demos.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.

- Section 9 provides reference information needed to write device drivers in the kernel environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man`(1) for more information about man pages in general.

NAME	This section gives the names of the commands or functions documented, followed by a brief description of what they do.
SYNOPSIS	This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required. The following special characters are used in this section: [ ] Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified. . . . Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "`filename` . . ." . | Separator. Only one of the arguments separated by this character can be specified at a time. { } Braces. The options and/or arguments enclosed within braces are

	interdependent, such that everything enclosed must be treated as a unit.
PROTOCOL	This section occurs only in subsection 3R to indicate the protocol description file.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.
IOCTL	This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the `ioctl`(2) system call is called `ioctl` and generates its own heading. `ioctl` calls for a specific device are listed alphabetically (on the man page for that specific device). `ioctl` calls are used for a particular class of devices all of which have an `io` ending, such as `mtio`(7I).
OPTIONS	This secton lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they

	failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.
USAGE	This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.
FILES	This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
ATTRIBUTES	This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes`(5) for more information.

SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

CHAPTER

Device Driver Interfaces

10

NAME

Intro – introduction to device driver interfaces

DESCRIPTION

Section 9 provides reference information needed to write device drivers for Solaris 2 and Solaris 7. It describes the interfaces provided by the Device Driver Interface Driver-Kernel Interface (DDI/DKI).

Porting

Software is usually considered portable if it can be adapted to run in a different environment more cheaply than it can be rewritten. The new environment may include a different processor, operating system, and even the language in which the program is written, if a language translator is available. Likewise the new environment might include multiple processors. More often, however, software is ported between environments that share an operating system, processor, and source language. The source code is modified to accommodate the differences in compilers or processors or releases of the operating system.

In the past, device drivers did not port easily for one or more of the following reasons:

- To enhance functionality, members had been added to kernel data structures accessed by drivers, or the sizes of existing members had been redefined.
- The calling or return syntax of kernel functions had changed.
- Driver developers did not use existing kernel functions where available, or relied on undocumented side effects that were not maintained in the next release.
- Architecture-specific code had been scattered throughout the driver when it could have been isolated.

Operating systems are periodically reissued to customers as a way to improve performance, fix bugs, and add new features. This is probably the most common threat to compatibility encountered by developers responsible for maintaining software. Another common problem is upgrading hardware. As new hardware is developed, customers occasionally decide to upgrade to faster, more capable computers of the same family. Although they may run the same operating system as those being replaced, architecture-specific code may prevent the software from porting.

Scope of Interfaces

Although application programs have all of the porting problems mentioned, developers attempting to port device drivers have special challenges. Before describing the DDI/DKI, it is necessary to understand the position of device drivers in operating systems.

Device drivers are kernel modules that control data transferred to and received from peripheral devices but are developed independently from the rest of the kernel. If the goal of achieving complete freedom in modifying the kernel is to be reconciled with the goal of binary compatibility with existing drivers, the interaction between drivers and the kernel must be rigorously regulated. This

driver / kernel service interface is the most important of the three distinguishable interfaces for a driver, summarized as follows:

- Driver–Kernel. I / O System calls result in calls to driver entry point routines. These make up the kernel-to-driver part of the service interface, described in Section 9E. Drivers may call any of the functions described in Section 9F. These are the driver-to-kernel part of the interface.
- Driver–Hardware. All drivers (except software drivers) must include code for interrupt handling, and may also perform direct memory access (DMA). These and other hardware-specific interactions make up the driver / hardware interface.
- Driver–Boot / Configuration Software. The interaction between the driver and the boot and configuration software is the third interface affecting drivers.

Scope of the DDI/DKI

The primary goal of the DDI / DKI is to facilitate both source and binary portability across successive releases of the operating systems on a particular machine. In addition, it promotes source portability across implementations of UNIX on different machines, and applies only to implementations based on System V Release 4. The DDI / DKI consists of several sections:

- DDI / DKI Architecture Independent - These interfaces are supported on all implementations of System V Release 4.
- DKI-only - These interfaces are part of System V Release 4, and may not be supported in future releases of System V. There are only two interfaces in this class, `segmap`(9E) and `hat_getkpfnum`(9F)
- Solaris DDI - These interfaces specific to Solaris.
- Solaris SPARC specific DDI - These interfaces are specific to the SPARC processor, and may not be available on other processors supported by Solaris.
- Solaris IA specific DDI - These interfaces are specific to the IA processor, and may not be available on other processors supported by Solaris.

To achieve the goal of source and binary compatibility, the functions, routines, and structures specified in the DDI / DKI must be used according to these rules.

- Drivers cannot access system state structures (for example, `u` and `sysinfo`) directly.
- For structures external to the driver that may be accessed directly, only the utility functions provided in Section 9F should be used. More generally, these functions should be used wherever possible.
- The headers `<sys/ddi.h>` and `<sys/sunddi.h>` must be the last header files included by the driver.

Audience

Section 9 is for software engineers responsible for creating, modifying, or maintaining drivers that run on this operating system and beyond. It assumes that the reader is familiar with system internals and the C Programming Language.

PCMCIA Standard

The *PC Card 95 Standard* is listed under the SEE ALSO heading in some Section 9 reference pages. This refers to documentation published by the Personal Computer Memory Card International Association (PCMCIA) and the Japan Electronic Industry Development Association (JEIDA).

How to Use Section 9

Section 9 is divided into three subsections:

9E Driver Entry Points – contains reference pages for all driver entry point routines.

9F Kernel Functions – contains reference pages for all driver support routines.

9S Data Structures – contains reference pages for driver-related structures.

SEE ALSO

`intro`(9E), `intro`(9F), `intro`(9S)

NOTES

SunSoft's implementation of the DDI/DKI was designed to provide binary compatibility for third-party device drivers across currently supported hardware platforms across minor releases of the operating system.

However, unforeseen technical issues may force changes to the binary interface of the DDI/DKI. We cannot therefore promise or in any way assure that DDI/DKI-compliant device drivers will continue to operate correctly on future releases.

NAME

pm – Power Management properties

DESCRIPTION

The `pm-hardware-state` property may be used to influence the behavior of the Power Management framework. Its syntax and interpretation is described below.

Note that this property is only interpreted by the system immediately after the device has successfully attached. Changes in the property made by the driver after the driver has attached will not be recognized.

`pm-hardware-state` is a string-valued property. The existence of the `pm-hardware-state` property indicates that a device needs special handling by the Power Management framework with regard to its hardware state.

If the value of this property is `needs-suspend-resume`, the device has a hardware state that cannot be deduced by the framework. The framework definition of a device with hardware state is one with a `reg` property. Some drivers, such as SCSI disk and tape drivers, have no `reg` property but manage devices with "remote" hardware. Such a device must have a `pm-hardware-state` property with a value of `needs-suspend-resume` for the system to identify it as needing a call to its `detach`(9E) entry point with command `DDI_SUSPEND` when system is suspended, and a call to `attach`(9E) with command `DDI_RESUME` when system is resumed. For devices using original Power Management interfaces (which are now obsolete) `detach`(9E) is also called with `DDI_PM_SUSPEND` before power is removed from the device, and `attach`(9E) is called with `DDI_PM_RESUME` after power is restored.

A value of `no-suspend-resume` indicates that, in spite of the existence of a `reg` property, a device has no hardware state that needs saving and restoring. A device exporting this property will not have its `detach( )` entry point called with command `DDI_SUSPEND` when system is suspended, nor will its `attach( )` entry point be called with command `DDI_RESUME` when system is resumed. For devices using the original (and now obsolete) Power Management interfaces, `detach`(9E) will not be called with `DDI_PM_SUSPEND` command before power is removed from the device, nor `attach`(9E) will be called with `DDI_PM_RESUME` command after power is restored to the device.

A value of `parental-suspend-resume` indicates that the device does not implement the `detach`(9E) `DDI_SUSPEND` semantics, nor the `attach( )` `DDI_RESUME` semantics, but that a call should be made up the device tree by the framework to effect the saving and / or restoring of hardware state for this device. For devices using original Power Management interfaces (which are now obsolete), it also indicates that the device does not implement the `detach`(9E) `DDI_PM_SUSPEND` semantics, nor the `attach`(9E) `DDI_PM_RESUME` sematics, but that a call should be made up the device tree by the framework to effect the saving and / or restoring the hardware state for this device.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface stability	Evolving

SEE ALSO

`power.conf`(4), `pm`(7D), `pm-components`(9), `attach`(9E), `detach`(9E), `pm_busy_component`(9F), `pm_create_components`(9F), `pm_destroy_components`(9F), `pm_idle_component`(9F)

Writing Device Drivers

NAME

pm-components – Power Management device property

DESCRIPTION

A device is power manageable if the power consumption of the device can be reduced when it is idle. In general, a power manageable device consists of a number of power manageable hardware units called components. Each component is separately controllable and has its own set of power parameters.

An example of a one-component power manageable device is a disk whose spindle motor can be stopped to save power when the disk is idle. An example of a two-component power manageable device is a frame buffer card with a connected monitor. The frame buffer electronics (with power that can be reduced when not in use) comprises the first component. The second component is the monitor, which can enter in a lower power mode when not in use. The combination of frame buffer electronics and monitor is considered as one device by the system.

In the Power Management framework, all components are considered equal and completely independent of each other. If this is not true for a particular device, the device driver must ensure that undesirable state combinations do not occur.

The `pm-components` property describes the Power Management model of a device driver to the Power Management framework. It lists each power manageable component by name and lists the power level supported by each component by numerical value and name. Its syntax and interpretation is described below.

This property is only interpreted by the system immediately after the device has successfully attached, or upon the first call into Power Management framework, whichever comes first. Changes in the property made by the driver after the property has been interpreted will not be recognized.

`pm-components` is a string array property. The existence of the `pm-components` property indicates that a device implements power manageable components and describes the Power Management model implemented by the device driver. The existence of `pm-components` also indicates to the framework that device is ready for Power Management if automatic device Power Management is enabled. See `power.conf`(4).

The `pm-component` property syntax is:

```
pm-components="NAME=component name", "numeric power level=power level name",
  "numeric power level=power level name"
  [, "numeric power level=power level name" ...]
  [, "NAME=component name", "numeric power level=power level name",
  "numeric power level=power level name"
  [, "numeric power level=power level name"...]...];
```

The start of each new component is represented by a string consisting of `NAME=` followed by the name of the component. This should be a short name that a user

would recognize, such as "Monitor" or "Spindle Motor." The succeeding elements in the string array must be strings consisting of the numeric value (can be decimal or 0x <hexadecimal number>) of a power level the component supports, followed by an equal sign followed by a short descriptive name for that power level. Again, the names should be descriptive, such as "On," "Off," "Suspend," "Standby," etc. The next component continues the array in the same manner, with a string that starts out `NAME=`, specifying the beginning of a new component (and its name), followed by specifications of the power levels the component supports.

The components must be listed in increasing order according to the component number as interpreted by the driver's `power`(9E) routine. (Components are numbered sequentially from 0). The power levels must be listed in increasing order of power consumption. Each component must support at least two power levels, or there is no possiblity of power level transitions. If a power level value of 0 is used, it must be the first one listed for that component. A power level value of 0 has a special meaning (off) to the Power Management framework.

EXAMPLES

An example of a `pm-components` entry from the `.conf` file of a driver which implements a single power managed component consisting of a disk spindle motor is shown below. This is component 0 and it supports 2 power level, which represent spindle stopped or full speed.

```
pm-components="NAME=Spindle Motor", "0=Stopped", "1=Full Speed";
...
```

Below is an example of how the above entry would be implemented in the `attach`(9E) function of the driver.

```
static char *pmcomps[] = {
  "NAME=Spindle Motor",
  "0=Stopped",
  "1=Full Speed"
};

...

xxattach(dev_info_t *dip, ddi_attach_cmd_t cmd)
{
...
  if (ddi_prop_update_string_array(DDI_DEV_T_NONE, dip, "pm-components",
     &pmcomp[0], sizeof (pmcomps) / sizeof (char *)) != DDI_PROP_SUCCESS)
  goto failed;
}
```

Below is an example for a frame buffer which implements two components. Component 0 is the frame buffer electronics which supports four different power levels. Component 1 represents the state of Power Management of the attached monitor.

```
pm-components="NAME=Frame Buffer", "0=Off"
  "1=Suspend", "2=Standby", "3=On",
```

```
  "NAME=Monitor", "0=Off", "1=Suspend", "2=Standby,"
  "3=On;
```

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface stability	Evolving

SEE ALSO

`power.conf`(4), `pm`(7D), `attach`(9E), `detach`(9E), `ddi_prop_update_string_array`(9F) `pm_busy_component`(9F), `pm_create_components`(9F), `pm_destroy_components`(9F), `pm_idle_component`(9F)

Writing Device Drivers

Index