```
**********************************************************
   13502 Sat Feb 15 09:54:06 2020
new/usr/src/man/man3c/cond_init.3c
12309 errors in section 9e of the manual
**********************************************************
.\"
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for
.\" permission to reproduce portions of its copyrighted documentation.
.\" Original documentation from The Open Group can be obtained online at
.\" http://www.opengroup.org/bookstore/.
.\"
.\" The Institute of Electrical and Electronics Engineers and The Open
.\" Group, have given us permission to reprint portions of their
.\" documentation.
.\"
.\" In the following statement, the phrase ‘‘this text’’ refers to portions
.\" of the system documentation.
.\"
.\" Portions of this text are reprinted and reproduced in electronic form
.\" in the SunOS Reference Manual, from IEEE Std 1003.1, 2004 Edition,
.\" Standard for Information Technology -- Portable Operating System
.\" Interface (POSIX), The Open Group Base Specifications Issue 6,
.\" Copyright (C) 2001-2004 by the Institute of Electrical and Electronics
.\" Engineers, Inc and The Open Group.  In the event of any discrepancy
.\" between these versions and the original IEEE and The Open Group
.\" Standard, the original IEEE and The Open Group Standard is the referee
.\" document.  The original Standard can be obtained online at
.\" http://www.opengroup.org/unix/online.html.
.\"
.\" This notice shall appear on any product containing this material.
.\"
.\" The contents of this file are subject to the terms of the
.\" Common Development and Distribution License (the "License").
.\" You may not use this file except in compliance with the License.
.\"
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
.\" or http://www.opensolaris.org/os/licensing.
.\" See the License for the specific language governing permissions
.\" and limitations under the License.
.\"
.\" When distributing Covered Code, include this CDDL HEADER in each
.\" file and include the License file at usr/src/OPENSOLARIS.LICENSE.
.\" If applicable, add the following below this CDDL HEADER, with the
.\" fields enclosed by brackets "[]" replaced with your own identifying
.\" information: Portions Copyright [yyyy] [name of copyright owner]
.\"
.\"
.\" Portions Copyright (c) 1995 IEEE.  All Rights Reserved.
.\" Copyright (c) 2001, The IEEE and The Open Group.  All Rights Reserved.
.\" Copyright (c) 2007, Sun Microsystems, Inc.  All Rights Reserved.
.\"
.TH COND_INIT 3C "February 15, 2020"
.TH COND_INIT 3C "Jun 5, 2007"
.SH NAME
cond_init, cond_wait, cond_timedwait, cond_reltimedwait, cond_signal,
cond_broadcast, cond_destroy \- condition variables
.SH SYNOPSIS
.LP
.nf
cc -mt [ \fIflag\fR... ] \fIfile\fR... [ \fIlibrary\fR... ]
#include <thread.h>
#include <synch.h>

\fBint\fR \fBcond_init\fR(\fBcond_t *\fR\fIcvp\fR, \fBint\fR \fItype\fR, \fBvoid
.fi
```



```
.LP
.nf
\fBint\fR \fBcond_wait\fR(\fBcond_t *\fR\fIcvp\fR, \fBmutex_t *\fR\fImp\fR);
.fi

.LP
.nf
\fBint\fR \fBcond_timedwait\fR(\fBcond_t *\fR\fIcvp\fR, \fBmutex_t *\fR\fImp\fR,
     \fBtimestruc_t *\fR\fIabstime\fR);
.fi

.LP
.nf
\fBint\fR \fBcond_reltimedwait\fR(\fBcond_t *\fR\fIcvp\fR, \fBmutex_t *\fR\fImp\
     \fBtimestruc_t *\fR\fIreltime\fR);
.fi

.LP
.nf
\fBint\fR \fBcond_signal\fR(\fBcond_t *\fR\fIcvp\fR);
.fi

.LP
.nf
\fBint\fR \fBcond_broadcast\fR(\fBcond_t *\fR\fIcvp\fR);
.fi

.LP
.nf
\fBint\fR \fBcond_destroy\fR(\fBcond_t *\fR\fIcvp\fR);
.fi

.SH DESCRIPTION
.SS "Initialize"
.sp
.LP
Condition variables and mutexes should be global. Condition variables that are
allocated in writable memory can synchronize threads among processes if they
are shared by the cooperating processes (see \fBmmap\fR(2)) and are initialized
for this purpose.
.sp
.LP
The scope of a condition variable is either intra-process or inter-process.
This is dependent upon whether the argument is passed implicitly or explicitly
to the initialization of that condition variable. A condition variable does not
need to be explicitly initialized. A condition variable is initialized with all
zeros, by default, and its scope is set to within the calling process. For
inter-process synchronization, a condition variable must be initialized once,
and only once, before use.
.sp
.LP
A condition variable must not be simultaneously initialized by multiple threads
or re-initialized while in use by other threads.
.sp
.LP
Attributes of condition variables can be set to the default or customized at
initialization.
.sp
.LP
The \fBcond_init()\fR function initializes the condition variable pointed to by
\fIcvp\fR. A condition variable can have several different types of behavior,
specified by \fItype\fR. No current type uses \fIarg\fR although a future type
may specify additional behavior parameters with \fIarg\fR. The \fItype\fR
argument c take one of the following values:
.sp
.ne 2
```

```
.na
\fB\fBUSYNC_THREAD\fR\fR
.ad
.RS 17n
The condition variable can synchronize threads only in this process. This is
the default.
.RE

.sp
.ne 2
.na
\fB\fBUSYNC_PROCESS\fR\fR
.ad
.RS 17n
The condition variable can synchronize threads in this process and other
processes. Only one process should initialize the condition variable. The
object initialized with this attribute must be allocated in memory shared
between processes, either in System V shared memory (see \fBshmop\fR(2)) or in
memory mapped to a file (see \fBmmap\fR(2)). It is illegal to initialize the
object this way and to not allocate it in such shared memory.
.RE

.sp
.LP
Initializing condition variables can also be accomplished by allocating in
zeroed memory, in which case, a \fItype\fR of \fBUSYNC_THREAD\fR is assumed.
.sp
.LP
If default condition variable attributes are used, statically allocated
condition variables can be initialized by the macro \fBDEFAULTCV\fR.
.sp
.LP
Default condition variable initialization (intra-process):
.sp
.in +2
.nf
cond_t cvp;

cond_init(&cvp, NULL, NULL); /*initialize condition variable
                                with default*/
.fi
.in -2

.sp
.LP
or
.sp
.in +2
.nf
cond_init(&cvp, USYNC_THREAD, NULL);
.fi
.in -2

.sp
.LP
or
.sp
.in +2
.nf
cond_t  cond  = DEFAULTCV;
.fi
.in -2

.sp
.LP
Customized condition variable initialization (inter-process):
```



```
.sp
.in +2
.nf
cond_init(&cvp, USYNC_PROCESS, NULL); /* initialize cv with
                                         inter-process scope */
.fi
.in -2

.SS "Condition Wait"
.sp
.LP
The condition wait interface allows a thread to wait for a condition and
atomically release the associated mutex that it needs to hold to check the
condition. The thread waits for another thread to make the condition true and
that thread's resulting call to signal and wakeup the waiting thread.
.sp
.LP
The \fBcond_wait()\fR function atomically releases the mutex pointed to by
\fImp\fR and causes the calling thread to block on the condition variable
pointed to by \fIcvp\fR. The blocked thread may be awakened by
\fBcond_signal()\fR, \fBcond_broadcast()\fR, or when interrupted by delivery of
a \fBUNIX\fR signal or a \fBfork()\fR.
.sp
.LP
The \fBcond_wait()\fR, \fBcond_timedwait()\fR, and \fBcond_reltimedwait()\fR
functions always return with the mutex locked and owned by the calling thread
even when returning an error, except when the mutex has the \fBLOCK_ROBUST\fR
attribute and has been left irrecoverable by the mutex's last owner. The
\fBcond_wait()\fR, \fBcond_timedwait()\fR, and \fBcond_reltimedwait()\fR
functions return the appropriate error value if they fail to internally
reacquire the mutex.
.SS "Condition Signaling"
.sp
.LP
A condition signal allows a thread to unblock a single thread waiting on the
condition variable, whereas a condition broadcast allows a thread to unblock
all threads waiting on the condition variable.
.sp
.LP
The \fBcond_signal()\fR function unblocks one thread that is blocked on the
condition variable pointed to by \fIcvp\fR.
.sp
.LP
The \fBcond_broadcast()\fR function unblocks all threads that are blocked on
the condition variable pointed to by \fIcvp\fR.
.sp
.LP
If no threads are blocked on the condition variable, then \fBcond_signal()\fR
and \fBcond_broadcast()\fR have no effect.
.sp
.LP
The \fBcond_signal()\fR or \fBcond_broadcast()\fR functions can be called by a
thread whether or not it currently owns the mutex that threads calling
\fBcond_wait()\fR, \fBcond_timedwait()\fR, or \fBcond_reltimedwait()\fR have
associated with the condition variable during their waits. If, however,
predictable scheduling behavior is required, then that mutex should be locked
by the thread prior to calling \fBcond_signal()\fR or \fBcond_broadcast()\fR.
.SS "Destroy"
.sp
.LP
The condition destroy functions destroy any state, but not the space,
associated with the condition variable.
.sp
.LP
The \fBcond_destroy()\fR function destroys any state associated with the
condition variable pointed to by \fIcvp\fR. The space for storing the condition
```

```
250 variable is not freed.
251 .SH RETURN VALUES
261 .sp
262 .LP
252 Upon successful completion, these functions return \fB0\fR. Otherwise, a
253 non-zero value is returned to indicate the error.
254 .SH ERRORS
266 .sp
267 .LP
255 The \fBcond_timedwait()\fR and \fBcond_reltimedwait()\fR functions will fail
256 if:
257 .sp
258 .ne 2
259 .na
260 \fB\fBETIME\fR\fR
261 .ad
262 .RS 9n
263 The time specified by \fIabstime\fR or \fIreltime\fR has passed.
264 .RE

266 .sp
267 .LP
268 The \fBcond_wait()\fR, \fBcond_timedwait()\fR, and \fBcond_reltimedwait()\fR
269 functions will fail if:
270 .sp
271 .ne 2
272 .na
273 \fB\fBEINTR\fR\fR
274 .ad
275 .RS 9n
276 Interrupted. The calling thread was awakened by the delivery of a UNIX signal.
277 .RE

279 .sp
280 .LP
281 If the mutex pointed to by \fImp\fR is a robust mutex (initialized with the
282 \fBLOCK_ROBUST\fR attribute), the \fBcond_wait()\fR, \fBcond_timedwait()\fR and
283 \fBcond_reltimedwait()\fR functions will, under the specified conditions,
284 return the following error values.  For complete information, see the
285 description of the \fBmutex_lock()\fR function on the \fBmutex_init\fR(3C)
286 manual page.
287 .sp
288 .ne 2
289 .na
290 \fB\fBENOTRECOVERABLE\fR\fR
291 .ad
292 .RS 19n
293 The mutex was protecting the state that has now been left irrecoverable. The
294 mutex has not been acquired.
295 .RE

297 .sp
298 .ne 2
299 .na
300 \fB\fBEOWNERDEAD\fR\fR
301 .ad
302 .RS 19n
303 The last owner of the mutex died while holding the mutex, possibly leaving the
304 state it was protecting inconsistent. The mutex is now owned by the caller.
305 .RE

307 .sp
308 .LP
309 These functions may fail if:
310 .sp
311 .ne 2
```



```
312 .na
313 \fB\fBEFAULT\fR\fR
314 .ad
315 .RS 10n
316 The \fIcond\fR, \fIattr\fR, \fIcvp\fR, \fIarg\fR, \fIabstime\fR, or \fImutex\fR
317 argument points to an illegal address.
318 .RE

320 .sp
321 .ne 2
322 .na
323 \fB\fBEINVAL\fR\fR
324 .ad
325 .RS 10n
326 Invalid argument. For \fBcond_init()\fR, \fItype\fR is not a recognized type.
327 For \fBcond_timedwait()\fR, the number of nanoseconds is greater than or equal
328 to 1,000,000,000.
329 .RE

331 .SH EXAMPLES
345 .LP
332 \fBExample 1 \fRUse \fBcond_wait()\fR in a loop to test some condition.
333 .sp
334 .LP
335 The \fBcond_wait()\fR function is normally used in a loop testing some
349 The \fBcond_wait()\fR functin is normally used in a loop testing some
336 condition, as follows:

338 .sp
339 .in +2
340 .nf
341 (void) mutex_lock(mp);
342 while (cond == FALSE) {
343     (void) cond_wait(cvp, mp);
344 }
_____unchanged_portion_omitted_
397 (void) mutex_unlock(mp);
398 .fi
399 .in -2

401 .SH ATTRIBUTES
416 .sp
417 .LP
402 See \fBattributes\fR(5) for descriptions of the following attributes:
403 .sp

405 .sp
406 .TS
407 box;
408 c | c
409 l | l .
410 ATTRIBUTE TYPE  ATTRIBUTE VALUE
411 _
412 MT-Level        MT-Safe
413 .TE

415 .SH SEE ALSO
432 .sp
433 .LP
416 \fBfork\fR(2), \fBmmap\fR(2), \fBsetitimer\fR(2), \fBshmop\fR(2),
417 \fBmutex_init\fR(3C), \fBsignal\fR(3C), \fBattributes\fR(5),
418 \fBcondition\fR(5), \fBmutex\fR(5), \fBstandards\fR(5)
419 .SH NOTES
438 .sp
439 .LP
420 If more than one thread is blocked on a condition variable, the order in which
```

```
421 threads are unblocked is determined by the scheduling policy. When each thread,
422 unblocked as a result of a \fBcond_signal()\fR or \fBcond_broadcast()\fR,
423 returns from its call to \fBcond_wait()\fR or \fBcond_timedwait()\fR , the
424 thread owns the mutex with which it called \fBcond_wait()\fR,
425 \fBcond_timedwait()\fR, or \fBcond_reltimedwait()\fR. The thread(s) that are
426 unblocked compete for the mutex according to the scheduling policy and as if
427 each had called \fBmutex_lock\fR(3C).
428 .sp
429 .LP
430 When \fBcond_wait()\fR returns the value of the condition is indeterminate and
431 must be reevaluated.
432 .sp
433 .LP
434 The \fBcond_timedwait()\fR and \fBcond_reltimedwait()\fR functions are similar
435 to \fBcond_wait()\fR, except that the calling thread will not wait for the
436 condition to become true past the absolute time specified by \fIabstime\fR or
437 the relative time specified by \fIreltime\fR. Note that \fBcond_timedwait()\fR
438 or \fBcond_reltimedwait()\fR might continue to block as it trys to reacquire
439 the mutex pointed to by \fImp\fR, which may be locked by another thread. If
440 either \fBcond_timedwait()\fR or \fBcond_reltimedwait()\fR returns because of a
441 timeout, it returns the error value \fBETIME\fR.
```

```
**********************************************************
    3621 Sat Feb 15 09:54:06 2020
new/usr/src/man/man9e/awrite.9e
12309 errors in section 9e of the manual
**********************************************************
'\" te
.\"  Copyright (c) 2008, Sun Microsystems, Inc.  All Rights Reserved.
.\" Copyright 1989 AT&T
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH AWRITE 9E "February 15, 2020"
.TH AWRITE 9E "Mar 28, 1997"
.SH NAME
awrite \- asynchronous write to a device
.SH SYNOPSIS
.LP
.nf
#include <sys/uio.h>
#include <sys/aio_req.h>
#include <sys/cred.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>

\fBint prefix\fR\fBawrite\fR(\fBdev_t\fR \fIdev\fR, \fBstruct aio_req *\fR\fIaio
\fBintprefix\fR\fBawrite\fR(\fBdev_t\fR \fIdev\fR, \fBstruct aio_req *\fR\fIaio_
     \fBcred_t *\fR\fIcred_p\fR);
.fi

.SH INTERFACE LEVEL
.sp
.LP
Solaris \fBDDI \fRspecific (Solaris DDI). This entry point is optional. Drivers
that do not support an \fBawrite()\fR entry point should use \fBnodev\fR(9F)
.SH PARAMETERS
.sp
.ne 2
.na
\fB\fIdev\fR\fR
.ad
.RS 12n
Device number.
.RE

.sp
.ne 2
.na
\fB\fIaio_reqp\fR\fR
.ad
.RS 12n
Pointer to the \fBaio_req\fR(9S) structure that describes where the data is
stored.
.RE

.sp
.ne 2
.na
\fB\fIcred_p\fR\fR
.ad
.RS 12n
Pointer to the credential structure.
.RE

.SH DESCRIPTION
.sp
.LP
```



```
The driver's \fBawrite()\fR routine is called to perform an asynchronous write.
\fBgetminor\fR(9F) can be used to access the minor number component of the
\fIdev\fR argument. \fBawrite()\fR may use the credential structure pointed to
by \fIcred_p\fR to check for superuser access by calling \fBdrv_priv\fR(9F).
The \fBawrite()\fR routine may also examine the \fBuio\fR(9S) structure
through the \fBaio_req\fR structure pointer, \fBaio_reqp\fR. \fBawrite()\fR
must call \fBaphysio\fR(9F) with the \fBaio_req\fR pointer and a pointer to the
driver's \fBstrategy\fR(9E) routine.
.sp
.LP
No fields of the \fBuio\fR(9S) structure pointed to by \fBaio_req\fR, other
than \fBuio_offset\fR or \fBuio_loffset\fR, may be modified for non-seekable
devices.
.SH RETURN VALUES
.sp
.LP
The \fBawrite()\fR routine should return  \fB0\fR for success, or the
appropriate error number.
.SH CONTEXT
.sp
.LP
This function is called from user context only.
.SH EXAMPLES
.LP
\fBExample 1 \fRUsing the \fBawrite()\fR routine:
.sp
.LP
The following is an example of an \fBawrite()\fR routine:

.sp
.in +2
.nf
static int
xxawrite(dev_t dev, struct aio_req *aio, cred_t *cred_p)
{
        int instance;
        struct xxstate *xsp;

        instance = getminor(dev);
        xsp = ddi_get_soft_state(statep, instance);
        /*Verify soft state structure has been allocated */
        if (xsp == NULL)
                return (ENXIO);
        return (aphysio(xxstrategy, anocancel, dev, B_WRITE, \e
        xxminphys, aio));
}
.fi
.in -2

.SH SEE ALSO
.sp
.LP
\fBwrite\fR(2), \fBaiowrite\fR(3C), \fBaread\fR(9E), \fBread\fR(9E),
\fBstrategy\fR(9E), \fBwrite\fR(9E), \fBanocancel\fR(9F), \fBaphysio\fR(9F),
\fBddi_get_soft_state\fR(9F), \fBdrv_priv\fR(9F), \fBgetminor\fR(9F),
\fBminphys\fR(9F), \fBnodev\fR(9F), \fBaio_req\fR(9S), \fBcb_ops\fR(9S),
\fBuio\fR(9S)
.sp
.LP
\fIWriting Device Drivers\fR
.SH BUGS
.sp
.LP
There is no way other than calling \fBaphysio\fR(9F) to accomplish an
asynchronous write.
```

```
**********************************************************
   14121 Sat Feb 15 09:54:06 2020
new/usr/src/man/man9e/ddi_ufm.9e
12309 errors in section 9e of the manual
**********************************************************
```

```
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\"
.\" Copyright 2019 Joyent, Inc.
.\"
.Dd February 15, 2020
.Dd Apr 30, 2019
.Dt DDI_UFM 9E
.Os
.Sh NAME
.Nm ddi_ufm ,
.Nm ddi_ufm_op_nimages ,
.Nm ddi_ufm_op_fill_image ,
.Nm ddi_ufm_op_fill_slot ,
.Nm ddi_ufm_op_getcaps
.Nd DDI upgradable firmware module entry points
.Sh SYNOPSIS
.Vt typedef struct ddi_ufm_handle ddi_ufm_handle_t
.Vt typedef struct ddi_ufm_ops ddi_ufm_ops_t
.In sys/ddi_ufm.h
.Ft int
.Fo ddi_ufm_op_getcaps
.Fa "ddi_ufm_handle_t *uhp"
.Fa "void *drv_arg"
.Fa "ddi_ufm_cap_t *caps"
.Fc
.Ft int
.Fo ddi_ufm_op_nimages
.Fa "ddi_ufm_handle_t *uhp"
.Fa "void *drv_arg"
.Fa "uint_t *nimgp"
.Fc
.Ft int
.Fo ddi_ufm_op_fill_image
.Fa "ddi_ufm_handle_t *uhp"
.Fa "void *drv_arg"
.Fa "uint_t imgid"
.Fa "ddi_ufm_image_t *uip"
.Fc
.Ft int
.Fo ddi_ufm_op_fill_slot
.Fa "ddi_ufm_handle_t *uhp"
.Fa "void *drv_arg"
.Fa "uint_t imgid"
.Fa "uint_t slotid"
.Fa "ddi_ufm_slot_t *usp"
.Fc
.Sh INTERFACE LEVEL
.Sy Evolving - This interface is evolving still in illumos. API and ABI stabilit
.Sh PARAMETERS
.Bl -tag -width Fa
.It Fa uhp
A handle corresponding to the device’s UFM handle.
```



```
This is the same value as returned in
.Xr ddi_ufm_init 9F .
.It Fa drv_arg
This is a private value that the driver passed in when calling
.Xr ddi_ufm_init 9F .
.It Fa nimgp
A pointer that the driver should set with a number of images.
.It Fa nslotp
A pointer that the driver should set with a number of slots.
.It Fa imgid
An integer indicating which image information is being requested for.
.It Fa uip
An opaque pointer that represents a UFM image.
.It Fa slotid
An integer indicating which slot information is being requested for.
.It Fa usp
An opaque pointer that represents a UFM slot.
.El
.Sh DESCRIPTION
Upgradable firmware modules (UFM) are a potential component of many
devices.
These interfaces aim to provide a simple series of callbacks
for a device driver to implement such that it is easy to report
information and in the future, manipulate firmware modules.
.Ss UFM Background
UFMs may come in different flavors and styles ranging from a
firmware blob, to an EEPROM image, to microcode, and more.
Take for example a hard drive.
While it is a field replaceable unit (FRU), it also contains some amount
of firmware that manages the drive which can be updated independently of
replacing the drive.
.Pp
The motherboard often has a UFM in the form of the BIOS or UEFI.
The Lights out management controller on a system has a UFM, which is usually
the entire system image.
CPUs also have a UFM in the form of microcode.
.Pp
An important property of a UFM is that it is a property of the device
itself.
For example, many WiFi device drivers are required to send a binary blob of
firmware to the device after every reset.
Because these images are not properties of the device and must be upgraded by
either changing the device driver or related system files, we do not consider
these UFMs.
.Pp
There are also devices that have firmware which is a property of the
device, but may not be upgradable from the running OS.
This may be because the vendor doesn’t have tooling to upgrade the image or
because the firmware image itself cannot be upgraded in the field at all.
For example, a YubiKey has a firmware image that’s burned into it in the
factory, but there is no way to change the firmware on it short of
replacing the device in its entirety.
However, because these images are a permanent part of the device, we also
consider them a UFM.
.Ss Images and Slots
A device that supports UFMs is made up of one or more distinct firmware
images.
Each image has its own unique purpose.
For example, a motherboard may have both a BIOS and a CPLD image, each of which
has independent firmware revisions.
.Pp
A given image may have a number of slots.
A slot represents a particular version of the image.
Only one slot can be active at a given time.
Devices support slots such that a firmware image can be downloaded
to the device without impacting the current device if it fails half-way
```

through.
The slot that’s currently in use is referred to as the
.Em active
slot.
.Pp
The various entry points are designed such that all a driver has to do is provide information about the image and its slots to the kernel, it does not have to wrangle with how that is marshalled to users and the appearance of those structures.
.Ss Registering with the UFM Subsystem
During a device driver’s
.Xr attach 9E
entry point, a device driver should register with the UFM subsystem by filling out a UFM operations vector and then calling
.Xr ddi_ufm_init 9F .
The driver may pass in a value, usually a pointer to its soft state pointer, which it will then receive when its subsequent entry points are called.
.Pp
Once the driver has finished initializing, it must call
.Xr ddi_ufm_update 9F
to indicate that the driver is in a state where it’s ready to receive calls to the entry points.
.Pp
The various UFM entry points may be called from an arbitrary kernel context.
However, they will only ever be called from a single thread at a given time.
.Ss UFM operations vector
The UFM operations vector is a structure that has the following members:
.Bd -literal -offset indent
typedef struct ddi_ufm_ops {
        int (*ddi_ufm_op_nimages)(ddi_ufm_handle_t *uhp, void *arg,
            uint_t *nimgp);
        int (*ddi_ufm_op_fill_image)(ddi_ufm_handle_t *uhp, void *arg,
            uint_t imgid, ddi_ufm_image_t *img);
        int (*ddi_ufm_op_fill_slot)(ddi_ufm_handle_t *uhp, void *arg,
            int imgid, ddi_ufm_image_t *img, uint_t slotid,
            ddi_ufm_slot_t *slotp);
        int (*ddi_ufm_op_getcaps)(ddi_ufm_handle_t *uhp, void *arg,
            ddi_ufm_cap_t *caps);
} ddi_ufm_ops_t;
.Ed
.Pp
The
.Fn ddi_ufm_op_nimages
entry point is optional.
If a device only has a single image, then there is no reason to implement the
.Fn ddi_ufm_op_nimages entry point.
The system will assume that there is only a single image.
.Pp
Slots and images are numbered starting at zero.
If a driver indicates support for multiple images or slots then the images or slots will be numbered sequentially going from 0 to the number of images or slots minus one.
These values will be passed to the various entry points to indicate which image and slot the system is interested in.
It is up to the driver to maintain a consistent view of the images and slots for a given UFM.
.Pp
The members of this structure should be filled in the following ways:
.Bl -tag -width Fn
.It Fn ddi_ufm_op_nimages
The
.Fn ddi_ufm_op_nimages
entry point is an optional entry point that answers the question of how

many different, distinct firmware images are present on the device.
Once the driver determines how many are present, it should set the value in
.Fa nimgp to the determined value.
.Pp
It is legal for a device to pass in zero for this value, which indicates that there are none present.
.Pp
Upon successful completion, the driver should return
.Sy 0 .
Otherwise, the driver should return the appropriate error number.
For a full list of error numbers, see
.Xr Intro 2 .
Common values are:
.Bl -tag -width Er -offset width
.It Er EIO
An error occurred while communicating with the device to determine the number of firmware images.
.El
.It Fn ddi_ufm_op_fill_image
The
.Fn ddi_ufm_op_fill_image
entry point is used to fill in information about a given image.
The value in
.Fa imgid
is used to indicate which image the system is asking to fill information about.
If the driver does not recognize the image ID in
.Fa imgid
then it should return an error.
.Pp
The
.Ft ddi_ufm_image_t
structure passed in
.Fa uip
is opaque.
To fill in information about the image, the driver should call the functions described in
.Xr ddi_ufm_image 9F .
.Pp
The driver should call the
.Xr ddi_ufm_image_set_desc 9F
function to set a description of the image which indicates its purpose.
This should be a human-readable string.
The driver may also set any ancillary data that it deems may be useful with the
.Xr ddi_ufm_image_set_misc 9F function.
This function takes an nvlist, allowing the driver to set arbitrary keys and val
.Pp
Once the driver has finished setting all of the information about the image then the driver should return
.Sy 0 .
Otherwise, the driver should return the appropriate error number.
For a full list of error numbers, see
.Xr Intro 2 .
Common values are:
.Bl -tag -width Er -offset width
.It Er EINVAL
The image indicated by
.Fa imgid
is unknown.
.It Er EIO
An error occurred talking to the device while trying to fill out firmware image information.
.It Er ENOMEM
The driver was unable to allocate memory while filling out image information.
.El

```
.It Fn ddi_ufm_op_fill_slot
The
.Fn ddi_ufm_op_fill_slot
function is used to fill in information about a specific slot for a
specific image.
The value in
.Fa imgid
indicates the image the system wants slot information for and the value
in
.Fa slotid
indicates which slot of that image the system is interested in.
If the device driver does not recognize the value in either or
.Fa imgid
or
.Fa slotid ,
then it should return an error.
.Pp
The
.Ft ddi_ufm_slot_t
structure passed in
.Fa usp
is opaque.
To fill in information about the image the driver should call the functions
described in
.Xr ddi_ufm_slot 9F .
.Pp
The driver should call the
.Xr ddi_ufm_slot_set_version 9F
function to indicate the version of the UFM.
The version is a device-specific character string.
It should contain the current version of the UFM as a human can understand it
and it should try to match the format used by device vendor.
.Pp
The
.Xr ddi_ufm_slot_set_attrs 9F
function should be used to set the attributes of the UFM slot.
These attributes include the following enumeration values:
.Bl -tag -width Dv
.It Dv DDI_UFM_ATTR_READABLE
This attribute indicates that the firmware image in the specified slot
may be read, even if the device driver does not currently support such
functionality.
.It Dv DDI_UFM_ATTR_WRITEABLE
This attributes indicates that the firmware image in the specified slot
may be updated, even if the driver does not currently support such
functionality.
.It Dv DDI_UFM_ATTR_ACTIVE
This attributes indicates that the firmware image in the specified slot
is the active
.Pq i.e. currently running
firmware.
Only one slot should be marked active.
.It Dv DDI_UFM_ATTR_EMPTY
This attributes indicates that the specified slot does not currently contain
any firmware image.
.El
.Pp
Finally, if there are any device-specific key-value pairs that form
useful, ancillary data, then the driver should assemble an nvlist and
pass it to the
.Xr ddi_ufm_set_misc 9F
function.
.Pp
Once the driver has finished setting all of the information about the
slot then the driver should return
.Sy 0 .
```



```
Otherwise, the driver should return the appropriate error number.
For a full list of error numbers, see
.Xr Intro 2 .
Common values are:
.Bl -tag -width Er -offset width
.It Er EINVAL
The image or slot indicated by
.Fa imgid
and
.Fa slotid
is unknown.
.It Er EIO
An error occurred talking to the device while trying to fill out
firmware slot information.
.It Er ENOMEM
The driver was unable to allocate memory while filling out slot
information.
.El
.It Fn ddi_ufm_op_getcaps
The
.Fn ddi_ufm_op_getcaps
function is used to indicate which DDI UFM capabilities are supported by this
driver instance.
Currently there is only a single capability
.Pq DDI_UFM_CAP_REPORT
which indicates that the driver is capable of reporting UFM information for this
instance.
Future UFM versions may add additional capabilities such as the ability to
obtain a raw dump of the firmware image or to upgrade the firmware.
.Pp
The driver should indicate the supported capabilities by setting the value in
the
.Ft caps
parameter.
Once the driver has populated
.Ft caps
with an appropriate value, then the driver should return
.Sy 0 .
Otherwise, the driver should return the appropriate error number.
For a full list of error numbers, see
.Xr Intro 2 .
Common values are:
.Bl -tag -width Er -offset width
.It Er EIO
An error occurred talking to the device while trying to discover firmware
**capabilities.**
*capabilties.*
.It Er ENOMEM
The driver was unable to allocate memory.
.El
.El
.Ss Caching and Updates
The system will fetch firmware and slot information on an as-needed
basis.
Once it obtains some information, it may end up caching this information on
behalf of the driver.
Whenever the driver believes that something could have changed -- it need know
that it has -- then the driver must call
.Xr ddi_ufm_update 9F .
.Ss Locking
All UFM operations on a single UFM handle will always be run serially.
However, the device driver may still need to apply adequate locking to
its structure members as other may be accessing the same data structure
or trying to communicate with the device.
.Ss Unregistering from the UFM subsystem
When a device driver is detached, it should unregister from the UFM
```

```
390 subsystem.
391 To do so, the driver should call
392 .Xr ddi_ufm_fini 9F .
393 By the time this function returns, the driver is guaranteed that no UFM
394 entry points will be called.
395 However, if there are outstanding UFM related activity, the function will
396 block until it is terminated.
397 .Ss ioctl Interface
398 Userland consumers can access UFM information via a set of ioctls that are
399 implemented by the
400 .Xr ufm 7D
401 driver.
402 .Sh CONTEXT
403 The various UFM entry points that a device driver must implement will
404 always be called from
405 .Sy kernel
406 context.
407 .Sh SEE ALSO
408 .Xr Intro 2 ,
409 .Xr ufd 7D ,
410 .Xr attach 9E ,
411 .Xr ddi_ufm_fini 9F ,
412 .Xr ddi_ufm_image 9F ,
413 .Xr ddi_ufm_image_set_desc 9F ,
414 .Xr ddi_ufm_image_set_misc 9F ,
415 .Xr ddi_ufm_image_set_nslots 9F ,
416 .Xr ddi_ufm_init 9F ,
417 .Xr ddi_ufm_slot 9F ,
418 .Xr ddi_ufm_slot_set_attrs 9F ,
419 .Xr ddi_ufm_slot_set_misc 9F ,
420 .Xr ddi_ufm_slot_set_version 9F ,
421 .Xr ddi_ufm_update 9F
```

```
**********************************************************
   12469 Sat Feb 15 09:54:06 2020
new/usr/src/man/man9e/mac_capab_transceiver.9e
12309 errors in section 9e of the manual
**********************************************************
   1 .\"
   2 .\" This file and its contents are supplied under the terms of the
   3 .\" Common Development and Distribution License ("CDDL"), version 1.0.
   4 .\" You may only use this file in accordance with the terms of version
   5 .\" 1.0 of the CDDL.
   6 .\"
   7 .\" A full copy of the text of the CDDL should have accompanied this
   8 .\" source.  A copy of the CDDL is also available via the Internet at
   9 .\" http://www.illumos.org/license/CDDL.
  10 .\"
  11 .\"
  12 .\" Copyright (c) 2017, Joyent, Inc.
  13 .\"
  14 .Dd February 15, 2020
  14 .Dd Nov 26, 2017
  15 .Dt MAC_CAPAB_TRANSCEIVER 9E
  16 .Os
  17 .Sh NAME
  18 .Nm mac_capab_transceiver ,
  19 .Nm mct_info ,
  20 .Nm mct_read
  21 .Nd MAC capability for networking transceivers
  22 .Sh SYNOPSIS
  23 .In sys/mac_provider.h
  24 .Vt typedef struct mac_capab_transceiver mac_capab_transceiver_t;
  25 .Ft int
  26 .Fo "mct_info"
  27 .Fa "void *driver"
  28 .Fa "uint_t id"
  29 .Fa "mac_transceiver_info_t *infop"
  30 .Fc
  31 .Ft int
  32 .Fo mct_read
  33 .Fa "void *driver"
  34 .Fa "uint_t id"
  35 .Fa "uint_t page"
  36 .Fa "void *buf"
  37 .Fa "size_t nbytes"
  38 .Fa "off_t offset"
  39 .Fa "size_t *nread"
  40 .Fc
  41 .Sh INTERFACE LEVEL
  42 .Sy Volatile -
  43 This interface is still evolving in illumos.
  44 API and ABI stability is
  45 not guaranteed.
  46 .Sh PARAMETERS
  47 .Bl -tag -width Fa
  48 .It Fa driver
  49 A pointer to the driver’s private data that was passed in via the
  50 .Sy m_pdata
  51 member of the
  52 .Xr mac_register 9S
  53 structure to the
  54 .Xr mac_register 9F
  55 function.
  56 .It Fa id
  57 An integer value indicating which transceiver is being inquired about.
  58 .It Fa infop
  59 An opaque structure which is used to set information about the
  60 transceiver.
```



```
  61 .It Fa page
  62 A value that indicates which page from the i2c bus is being requested.
  63 .It Fa buf
  64 A pointer to which data should be written to when reading from the
  65 device.
  66 .It Fa nbytes
  67 A value indicating the number of bytes being asked to read into
  68 .Fa buf .
  69 .It Fa offset
  70 A value indicating the offset into the page to start reading data.
  71 .It Fa nread
  72 A value to be updated by the driver with the number of successfully read
  73 bytes.
  74 .El
  75 .Sh DESCRIPTION
  76 The
  77 .Sy MAC_CAPAB_TRANSCEIVER
  78 capability allows for GLDv3 networking device drivers to provide
  79 information to the system about their transceiver.
  80 Implementing this capability is optional.
  81 For more information on how to handle capabilities and how to indicate
  82 that a capability is not supported, see
  83 .Xr mc_getcapab 9E .
  84 .Pp
  85 This capability should be implemented if the device in question supports
  86 a Small Form Factor (SFF) transceiver.
  87 These are more commonly known by names such as SFP, SFP+, SFP28, QSFP+,
  88 and QSFP28.
  89 This interface does not apply to traditional copper Ethernet phys.
  90 These transceivers provide standardized information over the i2c bus at
  91 specific pages.
  92 .Ss Supported Standards
  93 .Bl -tag -width Sy
  94 .It Sy INF-8074
  95 The
  96 .Sy INF-8084
  97 standard was the original multiple source agreement (MSA) for SFP
  98 devices.
  99 It proposed the original series of management pages at i2c page 0xa0.
 100 This page contained up to 512 bytes, however, only the first
 101 96 bytes are standardized.
 102 Bytes 97 to 127 are reserved for the vendor.
 103 The remaining bytes are reserved by the specification.
 104 The management page was subsequently adopted by SFP+ devices.
 105 .It Sy SFF-8472
 106 The
 107 .Sy SFF-8472
 108 standard extended the original SFP MSA.
 109 This standard added a second i2c page at 0xa2, while maintaining the
 110 original page at 0xa0.
 111 The page at 0xa0 is now explicitly 256 bytes.
 112 The page at 0xa2 is also 256 bytes.
 113 This standard was also adopted for all SFP28 parts, which are commonly
 114 used in transceivers for 25 Gb/s Ethernet.
 115 .It Sy SFF-8436
 116 The
 117 .Sy SFF-8436
 118 standard was developed for QSFP+ transceivers, which involve the
 119 bonding of 4 SFP+ links.
 120 QSFP+ is commonly used in the transceivers for 40 Gb/s Ethernet.
 121 This standard uses i2c page 0xa0 for read-only identification purposes.
 122 The lower half of the page is used for control, while the upper 128
 123 bytes is similar to the
 124 .Sy INF-8084
 125 and
 126 .Sy SFF-8472
```

```
127 standards.
128 .It Sy SFF-8636
129 The
130 .Sy SFF-8636
131 standard is a common management standard which is shared between both
132 SAS and QSFP+ 28 Gb/s transceivers.
133 The latter transceiver is commonly found in 100 Gb/s Ethernet.
134 The transceiver’s memory map is similar to that found in the
135 .Sy SFF-8436
136 specification.
137 The identification information is found in the upper 128
138 bytes of page 0xa0, while the lower part of the page is used for
139 control, among other purposes.
140 .El
141 .Pp
142 The following table summarizes the above information.
143 .Bl -column "Sy SFF-8636" "1 Gb/s, 10 Gb/s, 25 Gb/s" "256 bytes" "0xa0, 0xa2" -o
144 .Em "Standard" Ta Em Speeds Ta Em Size Ta Em i2c pages
145 .It INF-8074 Ta 1 Gb/s, 10 Gb/s Ta 128 bytes Ta 0xa0
146 .It SFF-8472 Ta 1 Gb/s, 10 Gb/s, 25 GB/s Ta 512 bytes Ta 0xa0, 0xa2
147 .It SFF-8436 Ta 40 Gb/s Ta 256 bytes Ta 0xa0
148 .It SFF-8636 Ta 100 Gb/s Ta 256 bytes Ta 0xa0
149 .El
150 .Ss MAC Capability Structure
151 When the device driver’s
152 .Xr mc_getcapab 9E
153 function entry point is called with the capability requested set to
154 .Sy MAC_CAPAB_TRANSCEIVER ,
155 then the value of the capability structure is the following structure:
156 .Bd -literal -offset indent
157 typedef struct mac_capab_transceiver {
158         uint_t  mct_flags;
159         uint_t  mct_ntransceivers;
159         uint_t  mct_ntransceiveres;
160         int     (*mct_info)(void *driver, uint_t id,
161                     mac_transceiver_info_t *infop),
162         int     (*mct_read)(void *driver, uint_t id, uint_t page,
163                     void *buf, size_t nbytes, off_t offset,
164                     size_t *nread)
165 } mac_capab_transceiver_t;
166 .Ed
167 .Pp
168 If the device driver supports the
169 .Sy MAC_CAPAB_TRANSCEIVER
170 capability, it should fill in this structure, based on the following
171 rules:
172 .Bl -tag -width Sy
173 .It Sy mct_flags
174 The
175 .Vt mct_flags
176 member is used to negotiate extensions with the driver.
177 MAC will set the value of
178 .Vt mct_flags
179 to include all of the currently known extensions.
180 The driver should intersect this list with the set that they actually
181 support.
182 At this time, no such features are defined and the driver should set the
183 member to
184 .Sy 0 .
185 .It Sy mct_ntransceivers
186 The value of
187 .Sy mct_ntransceivers
188 indicates the number of transceivers present in the device.
188 indicates that the number of transceivers present in the device.
189 For most devices, it is expected that this value will be set to one.
190 However, some devices do support multiple transceivers and PHYs that
```



```
191 show up behind a single logical MAC.
192 .Pp
193 It is expected that this value will not change across the lifetime of
194 the device being attached.
195 It is important to remember that this represents the total possible
196 number of transceivers in the device, not how many are currently present
197 and powered on.
198 .Pp
199 The number of transceivers will influence the
200 .Fa id
201 argument used in the
202 .Fn mct_info
203 and
204 .Fn mct_read
205 entry points.
206 The transceiver IDs will start at zero and go to the value of
207 .Fa mct_ntransceivers - 1 .
208 It is up to the driver to keep the mapping between actual transceivers
209 and the transceiver identifiers consistent.
210 .It Sy mct_info
211 The
212 .Fn mct_info
213 entry point is used to set basic information about the transceiver.
214 This entry point is
215 .Em required .
216 If the device driver cannot implement this entry point, then it should
217 not indicate that it supports the capability.
218 .Pp
219 The
220 .Fn mct_info
221 entry point should fill in information about the transceiver with an
222 identifier of
223 .Fa id .
224 See the description above of
225 .Sy mct_ntransceivers
226 for more information on how the IDs are determined.
227 .Pp
228 The driver should then proceed to fill in basic information by calling
229 the functions described in the section
230 .Sx Information Functions .
231 After successfully calling all of the functions, the driver should
232 return
233 .Sy 0 .
234 Otherwise, it should return the appropriate error number.
234 Othewrise, it should return the appropriate error number.
235 For a full list of error numbers, see
236 .Xr Intro 2 .
237 Common values are:
238 .Bl -tag -width Er -offset width
239 .It Er EINVAL
240 The transceiver identifier
241 .Fa id
242 was invalid.
243 .It Er ENOTSUP
244 This instance of the devices does not support a transceiver.
245 For example, a device which sometimes has copper PHYs and therefore this
246 instance does not have any PHYs.
247 .It Er EIO
248 An error occurred while trying to read device registers.
249 For example, an FM-aware device had an error.
250 .El
251 .It Sy mct_read
252 The
253 .Fn mct_read
254 function is used to read information from a transceiver’s i2c bus.
255 The
```

```
.Fn mct_read
entry point is an
.Em optional
entry point.
.Pp
The transceiver should first check the value of
.Fa id ,
which indicates which transceiver information is being requested.
See the description above of
.Sy mct_ntransceivers
for more information on how the IDs are determined.
.Pp
The driver should try to read up to
.Fa nbytes
of data from the i2c bus at page
.Fa page .
The driver should start reading at offset
.Fa offset .
Finally, it should update the value in
.Fa nread
with the number of bytes written to the buffer
.Fa buf .
.Pp
If for some reason the driver cannot read all of the requested bytes,
that is acceptable.
Instead it should perform a short read.
This may occur because the transceiver does not allow reads at a
requested region or the region is shorter than is common for most
devices.
.Pp
Upon successful completion, the driver should ensure that
.Fa nread
has been updated and then return
.Sy 0 .
Otherwise, the driver should return the appropriate error number.
For
a full list of error numbers, see
.Xr Intro 2 .
Common values are:
.Bl -tag -width Er -offset width
.It Er EINVAL
The value of
.Fa id
represented an invalid transceiver identifier.
The transceiver i2c page
.Fa page
is not valid for this type of device.
The value of
.Fa offset
is beyond the range supported for this
.Fa page .
.It Er EIO
An error occurred while trying to read the device i2c pages.
.El
.El
.Ss Transceiver Information Functions
The
.Fn mct_info
entry point is the primary required entry point for a device driver
which supports this capability.
The information structure is opaque to the device driver.
Instead, a series of informational functions is
available to the device driver to call on the transceiver.
The device drivers should try to call and fill in as many of these as
possible.
There are two different properties that a driver can set:
```



```
.Bl -enum -offset indent
.It
Whether the transceiver is present.
.It
Whether the transceiver is usable.
.El
.Pp
To set whether or not the transceiver is present, the driver should call
.Xr mac_transceiver_info_set_present 9F .
This is used to indicate whether the transceiver is plugged in or not.
If the transceiver is a part of the NIC, then this function should
always be called with the value set to
.Dv B_TRUE .
.Pp
Finally, the driver has the ability to provide information about whether
or not the transceiver is usable or not.
A transceiver may be present, but not usable, if the hardware and
firmware support a limited number of transceivers.
To set this information, the driver should call
.Xr mac_transceiver_info_set_usable 9F .
If the transceiver is not present, then the driver should not call this
function.
.Ss Opaque Transceivers
Some devices abstract the nature of the transceiver and do not allow
direct access to the transceiver.
In this case, if the device driver still has access to enough
information to know if the transceiver is at least present, then it
should still implement the
.Fn mct_info
entry point.
.Ss Locking and Data Access
Calls to get information about the transceivers may come at the same
time as general I/O requests to the device to send or receive data.
The driver should make sure that reading data from the i2c bus of the
transceiver does not interfere with the device's functionality in this
regard.
Different locks should be used.
.Pp
On some devices, reading from the transceiver's i2c bus might cause a
disruption of service to the device.
For example, on some devices a phy reset may be required or come about
as a side effect of trying to read the device.
If any kind of disruption would be caused, then the driver
must not implement the
.Ft mct_read
entry point.
.Sh CONTEXT
The various callback functions will be called from
.Sy kernel
context.
These functions will never be called from
.Sy interrupt
context.
.Sh SEE ALSO
.Xr Intro 2 ,
.Xr mac 9E ,
.Xr mc_getcapab 9E ,
.Xr mac_register 9F ,
.Xr mac_transceiver_info_set_present 9F ,
.Xr mac_transceiver_info_set_usable 9F ,
.Xr mac_register 9S
.Rs
.%N INF-8074i
.%T SFP (Small Formfactor Pluggable) Interface
.%Q SFF Committee
.%O Revision 1.0
```

```
 388 .%D May 12, 2001
 389 .Re
 390 .Rs
 391 .%N SFF-8472
 392 .%T Diagnostic Monitoring Interface for Optical Transceivers
 393 .%O Revision 12.2
 394 .%D November 21, 2014
 395 .Re
 396 .Rs
 397 .%N SFF-8436
 398 .%T QSFP+ 10 Gbs 4X PLUGGABLE TRANSCEIVER
 399 .%O Revision 4.8
 400 .%D October 31, 2013
 401 .Re
 402 .Rs
 403 .%N SFF-8636
 404 .%T Management Interface for Cabled Environments
 405 .%O Revision 2.7
 406 .%D January 26, 2016
 407 .Re
```

****
6020 Sat Feb 15 09:54:06 2020
new/usr/src/man/man9e/mc_getprop.9e
12309 errors in section 9e of the manual
****

```
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\"
.\" Copyright 2016 Joyent, Inc.
.\"
.Dd February 15, 2020
.Dd November 15, 2016
.Dt MC_GETPROP 9E
.Os
.Sh NAME
.Nm mc_getprop
.Nd get device properties
.Sh SYNOPSIS
.In sys/mac_provider.h
.Ft int
.Fo prefix_m_getprop
.Fa "void *driver"
.Fa "const char *pr_name"
.Fa "mac_prop_id_t pr_num"
.Fa "uint_t pr_valsize"
.Fa "void *pr_val"
.Fc
.Sh INTERFACE LEVEL
illumos DDI specific
.Sh PARAMETERS
.Bl -tag -width Fa
.It Fa driver
A pointer to the driver’s private data that was passed in via the
.Sy m_pdata
member of the
.Xr mac_register 9S
structure to the
.Xr mac_register 9F
function.
.It Fa pr_name
A null-terminated string that contains the name of the property.
.It Fa pr_num
A constant that is used to identify the property.
.It Fa pr_valsize
A value that indicates the size in bytes of
.Fa pr_val .
.It Fa pr_val
A pointer to a
.Fa pr_valsize
byte buffer that can store the property.
.El
.Sh DESCRIPTION
The
.Fn mc_getprop
entry point is used to obtain the value of a given device’s property and
place it into
.Fa pr_val .
.Pp
```



```
When the
.Fn mc_getprop
entry point is called, the driver needs to first identify the property.
The set of possible properties and their meaning is listed in the
.Sx PROPERTIES
section of
.Xr mac 9E .
It should identify the property based on the value of
.Fa pr_num .
Most drivers will use a
.Sy switch
statement and for any property that it supports it should then check if
the value in
.Fa pr_valsize
is sufficient for the property, comparing it to the minimum size
listed for the property in
.Xr mac 9E .
If it is not, then it should return an error.
Otherwise, it should copy the property’s value into
.Fa pr_val .
When an unknown or unsupported
property is encountered, generally the
.Sy default
case of the switch statement, the device driver should return an error.
.Pp
The special property
.Sy MAC_PROP_PRIVATE
indicates that this is a device driver specific private property.
The device driver must then look at the value of the
.Fa pr_name
argument and use
.Xr strcmp 9F
on it, comparing it to each of its private (bounded-size) properties to
identify which one it is.
.Pp
At this time, private properties are limited to being string based properties.
If other types of property values are used, they will not be rendered
correctly by
.Xr dladm 1M .
.Pp
The device
driver can access its device soft state by casting the
.Fa device
pointer to the appropriate structure.
As this may be called while other operations are ongoing, the device driver
should employ the appropriate locking while reading the properties.
.Sh CONTEXT
The
.Fn mc_getprop
function is generally called from
.Sy kernel
context.
.Sh RETURN VALUES
Upon successful completion, the device driver should have copied the
value of the property into
.Fa pr_val
and return
.Sy 0 .
Otherwise, a positive error should be returned to indicate failure.
.Sh EXAMPLES
The following example shows how a device driver might structure its
.Fn mc_getprop
entry point.
.Bd -literal
#include <sys/mac_provider.h>
```

```
 127 /*
 128  * Note, this example merely shows the structure of this function.
 129  * Different devices will manage their state in different ways. Like other
 130  * examples, this assumes that the device has state in a structure called
 131  * example_t and that there is a lock which keeps track of that state.
 132  */
 133 static char *example_priv_props[] = {
 134         "_rx_intr_throttle",
 135         "_tx_intr_throttle",
 136         NULL
 137 };

 139 static int
 140 example_m_getprop_private(example_t *ep, const char *pr_name, uint_t pr_valsize,
 141     void *pr_val)
 142 {
 143         uint32_t val;

 145         ASSERT(MUTEX_HELD(&ep->ep_lock));
 146         if (strcmp(pr_name, example_priv_props[0] == 0) {
 147                 val = ep->ep_rx_itr;
 148         } else if (strcmp(pr_name, example_priv_props[1] == 0) {
 148         } else if (strcmp(pr_name, exampe_priv_props[1] == 0) {
 149                 val = ep->ep_tx_itr;
 150         } else {
 151                 return (ENOTSUP);
 152         }

 154         /*
 155          * Due to issues in the GLDv3, these must be returned as string
 156          * properties.
 157          */
 158         if (snprintf(pr_val, pr_valsize, "%d", val) >= pr_valsize)
 159                 return (EOVERFLOW);

 161         return (0);
 162 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    6951 Sat Feb 15 09:54:06 2020
new/usr/src/man/man9e/mc_setprop.9e
12309 errors in section 9e of the manual
**********************************************************
```

```
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\"
.\" Copyright 2016 Joyent, Inc.
.\"
.Dd February 15, 2020
.Dd June 02, 2016
.Dt MC_SETPROP 9E
.Os
.Sh NAME
.Nm mc_setprop
.Nd set device properties
.Sh SYNOPSIS
.In sys/mac_provider.h
.Ft int
.Fo prefix_m_setprop
.Fa "void *driver"
.Fa "const char *pr_name"
.Fa "mac_prop_id_t pr_num"
.Fa "uint_t pr_valsize"
.Fa "const void *pr_val"
.Fc
.Sh INTERFACE LEVEL
illumos DDI specific
.Sh PARAMETERS
.Bl -tag -width Fa
.It Fa driver
A pointer to the driver’s private data that was passed in via the
.Sy m_pdata
member of the
.Xr mac_register 9S
structure to the
.Xr mac_register 9F
function.
.It Fa pr_name
A null-terminated string that contains the name of the property.
.It Fa pr_num
A constant that is used to identify the property.
.It Fa pr_valsize
A value that indicates the size in bytes of
.Fa pr_val .
.It Fa pr_val
A pointer to a
.Fa pr_valsize
byte buffer that contains the new value of the property.
.El
.Sh DESCRIPTION
The
.Fn mc_setprop
entry point is used to set the value of a given device’s property from
the copy stored in
.Fa pr_val .
.Pp
```



```
When the
.Fn mc_setprop
entry point is called, the driver needs to first identify the property.
The set of possible properties and their meaning is listed in the
.Sx PROPERTIES
section of
.Xr mac 9E .
It should identify the property based on the value of
.Fa pr_num .
Most drivers will use a
.Sy switch
statement and for any property that it supports it should then check if
the value in
.Fa pr_valsize
is sufficient for the property, comparing it to the minimum size
listed for the property in
.Xr mac 9E .
If it is not, then it should return an error.
Otherwise, it should update the property based on the value in
.Fa pr_val .
When an unknown or unsupported property is encountered, generally the
.Sy default
case of the switch statement, the device driver should return an error.
.Pp
The special property
.Sy MAC_PROP_PRIVATE
indicates that this is a device driver specific private property.
The device driver must then look at the value of the
.Fa pr_name
argument and use
.Xr strcmp 9F
on it, comparing it to each of its private properties to identify which
one it is.
.Pp
Not all properties are supposed to be writable.
Some devices may opt to not allow a property that is designated as read/write to
be set.
When such a property is encountered, the driver should return the appropriate
error.
.Pp
The device
driver can access its device soft state by casting the
.Fa device
pointer to the appropriate structure.
As this may be called while other operations are ongoing, the device driver
should employ the appropriate locking while writing the properties.
.Sh RETURN VALUES
Upon successful completion, the device driver should have copied the
value of the property into
.Fa pr_val
and return
.Sy 0 .
Otherwise, a positive error should be returned to indicate failure.
.Sh EXAMPLES
The following examples shows how a device driver might structure its
.Fn mc_setprop
.Fn mc_setporp
entry point.
.Bd -literal
#include <sys/mac_provider.h>

/*
 * Note, this example merely shows the structure of this function.
 * Different devices will manage their state in different ways. Like other
 * examples, this assumes that the device has state in a structure called
 * example_t and that there is a lock which keeps track of that state.
```

```
 126  *
 127  * For the purpose of this example, we assume that this device supports 100 Mb,
 128  * 1 GB, and 10 Gb full duplex speeds.
 129  */

 131 static int
 132 example_m_setprop(void *arg, const char *pr_name, mac_prop_id_t pr_num,
 132 exmple_m_setprop(void *arg, const char *pr_name, mac_prop_id_t pr_num,
 133     uint_t pr_valsize, const void *pr_val)
 134 {
 135         uint32_t new_mtu;
 136         int ret = 0;
 137         example_t *ep = arg;

 139         mutex_enter(&ep->ep_lock);
 140         switch (pr_num) {
 141         /*
 142          * These represent properties that can never be changed, regardless of
 143          * the type of PHY on the device (copper, fiber, etc.)
 144          */
 145         case MAC_PROP_DUPLEX:
 146         case MAC_PROP_SPEED:
 147         case MAC_PROP_STATUS:
 148         case MAC_PROP_ADV_100FDX_CAP:
 149         case MAC_PROP_ADV_1000FDX_CAP:
 150         case MAC_PROP_ADV_10GFDX_CAP:
 151                 ret = ENOTSUP;
 152                 break;

 154         /*
 155          * These EN properties are used to control the advertised speeds of the
 156          * device. For this example, we assume that this device does not have a
 157          * copper phy, at which point auto-negotiation and the speeds in
 158          * question cannot be changed. These are called out separately as they
 159          * should be controllable for copper based devices or it may need to be
 160          * conditional depending on the type of phy present.
 161          */
 162         case MAC_PROP_EN_100FDX_CAP:
 163         case MAC_PROP_EN_1000FDX_CAP:
 164         case MAC_PROP_EN_10GFDX_CAP:
 165         case MAC_PROP_AUTONEG:
 166                 ret = ENOTSUP;
 167                 break;

 169         case MAC_PROP_MTU:
 170                 if (pr_valsize < sizeof (uint32_t)) {
 171                         ret = EOVERFLOW;
 172                         break;
 173                 }
 174                 bcopy(&new_mtu, pr_val, sizeof (uint32_t));

 176                 if (new_mtu < ep->ep_min_mtu ||
 177                     new_mtu > ep->ep_max_mtu) {
 178                         ret = EINVAL;
 179                         break;
 180                 }

 182                 /*
 183                  * We first ask MAC to update the MTU before we do anything.
 184                  * This may fail. It returns zero on success. The
 185                  * example_update_mtu function does device specific updates to
 186                  * ensure that the MTU on the device is updated and any internal
 187                  * data structures are up to date.
 188                  */
 189                 ret = mac_maxdsu_update(&ep->ep_mac_hdl, new_mtu);
 190                 if (ret == 0) {
```



```
 191                         example_update_mtu(ep, new_mtu);
 192                 }
 193                 break;

 195         /*
 196          * Devices may have their own private properties. If they do, they
 197          * should not return ENOTSUP, but instead see if it's a property they
 198          * recognize and handle it similar to those above. If it doesn't
 199          * recognize the name, then it should return ENOTSUP.
 200          */
 201         case MAC_PROP_PRIVATE:
 202                 ret = ENOTSUP;
 203                 break;

 205         default:
 206                 ret = ENOTSUP;
 207                 break;
 208         }
 209         mutex_exit(&ep->ep_lock);

 211         return (ret);
 212 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3716 Sat Feb 15 09:54:07 2020
new/usr/src/man/man9e/probe.9e
12309 errors in section 9e of the manual
**********************************************************
'\" te
.\" Copyright (c) 2000, Sun Microsystems, Inc.
.\"  All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH PROBE 9E "February 15, 2020"
.TH PROBE 9E "Nov 18, 1992"
.SH NAME
probe \- determine if a non-self-identifying device is present
.SH SYNOPSIS
.LP
.nf
#include <sys/conf.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>



\fBstatic int prefix\fR\fBprobe\fR(\fBdev_info_t *\fR\fIdip\fR);
\fBstatic intprefix\fR\fBprobe\fR(\fBdev_info_t *\fR\fIdip\fR);
.fi

.SH INTERFACE LEVEL
.sp
.LP
Solaris DDI specific (Solaris DDI). This entry point is required for
non-self-identifying devices.  You must write it for such devices. For
self-identifying devices, \fBnulldev\fR(9F) should be specified in the
\fBdev_ops\fR(9S) structure if a probe routine is not necessary.
.SH ARGUMENTS
.sp
.ne 2
.na
\fB\fIdip\fR \fR
.ad
.RS 8n
Pointer to the device's \fBdev_info\fR structure.
.RE

.SH DESCRIPTION
.sp
.LP
\fBprobe()\fR determines whether the device corresponding to \fIdip\fR actually
exists and is a valid device for this driver. \fBprobe()\fR is called after
\fBidentify\fR(9E) and before \fBattach\fR(9E) for a given \fIdip\fR. For
example, the \fBprobe()\fR routine can map the device registers using
\fBddi_map_regs\fR(9F) then attempt to access the hardware using
\fBddi_peek\fR(9F) or \fBddi_poke\fR(9F) and determine if the device exists.
Then the device registers should be unmapped using \fBddi_unmap_regs\fR(9F).
.sp
.LP
To probe a device that was left powered off after the last \fBdetach()\fR, it
might be necessary to power it up. If so, the driver must power up the device
by accessing device registers directly. \fBpm_raise_power\fR(9F) will be not be
available until \fBattach\fR(9E). The framework ensures that the ancestors of
the node being probed and all relevant platform-specific power management
hardware is at full power at the time that \fBprobe()\fR is called.
.sp
.LP
\fBprobe()\fR should only probe the device. It should not change any software
```



```
state and should not create any software state. Device initialization should be
done in \fBattach\fR(9E).
.sp
.LP
For a self-identifying device, this entry point is not necessary. However, if a
device exists in both self-identifying and non-self-identifying forms, a
\fBprobe()\fR routine can be provided to simplify the driver.
\fBddi_dev_is_sid\fR(9F) can then be used to determine whether \fBprobe()\fR
needs to do any work. See \fBddi_dev_is_sid\fR(9F) for an example.
.SH RETURN VALUES
.sp
.ne 2
.na
\fB\fBDDI_PROBE_SUCCESS\fR \fR
.ad
.RS 23n
If the probe was successful.
.RE

.sp
.ne 2
.na
\fB\fBDDI_PROBE_FAILURE\fR \fR
.ad
.RS 23n
If the probe failed.
.RE

.sp
.ne 2
.na
\fB\fBDDI_PROBE_DONTCARE\fR \fR
.ad
.RS 23n
If the probe was unsuccessful, yet \fBattach\fR(9E) should still be called.
.RE

.sp
.ne 2
.na
\fB\fBDDI_PROBE_PARTIAL\fR \fR
.ad
.RS 23n
If the instance is not present now, but may be present in the future.
.RE

.SH SEE ALSO
.sp
.LP
\fBattach\fR(9E), \fBidentify\fR(9E), \fBddi_dev_is_sid\fR(9F),
\fBddi_map_regs\fR(9F), \fBddi_peek\fR(9F), \fBddi_poke\fR(9F),
\fBnulldev\fR(9F), \fBdev_ops\fR(9S)
.sp
.LP
\fIWriting Device Drivers\fR
```

```
**********************************************************
    2623 Sat Feb 15 09:54:07 2020
new/usr/src/man/man9e/usba_hcdi_hub_update.9e
12309 errors in section 9e of the manual
**********************************************************
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\" Copyright 2016 Joyent, Inc.
.\"
.Dd February 15, 2020
.Dd Dec 20, 2016
.Dt USBA_HCDI_HUB_UPDATE 9E
.Os
.Sh NAME
.Nm usba_hcdi_hub_update
.Nd USB HCD hub update entry point
.Sh SYNOPSIS
.In sys/usb/usba/hcdi.h
.Ft int
.Fo prefix_hcdi_hub_update
.Fa "usba_device_t *ud"
.Fa "uint8_t nports"
.Fa "uint8_t tt"
.Fc
.Sh INTERFACE LEVEL
.Sy Volatile -
illumos USB HCD private function
.Pp
This is a private function that is not part of the stable DDI.
It may be removed or changed at any time.
.Sh PARAMETERS
.Bl -tag -width Fa
.It Fa ud
Pointer to a USB device structure being updated.
See
.Xr usba_device 9S
for more information.
.It Fa nports
The number of ports present on the hub.
.It Fa tt
The value of the Think Time property as defined in the USB
specification's hub descriptor.
.El
.Sh DESCRIPTION
The
.Fn usba_hcdi_hub_update
entry point is an optional entry point for USB host controller drivers.
It is used by some controllers to allow them to update information about
a device in the controller after a device has been determined to be a
hub during enumeration.
If a host controller does not need to take any specific action after enumerating
a hub, then it should simply set this entry point in the
.Xr usba_hcdi_ops 9S
structure to
.Dv NULL .
.Pp
The
```



```
.Fa nports
and
.Fa tt
members provide relevant information from the device's hub class
descriptor which can be used to help program the host controller.
Any programming should be performed synchronously and be completed before
this function returns.
.Pp
This function will be called after
.Xr usba_hcdi_device_init 9E
has been called.
Any private data registered with that function will be available.
.Pp
If this function fails, the enumeration of this device will fail, the
hub driver will not attach to this USB device, and all devices plugged
into this hub will not be detected by the system.
.Sh CONTEXT
.Sy This function is called from kernel context only.
This functin is called from kernel context only.
.Sh RETURN VALUES
Upon successful completion, the
.Fn usba_hcdi_hub_update
function should return
.Sy USB_SUCCESS .
Otherwise, it should return the appropriate USB error.
If uncertain, use
.Sy USB_FAILURE .
.Sh SEE ALSO
.Xr usba_hcdi_device_init 9E ,
.Xr usba_device 9S ,
.Xr usba_hcdi_ops 9S
```

```
**********************************************************
    5117 Sat Feb 15 09:54:07 2020
new/usr/src/man/man9e/usba_hcdi_pipe_open.9e
12309 errors in section 9e of the manual
**********************************************************
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\"
.\" Copyright 2016 Joyent, Inc.
.\"
.Dd February 15, 2020
.Dd Nov 26, 2017
.Dt USBA_HCDI_PIPE_OPEN 9E
.Os
.Sh NAME
.Nm usba_hcdi_pipe_open ,
.Nm usba_hcdi_pipe_close
.Nd open and close a USB pipe
.Sh SYNOPSIS
.In sys/usb/usba/hcdi.h
.Ft int
.Fo prefix_hcdi_pipe_open
.Fa "usba_pipe_handle_data_t *ph"
.Fa "usb_flags_t usb_flags"
.Fc
.Ft int
.Fo prefix_hcdi_pipe_close
.Fa "usba_pipe_handle_data_t *ph"
.Fa "usb_flags_t usb_flags"
.Fc
.Sh INTERFACE LEVEL
.Sy Volatile -
illumos USB HCD private function
.Pp
This is a private function that is not part of the stable DDI.
It may be removed or changed at any time.
.Sh PARAMETERS
.Bl -tag -width Fa
.It Fa ph
A pointer to a USB pipe handle as defined in
.Xr usba_pipe_handle_data 9S .
.It Fa usb_flags
Flags which describe how allocations should be performed.
Valid flags are:
.Bl -tag -width Sy
.It Sy USB_FLAGS_NOSLEEP
Do not block waiting for memory.
If memory is not available the allocation will fail.
.It Sy USB_FLAGS_SLEEP
Perform a blocking allocation.
If memory is not available, the function will wait until memory is made
available.
.Pp
Note, the request may still fail even if
.Sy USB_FLAGS_SLEEP
is specified.
.El
.El
```



```
.Sh DESCRIPTION
The
.Fn usba_hcdi_pipe_open
and
.Fn usba_hcdi_pipe_close
entry points are called by the USB framework whenever a client, or the
framework itself, need to open or close a specific pipe.
For additional background see
.Xr usba_hcdi 9E .
.Pp
When a pipe is opened, the host controller driver is responsible for
preparing the specified endpoint for performing transfers.
This may include allocating bandwidth, programming the controller, and more.
When the pipe is closed, the host controller driver is responsible for
cleaning up any resources that were allocated during the open call.
.Pp
The pipe handle,
.Fa ph ,
identifies the endpoint that it the USBA is trying to open or close
through its endpoint descriptor in the
.Sy p_ep
member.
The endpoint descriptor is described in
.Xr usb_ep_descr 9S .
From the endpoint descriptor the driver can determine the type of
endpoint, what the address of the endpoint is, and what direction the
endpoint is in.
When combined, these uniquely describe the pipe.
.Pp
To open a pipe, the driver may need additional companion endpoint
descriptors.
If these are available, they will be in the
.Sy p_xep
member of the pipe handle.
See
.Xr usb_ep_xdescr 9S
for more information on how to determine which descriptors are present
and get the information encoded in them.
.Pp
Host controller drivers should check the USB address of the
USB device that
.Fa ph
belongs to.
The driver may be asked to open a pipe to the root hub.
As the root hub is often synthetic, the driver may need to take a different
path than normal.
.Ss Pipe open specifics
A given endpoint on a device can only be opened once.
If there’s a request to open an already open endpoint, then the request to open
the pipe should be failed.
.Pp
By the time the call to open a pipe returns, the driver should expect
that any of the pipe transfer or reset entry points will be called on
the pipe.
.Pp
A driver can establish private data on an endpoint.
During pipe open it may set the
.Sy p_hcd_private
member to any value.
Generally this points to an allocated structure that contains data specific to
the host controller.
This value will remain on the pipe handle.
It is the responsibility of the driver to clear the data when the pipe is
closed.
.Ss Pipe close specifics
When a pipe is closed, the driver must clean up all of the resources
```

```
127 that it allocated when opening the pipe.
128 For non-periodic transfers, the host controller driver may assume that there
128 For non-periodic transfers, the host controller driver may assueme that there
129 are no outstanding transfers that need to be cleaned up.
130 However, the same is not true for periodic pipes.
131 .Pp
132 For pipes that have outstanding periodic transfers, the host controller
133 driver needs to clean them up and quiesce them as though a call to
134 either
135 .Xr usba_hcdi_pipe_stop_intr_polling 9E
136 or
137 .Xr usba_hcdi_pipe_stop_isoc_polling 9E
138 had been called.
139 .Pp
140 Just as with opening the pipe, the driver should pay attention to the
141 address of the USB device, as it may be the root hub, which may be a
142 synthetic pipe.
143 .Pp
144 When a call to
145 .Fn usba_hcdi_pipe_close
146 completes, the device should be in a state that the pipe can be opened
147 again.
148 .Sh RETURN VALUES
149 Upon successful completion, the
150 .Fn usba_hcdi_pipe_open
151 and
152 .Fn uba_hcdi_pipe_close
153 functions should return
154 .Sy USB_SUCCESS .
155 Otherwise, it should return the appropriate USB error.
156 If uncertain, use
157 .Sy USB_FAILURE .
158 .Sh SEE ALSO
159 .Xr usba_hcdi 9E ,
160 .Xr usba_hcdi_pipe_stop_intr_polling 9E ,
161 .Xr usba_hcdi_pipe_stop_isoc_polling 9E ,
162 .Xr usb_ep_descr 9S ,
163 .Xr usb_ep_xdescr 9S ,
164 .Xr usba_pipe_handle_data 9S
```

```
**********************************************************
    8807 Sat Feb 15 09:54:07 2020
new/usr/src/man/man9f/gld.9f
12309 errors in section 9e of the manual
**********************************************************
'\" te
.\" Copyright (c) 2003, Sun Microsystems, Inc.
.\" All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH GLD 9F "February 15, 2020"
.TH GLD 9F "Aug 28, 2003"
.SH NAME
gld, gld_mac_alloc, gld_mac_free, gld_register, gld_unregister, gld_recv,
gld_sched, gld_intr \- Generic LAN Driver service routines
.SH SYNOPSIS
.LP
.nf
#include <sys/gld.h>

\fBgld_mac_info_t *\fR\fBgld_mac_alloc\fR(\fBdev_info_t *\fR\fIdip\fR);
.fi

.LP
.nf
\fBvoid\fR \fBgld_mac_free\fR(\fBgld_mac_info_t *\fR\fImacinfo\fR);
.fi

.LP
.nf
\fBint\fR \fBgld_register\fR(\fBdev_info_t *\fR\fIdip\fR, \fBchar *\fR\fIname\fR
.fi

.LP
.nf
\fBint\fR \fBgld_unregister\fR(\fBgld_mac_info_t *\fR\fImacinfo\fR);
.fi

.LP
.nf
\fBvoid\fR \fBgld_recv\fR(\fBgld_mac_info_t *\fR\fImacinfo\fR, \fBmblk_t *\fR\fI
.fi

.LP
.nf
\fBvoid\fR \fBgld_sched\fR(\fBgld_mac_info_t *\fR\fImacinfo\fR);
.fi

.LP
.nf
\fBuint_t\fR \fBgld_intr\fR(\fBcaddr_t);\fR
.fi

.LP
.nf
\fBvoid\fR \fBgld_linkstate\fR(\fBgld_mac_info_t *\fR\fImacinfo\fR, \fBint32_t\f
.fi

.SH INTERFACE LEVEL
.sp
.LP
Solaris architecture specific (Solaris DDI).
.SH PARAMETERS
.sp
.ne 2
```



```
.na
\fB\fImacinfo\fR \fR
.ad
.RS 13n
Pointer to a \fBgld_mac_info\fR(9S) structure.
.RE

.sp
.ne 2
.na
\fB\fIdip\fR \fR
.ad
.RS 13n
Pointer to \fBdev_info\fR structure.
.RE

.sp
.ne 2
.na
\fB\fIname\fR \fR
.ad
.RS 13n
Device interface name.
.RE

.sp
.ne 2
.na
\fB\fImp\fR \fR
.ad
.RS 13n
Pointer to a message block containing a received packet.
.RE

.sp
.ne 2
.na
\fB\fInewstate\fR \fR
.ad
.RS 13n
Media link state.
.RE

.SH DESCRIPTION
.sp
.LP
\fBgld_mac_alloc\fR(\|) allocates a new \fBgld_mac_info\fR(9S) structure and
returns a pointer to it. Some of the GLD-private elements of the structure may
be initialized before \fBgld_mac_alloc\fR(\|) returns; all other elements are
initialized to zero. The device driver must initialize some structure members,
as described in \fBgld_mac_info\fR(9S), before passing the mac_info pointer to
\fBgld_register\fR(\|).
.sp
.LP
\fBgld_mac_free\fR(\|) frees a \fBgld_mac_info\fR(9S) structure previously
allocated by \fBgld_mac_alloc\fR(\|).
.sp
.LP
\fBgld_register\fR(\|) is called from the device driver's \fBattach\fR(9E)
routine, and is used to link the GLD-based device driver with the GLD
framework. Before calling \fBgld_register\fR(\|) the device driver's
\fBattach\fR(9E) routine must first use \fBgld_mac_alloc\fR(\|) to allocate a
\fBgld_mac_info\fR(9S) structure, and initialize several of its structure
elements. See \fBgld_mac_info\fR(9S) for more information. A successful call to
\fBgld_register\fR(\|) performs the following actions:
.RS +4
```

```
.TP
.ie t \(bu
.el o
links the device-specific driver with the GLD system;
.RE
.RS +4
.TP
.ie t \(bu
.el o
sets the device-specific driver's private data pointer (using
\fBddi_set_driver_private\fR(9F)) to point to the \fBmacinfo\fR structure;
.RE
.RS +4
.TP
.ie t \(bu
.el o
creates the minor device node.
.RE
.sp
.LP
The device interface name passed to \fBgld_register\fR(\|) must exactly match
the name of the driver module as it exists in the filesystem.
.sp
.LP
The driver's \fBattach\fR(9E) routine should return \fBDDI_SUCCESS\fR if
\fBgld_register\fR(\|) succeeds. If \fBgld_register\fR(\|) returns
\fBDDI_FAILURE\fR, the \fBattach\fR(9E) routine should deallocate any resources
it allocated before calling \fBgld_register\fR(\|) and then also return
\fBDDI_FAILURE\fR.
.sp
.LP
\fBgld_unregister\fR(\|) is called by the device driver's \fBdetach\fR(9E)
function, and if successful, performs the following tasks:
.RS +4
.TP
.ie t \(bu
.el o
ensures the device's interrupts are stopped, calling the driver's
\fBgldm_stop\fR(\|) routine if necessary;
.RE
.RS +4
.TP
.ie t \(bu
.el o
removes the minor device node;
.RE
.RS +4
.TP
.ie t \(bu
.el o
unlinks the device-specific driver from the GLD system.
.RE
.sp
.LP
If \fBgld_unregister\fR(\|) returns \fBDDI_SUCCESS\fR, the \fBdetach\fR(9E)
routine should deallocate any data structures allocated in the \fBattach\fR(9E)
routine, using \fBgld_mac_free\fR(\|) to deallocate the \fBmacinfo\fR
structure, and return \fBDDI_SUCCESS\fR. If \fBgld_unregister\fR(\|) returns
\fBDDI_FAILURE\fR, the driver's \fBdetach\fR(9E) routine must leave the device
operational and return \fBDDI_FAILURE\fR.
.sp
.LP
\fBgld_recv\fR(\|) is called by the driver's interrupt handler to pass a
received packet upstream. The driver must construct and pass a STREAMS
\fBM_DATA\fR message containing the raw packet. \fBgld_recv\fR(\|) determines
which STREAMS queues, if any, should receive a copy of the packet, duplicating
```



```
it if necessary. It then formats a \fBDL_UNITDATA_IND\fR message, if required,
and passes the data up all appropriate streams.
.sp
.LP
The driver should avoid holding mutex or other locks during the call to
\fBgld_recv\fR(\|). In particular, locks that could be taken by a transmit
thread may not be held during a call to \fBgld_recv\fR(\|): the interrupt
thread that calls \fBgld_recv\fR(\|) may in some cases carry out processing
that includes sending an outgoing packet, resulting in a call to the driver's
\fBgldm_send\fR(\|) routine. If the \fBgldm_send\fR(\|) routine were to try to
acquire a mutex being held by the \fBgldm_intr\fR(\|) routine at the time it
calls \fBgld_recv\fR(\|), this could result in a panic due to recursive mutex
entry.
.sp
.LP
\fBgld_sched\fR(\|) is called by the device driver to reschedule stalled
outbound packets. Whenever the driver's \fBgldm_send\fR(\|) routine has
returned \fBGLD_NORESOURCES\fR, the driver must later call \fBgld_sched\fR(\|)
to inform the GLD framework that it should retry the packets that previously
could not be sent. \fBgld_sched\fR(\|) should be called as soon as possible
after resources are again available, to ensure that GLD resumes passing
outbound packets to the driver's \fBgldm_send\fR(\|) routine in a timely way.
(If the driver's \fBgldm_stop\fR(\|) routine is called, the driver is absolved
from this obligation until it later again returns \fBGLD_NORESOURCES\fR from
its \fBgldm_send\fR(\|) routine; however, extra calls to \fBgld_sched\fR(\|)
will not cause incorrect operation.)
.sp
.LP
\fBgld_intr\fR(\|) is GLD's main interrupt handler. Normally it is specified as
the interrupt routine in the device driver's call to \fBddi_add_intr\fR(9F).
The argument to the interrupt handler (specified as \fIint_handler_arg\fR in
the call to \fBddi_add_intr\fR(9F)) must be a pointer to the
\fBgld_mac_info\fR(9S) structure. \fBgld_intr\fR(\|) will, when appropriate,
call the device driver's \fBgldm_intr\fR(\|) function, passing that pointer to
the \fBgld_mac_info\fR(9S) structure. However, if the driver uses a high-level
interrupt, it must provide its own high-level interrupt handler, and trigger a
soft interrupt from within that. In this case, \fBgld_intr\fR(\|) may be
specified as the soft interrupt handler in the call to
\fBddi_add_softintr\fR(\|).
.sp
.LP
\fBgld_linkstate()\fR is called by the device driver to notify GLD of changes
in the media link state. The newstate argument should be set to one of the
following:
.sp
.ne 2
.na
\fB\fBGLD_LINKSTATE_DOWN\fR \fR
.ad
.RS 26n
The media link is unavailable.
.RE

.sp
.ne 2
.na
\fB\fBGLD_LINKSTATE_UP\fR \fR
.ad
.RS 26n
The media link is unavailable.
.RE

.sp
.ne 2
.na
\fB\fBGLD_LINKSTATE_UNKNOWN\fR \fR
```

```
.ad
.RS 26n
The status of the media link is unknown.
.RE

.sp
.LP
If a driver calls \fBgld_linkstate()\fR, it must also set the GLD_CAP_LINKSTATE
bit in the gldm_capabilities field of the \fBgld_mac_info\fR(9S) structure.
bit  in the  gldm_capabilties field of the \fBgld_mac_info\fR(9S) structure.
.SH RETURN VALUES
.sp
.LP
\fBgld_mac_alloc\fR(\|) returns a pointer to a new \fBgld_mac_info\fR(9S)
structure.
.sp
.LP
\fBgld_register\fR(\|) and \fBgld_unregister\fR(\|) return:
.sp
.ne 2
.na
\fB\fBDDI_SUCCESS\fR \fR
.ad
.RS 16n
on success.
.RE

.sp
.ne 2
.na
\fB\fBDDI_FAILURE\fR \fR
.ad
.RS 16n
on failure.
.RE

.sp
.LP
\fBgld_intr\fR(\|) returns a value appropriate for an interrupt handler.
.SH SEE ALSO
.sp
.LP
\fBgld\fR(7D), \fBgld\fR(9E), \fBgld_mac_info\fR(9S), \fBgld_stats\fR(9S),
\fBdlpi\fR(7P), \fBattach\fR(9E), \fBddi_add_intr\fR(9F)
.sp
.LP
\fIWriting Device Drivers\fR
```

```
**********************************************************
    6797 Sat Feb 15 09:54:07 2020
new/usr/src/man/man9f/mac_register.9f
12309 errors in section 9e of the manual
**********************************************************
```

```
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\"
.\" Copyright (c) 2017, Joyent, Inc.
.\"
.Dd February 15, 2020
.Dd September 22, 2017
.Dt MAC_REGISTER 9F
.Os
.Sh NAME
.Nm mac_register ,
.Nm mac_unregister
.Nd register and unregister a device driver from the MAC framework
.Sh SYNOPSIS
.In sys/mac_provider.h
.Ft int
.Fo mac_register
.Fa "mac_register_t *mregp"
.Fa "mac_handle_t *mhp"
.Fc
.Ft int
.Fo mac_unregister
.Fa "mac_handle_t mh"
.Fc
.Sh INTERFACE LEVEL
illumos DDI specific
.Sh PARAMETERS
.Bl -tag -width Fa
.It Fa mregp
A pointer to a
.Xr mac_register 9S
structure allocated by calling
.Xr mac_alloc 9F
and filled in by the device driver.
.It Fa mhp
A pointer to a driver-backed handle to the MAC framework.
.It Fa mh
The driver-backed handle to the MAC framework.
.El
.Sh DESCRIPTION
The
.Fn mac_register
function is used to register an instance of a device driver with the
.Xr mac 9E
framework.
Upon successfully calling the
.Fn mac_register
function, the device will start having its
.Xr mac_callbacks 9S
entry points called.
The device driver should call this function during it's
.Xr attach 9E
entry point after the device has been configured and is set up.
```



```
For a more detailed explanation of the exact steps that the device driver
should take and where in the sequence of a driver's
.Xr attach 9E
entry point this function should be called, see the
.Sx Registering with MAC
section of
.Xr mac 9E .
.Pp
The driver should provide a pointer to a
.Ft mac_handle_t
structure as the second argument to the
.Fn mac_register
function.
This handle will be used when the device driver needs to interact with the
framework in various ways throughout its life.
It is also where the driver gets the
.Fa mh
argument for calling the
.Fn mac_unregister
function.
It is recommended that the device driver keep the handle around in its soft
state structure for a given instance.
.Pp
If the call to the
.Fn mac_register
function fails, the device driver should unwind its
.Xr attach 9E
entry point, tear down everything that it initialized, and ultimately
return an error from its
.Xr attach 9E
entry point.
.Pp
If the
.Xr attach 9E
routine fails for some reason after the call to the
.Fn mac_register
function has succeeded, then the driver should call the
.Fn mac_unregister
function as part of unwinding all of its state.
.Pp
When a driver is in its
.Xr detach 9E
entry point, it should call the
.Fn mac_unregister
function immediately after draining any of its transmit and receive
resources that might have been given to the rest of the operating system
through DMA binding.
See the
.Sx MBLKS AND DMA
section of
.Xr mac 9E
for more information.
This should be done before the driver does any tearing down.
The call to the
.Fn mac_unregister
function may fail.
This may happen because the networking stack is still using the device.
In such a case, the driver should fail the call to
.Xr detach 9E
and return
.Sy DDI_FAILURE .
.Sh CONTEXT
The
.Fn mac_register
function is generally only called from a driver's
.Xr attach 9E
```

```
127 entry point.
128 The
129 .Fn mac_unregister
130 function is generally only called from a driver's
131 .Xr attach 9E
132 and
133 .Xr detach 9E
134 entry point.
135 However, both functions may be called from either
136 .Sy user
137 or
138 .Sy kernel
139 context.
140 .Sh RETURN VALUES
141 Upon successful completion, the
142 .Fn mac_register
143 and
144 .Fn mac_unregister
145 functions both return
146 .Sy 0 .
147 Otherwise, they return an error number.
148 .Sh EXAMPLES
149 The following example shows how a device driver might call the
150 .Fn mac_register
151 function.
152 .Bd -literal
153 #include <sys/mac_provider.h>
154 #include <sys/mac_ether.h>

156 /*
157  * The call to mac_register(9F) generally comes from the context of
158  * attach(9E). This function encapsulates setting up and initializing
159  * the mac_register_t structure and should be assumed to be called from
160  * attach.
161  *
162  * The exact set of callbacks and private properties will vary based
163  * upon the driver.
164  */

166 static char *example_priv_props[] = {
167         "_rx_intr_throttle",
168         "_tx_intr_throttle",
169         NULL
170 };

172 static mac_callbacks_t example_m_callbacks = {
173         .mc_callbacks = MC_GETCAPAB | MC_SETPROP | MC_GETPROP | MC_PROPINFO |
173         .mc_callbacsk = MC_GETCAPAB | MC_SETPROP | MC_GETPROP | MC_PROPINFO |
174             MC_IOCTL,
175         .mc_start = example_m_start,
176         .mc_stop = example_m_stop,
177         .mc_setpromisc = example_m_setpromisc,
178         .mc_multicst = example_m_multicst,
179         .mc_unicst = example_m_unicst,
180         .mc_tx = example_m_tx,
181         .mc_ioctl = example_m_ioctl,
182         .mc_getcapab = example_m_getcapab,
183         .mc_getprop = example_m_getprop,
184         .mc_setprop = example_m_setprop,
185         .mc_propinfo = example_m_propinfo
186 };

188 static boolean_t
189 example_register_mac(example_t *ep)
190 {
191         int status;
```



```
192         mac_register_t *mac;

194         mac = mac_alloc(MAC_VERSION);
195         if (mac == NULL)
196                 return (B_FALSE);

198         mac->m_type_ident = MAC_PLUGIN_IDENT_ETHER;
199         mac->m_driver = ep;
200         mac->m_dip = ep->ep_dev_info;
201         mac->m_src_addr = ep->ep_mac_addr;
202         mac->m_callbacks = &example_m_callbacks;
203         mac->m_min_sdu = 0;
204         mac->m_max_sdu = ep->ep_sdu;
205         mac->m_margin = VLAN_TAGSZ;
206         mac->m_priv_props = example_priv_props;
206         mac->m_priv_props = exmple_priv_props;

208         status = mac_register(mac, &ep->ep_mac_hdl);
209         mac_free(mac);

211         return (status == 0);
212 }
```

_____*unchanged_portion_omitted_*