NAME

MAC_CAPAB_TRANSCEIVER, **mac_transceiver_kind**, **mac_capab_transciever**, **mct_info**, **mct_read -** MAC capability for networking transceivers

SYNOPSIS

#include <sys/mac_provider.h>

typedef struct mac_capab_transceiver mac_capab_transceiver_t;
typedef enum mac_transceiver_kind mac_transceiver_kind_t;

int
mct_info(*void *driver, uint_t id, mac_transceiver_info_t *infop*);

int
mct_read(*void *driver, uint_t id, uint_t page, void *buf, size_t nbytes, off_t offset, size_t *nwritten*);

INTERFACE LEVEL

Evolving - This interface is evolving still in illumos. API and ABI stability is not guaranteed.

PARAMETERS

driver A pointer to the driver's private data that was passed in via the **m_pdata** member of the mac_register(9S) structure to the mac_register(9F) function.

id An integer value indicating which transceiver is being inquired about.

infop An opaque structure which is used to set information about the transceiver.

page A value that indicates which page from the i2c bus is being requested.

buf A pointer to which data should be written to when reading from the device.

nbytes A value indicating the number of bytes being asked to read into *buf*.

offset A value indicating the offset into the page to start reading data.

nwritten A value to be updated by the driver with the number of successfully read bytes.

DESCRIPTION

The **MAC_CAPAB_TRANSCEIVER** capability allows for GLDv3 networking device drivers to provide information to the system about their transceiver. Implementing this capability is optional. For

more information on how to handle capabilities and how to indicate that a capability is not supported, see mc_getcapab(9E).

This capability should be implemented if the device in question supports a Small Form Factor (SFF) transceiver. These are more commonly known by names such as SFP, SFP+, SFP28, QSFP+, and QSFP28. This interface does not apply to traditional copper Ethernet phys. These transceivers provide standardized information over the i2c bus at specific pages.

Supported Standards

INF-8074

The **INF-8084** standard was the original multisource agreement (MSA) for SFP devices. It proposed the original series of management pages at i2c page 0xa0. This page contained up to 512 bytes, however, only the first 96 bytes are standardized. The remaining bytes are The management page was subsequently adopted by SFP+ devices.

SFF-8472

The **SFF-8472** standard extended the original SFP MSA. This standard added a second i2c page at 0xa2, while maintaining the original page at 0xa0. The page at 0xa0 is now explicitly 256 bytes. The page at 0xa2 is also 256 bytes. This standard was also adopted for all SFP28 parts, which are commonly used in transceivers for 25 Gb/s Ethernet.

SFF-8436

The **SFF-8436** standard was developed for QSFP+ transceivers, which involve the bonding of 4 SFP+ links. QSFP+ is commonly used in the transceivers for 40 Gb/s Ethernet. This standard uses i2c page 0x00 for read-only identification purposes. The lower half of the page is used for control, while the upper 128 bytes is similar to the **INF-8084** and **SFF-8472** standards.

SFF-8636

The **SFF-8636** standard is a common management standard which is shared between both SAS and QSFP+ 28 Gb/s transceivers. The latter transceiver is commonly found in 100 Gb/s Ethernet. The transceiver's memory map is similar to that found in the **SFF-8436** specification. The identification information is found in the upper 128 bytes of page 0x00, while the lower part of the page is used for control, among other purposes.

The following table summarizes the above information.

Standard	*Speeds*	*Size*	*i2c pages*
INF-8074	1 Gb/s, 10 Gb/s	256 bytes	0xa0
SFF-8472	1 Gb/s, 10 Gb/s, 25 GB/s	512 bytes	0xa0, 0xa2
SFF-8436	40 Gb/s	256 bytes	0x00

SFF-8636	100 Gb/s	256 bytes	0x00

MAC Capability Structure

When the device driver's mc_getcapab(9E) function entry point is called with the capability requested set to **MAC_CAPAB_TRANSCEIVER**, then the value of the capability structure is the following structure:

```
typedef struct mac_capab_transceiver {
        uint_t      mct_flags;
        uint_t      mct_ntransceiveres;
        int    (*mct_info)(void *driver, uint_t id,
                    mac_transceiver_info_t *infop),
        int    (*mct_read)(void *driver, uint_t id, uint_t page,
                    void *buf, size_t nbytes, off_t offset,
                    size_t *nwritten)
} mac_capab_transceiver_t;
```

If the device driver supports the **MAC_CAPAB_TRANSCEIVER** capability, it should fill in this structure, based on the following rules:

mct_flags

The *mct_flags* member is used to negotiate extensions with the driver. MAC will set the value of *mct_flags* to include all of the currently known extensions. The driver should intersect this list with the set that they actually support. At this time, no such features are defined and the driver should set the member to **0**.

mct_ntransceivers

The value of **mct_ntransceivers** indicates that the number of transceivers present in the device. For most devices, it is expected that this value will be set to one. However, some devices do support multiple transceivers and PHYs that show up behind a single logical MAC.

It is expected that this value will not change across the lifetime of the device being attached. It is important to remember that this represents the total possible number of transceivers in the device, not how many are currently present and powered on.

The number of transceivers will influence the *id* argument used in the **mct_info**() and **mct_read**() entry points. The transceiver IDs will start at zero and go to the value of *mct_ntransceivers - 1*. It is up to the driver to keep the mapping between actual transceivers and the transceiver identifiers consistent.

mct_info

The **mct_info**() entry point is used to set basic information about the transceiver. This entry point is *required*. If the device driver cannot implement this entry point, then it should not indicate that it supports the capability.

The **mct_info**() entry point should fill in information about the transceiver with an identifier of *id*. See the description above of **mct_ntransceivers** for more information on how the IDs are determined.

The driver should then proceed to fill in basic information by calling the functions described in the section *Information Functions*. After successfully calling all of the functions, the driver should return **0**. Othewrise, it should return the appropriate error number. For a full list of error numbers, see Intro(2). Common values are:

EINVAL	The transceiver identifier *id* was invalid.
ENOTSUP	This instance of the devices does not support a transceiver. For example, a device which sometimes has copper PHYsand therefore this instance does not have any PHYs.
EIO	An error occurred while trying to read device registers. For example, an FM-aware device had an error.

mct_read

The **mct_read**() function is used to read information from a transceiver's i2c bus. The **mct_read**() entry point is an *optional* entry point.

The transceiver should first check the value of *id*, which indicates which transceiver information is being requested. See the description above of **mct_ntransceivers** for more information on how the IDs are determined.

The driver should try to read up to *nbytes* of data from the i2c bus at page *page*. The driver should start reading at offset *offset*. Finally, it should update the value in *nwritten* with the number of bytes written to the buffer *buf*.

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.

Upon successful completion, the driver should ensure that *nwritten* has been updated and then

return **0**. Otherwise, the driver should return the appropriate error number. For a full list of error numbers, see Intro(2). Common values are:

EINVAL — The value of *id* represented an invalid transceiver identifier. The transceiver i2c page *page* is not valid for this type of device. The value of *offset* is beyond the range supported for this *page*.

EIO — An error occurred while trying to read the device i2c pages.

Transceiver Information Functions

The **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 three different properties that a driver can set:

1. The kind of the transceiver.
2. Whether the transceiver is present.
3. Whether the transceiver is usable.

To set the transceiver kind, the driver should call mac_transceiver_info_set_type(9F). The transceiver's kind comes from the following enumeration of values from the enumerator mac_transceiver_kind_t:

MAC_TRANSCEIVER_UNKNOWN
: The transceiver's kind is unknown.

MAC_TRANSCEIVER_SYNTHETIC
: The device firmware does not allow direct access to the transceiver and instead abstracts it.

MAC_TRANSCEIVER_INF_8074
: The transceiver is compliant with the **INF-8074** MSA and implements the i2c informational page 0xa0.

MAC_TRANSCEIVER_SFF_8472
: The transceiver conforms to the **INF-8472** standard and implements the i2c informational page 0xa0 and 0xa2.

MAC_TRANSCEIVER_SFF_8436

The transceiver conforms to the **SFF-8436** standard and implements the i2c informational page 0x00.

MAC_TRANSCEIVER_SFF_8636
The transceiver conforms to the **SFF-8636** standard and implements the i2c informational page 0x00.

To set whether or not the transceiver is present, the driver should call 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 B_TRUE.

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 mac_transceiver_info_set_usable(9F). If the transceiver is not present, then the driver should not call this function.

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 **mct_info**() entry point and set the transceiver kind to MAC_TRANSCEIVER_SYNTHETIC.

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.

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 *mct_read* entry point.

CONTEXT

The various callback functions will be called from **kernel** context. These functions will never be called from **interrupt** context.

SEE ALSO

Intro(2), mac(9E), mc_getcapab(9E), mac_register(9F), mac_transceiver_info_set_present(9F), mac_transceiver_info_set_type(9F), mac_transceiver_info_set_usable(9F), mac_register(9S),

SFP (Small Formfactor Pluggable) Interface, INF-8074i, SFF Committee, May 12, 2001, Revision 1.0.

Diagnostic Monitoring Interface for Optical Transceivers, SFF-8472, November 21, 2014, Revision 12.2.

QSFP+ 10 Gbs 4X PLUGGABLE TRANSCEIVER, SFF-8436, October 31, 2013, Revision 4.8.

Management Interface for Cabled Environments, SFF-8636, January 26, 2016, Revision 2.7.