NAME

Intro, intro – introduction to device driver entry points

DESCRIPTION

Section 9E describes the entry-point routines a developer may include in a device driver. These are called entry-point because they provide the calling and return syntax from the kernel into the driver. Entry-points are called, for instance, in response to system calls, when the driver is loaded, or in response to STREAMS events.

Kernel functions usable by the driver are described in section 9F.

In this section, reference pages contain the following headings:

- **NAME** describes the routine's purpose.
- **SYNOPSIS** summarizes the routine's calling and return syntax.
- **INTERFACE LEVEL** describes any architecture dependencies. It also indicates whether the use of the entry point is required, optional, or discouraged.
- **ARGUMENTS** describes each of the routine's arguments.
- **DESCRIPTION** provides general information about the routine.
- **RETURN VALUES** describes each of the routine's return values.
- **SEE ALSO** gives sources for further information.

Overview of Driver Entry-Point Routines and Naming Conventions

By convention, a prefix string is added to the driver routine names. For a driver with the prefix *prefix*, the driver code may contain routines named *prefix*open, *prefix*close, *prefix*read, *prefix*write, and so forth. also use the same prefix.

All routines and data should be declared as **static**.

Every driver MUST include **<sys/ddi.h>** and **<sys/sunddi.h>**, in that order, and after all other include files.

The following table summarizes the STREAMS driver entry points described in this section.

Routine	*Type*
put	DDI/DKI
srv	DDI/DKI

The following table summarizes the driver entry points described in this section.

Routine	*Type*
_fini	Solaris DDI
_info	Solaris DDI
_init	Solaris DDI
aread	Solaris DDI
attach	Solaris DDI
awrite	Solaris DDI
chpoll	DDI/DKI
close	DDI/DKI
detach	Solaris DDI
dump	Solaris DDI

getinfo	Solaris DDI
identify	Solaris DDI
ioctl	DDI/DKI
ks_update	Solaris DDI
mapdev_access	Solaris DDI
mapdev_dup	Solaris DDI
mapdev_free	Solaris DDI
mmap	DKI only
open	DDI/DKI
print	DDI/DKI
probe	Solaris DDI
prop_op	Solaris DDI
read	DDI/DKI
segmap	DKI only
strategy	DDI/DKI
tran_abort	Solaris DDI
tran_destroy_pkt	Solaris DDI
tran_dmafree	Solaris DDI
tran_getcap	Solaris DDI
tran_init_pkt	Solaris DDI
tran_reset	Solaris DDI
tran_reset_notify	Solaris DDI
tran_setcap	Solaris DDI
tran_start	Solaris DDI
tran_sync_pkt	Solaris DDI
tran_tgt_free	Solaris DDI
tran_tgt_init	Solaris DDI
tran_tgt_probe	Solaris DDI
write	DDI/DKI

The table below lists the error codes that should be returned by a driver routine when an error is encountered. It lists the error values in alphabetic order. All the error values are defined in **<sys/errno.h>**. In the driver **open**(9E), **close**(9E), **ioctl**(9E), **read**(9E), and **write**(9E) routines, errors are passed back to the user by returning the value. In the driver **strategy**(9E) routine, errors are passed back to the user by setting the **b_error** member of the **buf**(9S) structure to the error code. For STREAMS **ioctl** routines, errors should be sent upstream in an **M_IOCNAK** message. For STREAMS **read** and **write** routines, errors should be sent upstream in an **M_ERROR** message. The driver **print** routine should not return an error code, as the function that it calls, **cmn_err**(9F), is declared as **void** (no error is returned).

Error Value	Error Description	Use in these Driver Routines (9E)
EAGAIN	Kernel resources, such as the **buf** structure or cache memory, are not available at this time (device may be busy, or the system resource is not available).	**open, ioctl, read, write, strategy**
EFAULT	An invalid address has been passed as an argument; memory addressing error.	**open, close, ioctl, read, write, strategy**
EINTR	Sleep interrupted by signal.	**open, close, ioctl, read, write, strategy**
EINVAL	An invalid argument was passed to the routine.	**open, ioctl, read, write, strategy**
EIO	A device error occurred; an error condition was detected in a device status register (the I/O request was valid, but an error occurred on the device).	**open, close, ioctl, read, write, strategy**
ENXIO	An attempt was made to access a device or subdevice that does not exist (one that is not configured); an attempt was made to perform an invalid I/O operation; an incorrect minor number was specified.	**open, close, ioctl, read, write, strategy**
EPERM	A process attempting an operation did not have required permission.	**open, ioctl, read, write, close**
EROFS	An attempt was made to open for writing a read-only device.	**open**

The table below cross references error values to the driver routines from which the error values can be returned.

open	**close**	**ioctl**	**read, write, and strategy**
EAGAIN	**EFAULT**	**EAGAIN**	**EAGAIN**
EFAULT	**EINTR**	**EFAULT**	**EFAULT**
EINTR	**EIO**	**EINTR**	**EINTR**
EINVAL	**ENXIO**	**EINVAL**	**EINVAL**
EIO		**EIO**	**EIO**
ENXIO		**ENXIO**	**ENXIO**
EPERM		**EPERM**	
EROFS			

Name	**Description**
aread(9E)	asynchronous read from a device
attach(9E)	attach a device to the system
awrite(9E)	asynchronous write to a device
chpoll(9E)	poll entry point for a non-STREAMS character driver
close(9E)	relinquish access to a device
detach(9E)	detach a device
dump(9E)	dump memory to device during system failure
_fini(9E)	loadable module configuration entry points
getinfo(9E)	get device driver information
identify(9E)	claim to drive a device
_info(9E)	See **_fini**(9E)
_init(9E)	See **_fini**(9E)
ioctl(9E)	control a character device
ks_update(9E)	dynamically update kstats
mapdev_access(9E)	device mapping access entry point
mapdev_dup(9E)	device mapping duplication entry point
mapdev_free(9E)	device mapping free entry point
mmap(9E)	check virtual mapping for memory mapped device
open(9E)	gain access to a device
print(9E)	display a driver message on system console
probe(9E)	determine if a non-self-identifying device is present
prop_op(9E)	report driver property information
put(9E)	receive messages from the preceding queue
read(9E)	read data from a device
segmap(9E)	map device memory into user space
srv(9E)	service queued messages
strategy(9E)	perform block I/O
tran_abort(9E)	abort a SCSI command
tran_destroy_pkt(9E)	See **tran_init_pkt**(9E)
tran_dmafree(9E)	SCSI HBA DMA deallocation entry point
tran_getcap(9E)	get/set SCSI transport capability
tran_init_pkt(9E)	SCSI HBA packet preparation and deallocation
tran_reset(9E)	reset a SCSI bus or target

tran_reset_notify(9E)	request to notify SCSI target of bus reset
tran_setcap(9E)	See **tran_getcap**(9E)
tran_start(9E)	request to transport a SCSI command
tran_sync_pkt(9E)	SCSI HBA memory synchronization entry point
tran_tgt_free(9E)	request to free HBA resources allocated on behalf of a target
tran_tgt_init(9E)	request to initialize HBA resources on behalf of a particular target
tran_tgt_probe(9E)	request to probe SCSI bus for a particular target
write(9E)	write data to a device