Netlink interface for ethtool

Basic information

Netlink interface for ethtool uses generic netlink family `ethtool` (userspace application should use macros `ETHTOOL_GENL_NAME` and `ETHTOOL_GENL_VERSION` defined in `<linux/ethtool_netlink.h>` uapi header). This family does not use a specific header, all information in requests and replies is passed using netlink attributes.

The ethtool netlink interface uses extended ACK for error and warning reporting, userspace application developers are encouraged to make these messages available to user in a suitable way.

Requests can be divided into three categories: "get" (retrieving information), "set" (setting parameters) and "action" (invoking an action).

All "set" and "action" type requests require admin privileges (`CAP_NET_ADMIN` in the namespace). Most "get" type requests are allowed for anyone but there are exceptions (where the response contains sensitive information). In some cases, the request as such is allowed for anyone but unprivileged users have attributes with sensitive information (e.g. wake-on-lan password) omitted.

Conventions

Attributes which represent a boolean value usually use NLA_U8 type so that we can distinguish three states: "on", "off" and "not present" (meaning the information is not available in "get" requests or value is not to be changed in "set" requests). For these attributes, the "true" value should be passed as number 1 but any non-zero value should be understood as "true" by recipient. In the tables below, "bool" denotes NLA_U8 attributes interpreted in this way.

In the message structure descriptions below, if an attribute name is suffixed with "+", parent nest can contain multiple attributes of the same type. This implements an array of entries.

Attributes that need to be filled-in by device drivers and that are dumped to user space based on whether they are valid or not should not use zero as a valid value. This avoids the need to explicitly signal the validity of the attribute in the device driver API.

Request header

Each request or reply message contains a nested attribute with common header. Structure of this header is

`ETHTOOL_A_HEADER_DEV_INDEX`	u32	device ifindex
`ETHTOOL_A_HEADER_DEV_NAME`	string	device name
`ETHTOOL_A_HEADER_FLAGS`	u32	flags common for all requests

`ETHTOOL_A_HEADER_DEV_INDEX` and `ETHTOOL_A_HEADER_DEV_NAME` identify the device message relates to. One of them is sufficient in requests, if both are used, they must identify the same device. Some requests, e.g. global string sets, do not require device identification. Most `GET` requests also allow dump requests without device identification to query the same information for all devices providing it (each device in a separate message).

`ETHTOOL_A_HEADER_FLAGS` is a bitmap of request flags common for all request types. The interpretation of these flags is the same for all request types but the flags may not apply to requests. Recognized flags are:

`ETHTOOL_FLAG_COMPACT_BITSETS`	use compact format bitsets in reply
`ETHTOOL_FLAG_OMIT_REPLY`	omit optional reply (_SET and _ACT)
`ETHTOOL_FLAG_STATS`	include optional device statistics

New request flags should follow the general idea that if the flag is not set, the behaviour is backward compatible, i.e. requests from old clients not aware of the flag should be interpreted the way the client expects. A client must not set flags it does not understand.

Bit sets

For short bitmaps of (reasonably) fixed length, standard `NLA_BITFIELD32` type is used. For arbitrary length bitmaps, ethtool netlink uses a nested attribute with contents of one of two forms: compact (two binary bitmaps representing bit values and mask of affected bits) and bit-by-bit (list of bits identified by either index or name).

Verbose (bit-by-bit) bitsets allow sending symbolic names for bits together with their values which saves a round trip (when the bitset is passed in a request) or at least a second request (when the bitset is in a reply). This is useful for one shot applications like traditional ethtool command. On the other hand, long running applications like ethtool monitor (displaying notifications) or network management daemons may prefer fetching the names only once and using compact form to save message size. Notifications from ethtool netlink interface always use compact form for bitsets.

A bitset can represent either a value/mask pair (`ETHTOOL_A_BITSET_NOMASK` not set) or a single bitmap (`ETHTOOL_A_BITSET_NOMASK` set). In requests modifying a bitmap, the former changes the bit set in mask to values set in value and

preserves the rest; the latter sets the bits set in the bitmap and clears the rest.

Compact form: nested (bitset) atrribute contents:

ETHTOOL_A_BITSET_NOMASK	flag	no mask, only a list
ETHTOOL_A_BITSET_SIZE	u32	number of significant bits
ETHTOOL_A_BITSET_VALUE	binary	bitmap of bit values
ETHTOOL_A_BITSET_MASK	binary	bitmap of valid bits

Value and mask must have length at least ETHTOOL_A_BITSET_SIZE bits rounded up to a multiple of 32 bits. They consist of 32-bit words in host byte order, words ordered from least significant to most significant (i.e. the same way as bitmaps are passed with ioctl interface).

For compact form, ETHTOOL_A_BITSET_SIZE and ETHTOOL_A_BITSET_VALUE are mandatory. ETHTOOL_A_BITSET_MASK attribute is mandatory if ETHTOOL_A_BITSET_NOMASK is not set (bitset represents a value/mask pair); if ETHTOOL_A_BITSET_NOMASK is not set, ETHTOOL_A_BITSET_MASK is not allowed (bitset represents a single bitmap.

Kernel bit set length may differ from userspace length if older application is used on newer kernel or vice versa. If userspace bitmap is longer, an error is issued only if the request actually tries to set values of some bits not recognized by kernel.

Bit-by-bit form: nested (bitset) attribute contents:

ETHTOOL_A_BITSET_NOMASK	flag	no mask, only a list
ETHTOOL_A_BITSET_SIZE	u32	number of significant bits
ETHTOOL_A_BITSET_BITS	nested	array of bits
ETHTOOL_A_BITSET_BITS_BIT+	nested	one bit
ETHTOOL_A_BITSET_BIT_INDEX	u32	bit index (0 for LSB)
ETHTOOL_A_BITSET_BIT_NAME	string	bit name
ETHTOOL_A_BITSET_BIT_VALUE	flag	present if bit is set

Bit size is optional for bit-by-bit form. ETHTOOL_A_BITSET_BITS nest can only contain ETHTOOL_A_BITSET_BITS_BIT attributes but there can be an arbitrary number of them. A bit may be identified by its index or by its name. When used in requests, listed bits are set to 0 or 1 according to ETHTOOL_A_BITSET_BIT_VALUE, the rest is preserved. A request fails if index exceeds kernel bit length or if name is not recognized.

When ETHTOOL_A_BITSET_NOMASK flag is present, bitset is interpreted as a simple bitmap. ETHTOOL_A_BITSET_BIT_VALUE attributes are not used in such case. Such bitset represents a bitmap with listed bits set and the rest zero.

In requests, application can use either form. Form used by kernel in reply is determined by ETHTOOL_FLAG_COMPACT_BITSETS flag in flags field of request header. Semantics of value and mask depends on the attribute.

List of message types

All constants identifying message types use ETHTOOL_CMD_ prefix and suffix according to message purpose:

_GET	userspace request to retrieve data
_SET	userspace request to set data
_ACT	userspace request to perform an action
_GET_REPLY	kernel reply to a GET request
_SET_REPLY	kernel reply to a SET request
_ACT_REPLY	kernel reply to an ACT request
_NTF	kernel notification

Userspace to kernel:

ETHTOOL_MSG_STRSET_GET	get string set
ETHTOOL_MSG_LINKINFO_GET	get link settings
ETHTOOL_MSG_LINKINFO_SET	set link settings
ETHTOOL_MSG_LINKMODES_GET	get link modes info
ETHTOOL_MSG_LINKMODES_SET	set link modes info
ETHTOOL_MSG_LINKSTATE_GET	get link state
ETHTOOL_MSG_DEBUG_GET	get debugging settings
ETHTOOL_MSG_DEBUG_SET	set debugging settings
ETHTOOL_MSG_WOL_GET	get wake-on-lan settings
ETHTOOL_MSG_WOL_SET	set wake-on-lan settings
ETHTOOL_MSG_FEATURES_GET	get device features

ETHTOOL_MSG_FEATURES_SET	set device features
ETHTOOL_MSG_PRIVFLAGS_GET	get private flags
ETHTOOL_MSG_PRIVFLAGS_SET	set private flags
ETHTOOL_MSG_RINGS_GET	get ring sizes
ETHTOOL_MSG_RINGS_SET	set ring sizes
ETHTOOL_MSG_CHANNELS_GET	get channel counts
ETHTOOL_MSG_CHANNELS_SET	set channel counts
ETHTOOL_MSG_COALESCE_GET	get coalescing parameters
ETHTOOL_MSG_COALESCE_SET	set coalescing parameters
ETHTOOL_MSG_PAUSE_GET	get pause parameters
ETHTOOL_MSG_PAUSE_SET	set pause parameters
ETHTOOL_MSG_EEE_GET	get EEE settings
ETHTOOL_MSG_EEE_SET	set EEE settings
ETHTOOL_MSG_TSINFO_GET	get timestamping info
ETHTOOL_MSG_CABLE_TEST_ACT	action start cable test
ETHTOOL_MSG_CABLE_TEST_TDR_ACT	action start raw TDR cable test
ETHTOOL_MSG_TUNNEL_INFO_GET	get tunnel offload info
ETHTOOL_MSG_FEC_GET	get FEC settings
ETHTOOL_MSG_FEC_SET	set FEC settings
ETHTOOL_MSG_MODULE_EEPROM_GET	read SFP module EEPROM
ETHTOOL_MSG_STATS_GET	get standard statistics
ETHTOOL_MSG_PHC_VCLOCKS_GET	get PHC virtual clocks info
ETHTOOL_MSG_MODULE_SET	set transceiver module parameters
ETHTOOL_MSG_MODULE_GET	get transceiver module parameters

Kernel to userspace:

ETHTOOL_MSG_STRSET_GET_REPLY	string set contents
ETHTOOL_MSG_LINKINFO_GET_REPLY	link settings
ETHTOOL_MSG_LINKINFO_NTF	link settings notification
ETHTOOL_MSG_LINKMODES_GET_REPLY	link modes info
ETHTOOL_MSG_LINKMODES_NTF	link modes notification
ETHTOOL_MSG_LINKSTATE_GET_REPLY	link state info
ETHTOOL_MSG_DEBUG_GET_REPLY	debugging settings
ETHTOOL_MSG_DEBUG_NTF	debugging settings notification
ETHTOOL_MSG_WOL_GET_REPLY	wake-on-lan settings
ETHTOOL_MSG_WOL_NTF	wake-on-lan settings notification
ETHTOOL_MSG_FEATURES_GET_REPLY	device features
ETHTOOL_MSG_FEATURES_SET_REPLY	optional reply to FEATURES_SET
ETHTOOL_MSG_FEATURES_NTF	netdev features notification
ETHTOOL_MSG_PRIVFLAGS_GET_REPLY	private flags
ETHTOOL_MSG_PRIVFLAGS_NTF	private flags
ETHTOOL_MSG_RINGS_GET_REPLY	ring sizes
ETHTOOL_MSG_RINGS_NTF	ring sizes
ETHTOOL_MSG_CHANNELS_GET_REPLY	channel counts
ETHTOOL_MSG_CHANNELS_NTF	channel counts
ETHTOOL_MSG_COALESCE_GET_REPLY	coalescing parameters
ETHTOOL_MSG_COALESCE_NTF	coalescing parameters
ETHTOOL_MSG_PAUSE_GET_REPLY	pause parameters
ETHTOOL_MSG_PAUSE_NTF	pause parameters
ETHTOOL_MSG_EEE_GET_REPLY	EEE settings
ETHTOOL_MSG_EEE_NTF	EEE settings
ETHTOOL_MSG_TSINFO_GET_REPLY	timestamping info
ETHTOOL_MSG_CABLE_TEST_NTF	Cable test results
ETHTOOL_MSG_CABLE_TEST_TDR_NTF	Cable test TDR results
ETHTOOL_MSG_TUNNEL_INFO_GET_REPLY	tunnel offload info
ETHTOOL_MSG_FEC_GET_REPLY	FEC settings
ETHTOOL_MSG_FEC_NTF	FEC settings
ETHTOOL_MSG_MODULE_EEPROM_GET_REPLY	read SFP module EEPROM
ETHTOOL_MSG_STATS_GET_REPLY	standard statistics
ETHTOOL_MSG_PHC_VCLOCKS_GET_REPLY	PHC virtual clocks info

ETHTOOL_MSG_MODULE_GET_REPLY	transceiver module parameters

GET requests are sent by userspace applications to retrieve device information. They usually do not contain any message specific attributes. Kernel replies with corresponding "GET_REPLY" message. For most types, GET request with NLM_F_DUMP and no device identification can be used to query the information for all devices supporting the request.

If the data can be also modified, corresponding SET message with the same layout as corresponding GET_REPLY is used to request changes. Only attributes where a change is requested are included in such request (also, not all attributes may be changed). Replies to most SET request consist only of error code and extack; if kernel provides additional data, it is sent in the form of corresponding SET_REPLY message which can be suppressed by setting ETHTOOL_FLAG_OMIT_REPLY flag in request header.

Data modification also triggers sending a NTF message with a notification. These usually bear only a subset of attributes which was affected by the change. The same notification is issued if the data is modified using other means (mostly ioctl ethtool interface). Unlike notifications from ethtool netlink code which are only sent if something actually changed, notifications triggered by ioctl interface may be sent even if the request did not actually change any data.

ACT messages request kernel (driver) to perform a specific action. If some information is reported by kernel (which can be suppressed by setting ETHTOOL_FLAG_OMIT_REPLY flag in request header), the reply takes form of an ACT_REPLY message. Performing an action also triggers a notification (NTF message).

Later sections describe the format and semantics of these messages.

STRSET_GET

Requests contents of a string set as provided by ioctl commands ETHTOOL_GSSET_INFO and ETHTOOL_GSTRINGS. String sets are not user writeable so that the corresponding STRSET_SET message is only used in kernel replies. There are two types of string sets: global (independent of a device, e.g. device feature names) and device specific (e.g. device private flags).

Request contents:

ETHTOOL_A_STRSET_HEADER	nested	request header
ETHTOOL_A_STRSET_STRINGSETS	nested	string set to request
ETHTOOL_A_STRINGSETS_STRINGSET+	nested	one string set
ETHTOOL_A_STRINGSET_ID	u32	set id

Kernel response contents:

ETHTOOL_A_STRSET_HEADER	nested	reply header
ETHTOOL_A_STRSET_STRINGSETS	nested	array of string sets
ETHTOOL_A_STRINGSETS_STRINGSET+	nested	one string set
ETHTOOL_A_STRINGSET_ID	u32	set id
ETHTOOL_A_STRINGSET_COUNT	u32	number of strings
ETHTOOL_A_STRINGSET_STRINGS	nested	array of strings
ETHTOOL_A_STRINGS_STRING+	nested	one string
ETHTOOL_A_STRING_INDEX	u32	string index
ETHTOOL_A_STRING_VALUE	string	string value
ETHTOOL_A_STRSET_COUNTS_ONLY	flag	return only counts

Device identification in request header is optional. Depending on its presence a and NLM_F_DUMP flag, there are three type of STRSET_GET requests:

- no NLM_F_DUMP, no device: get "global" stringsets
- no NLM_F_DUMP, with device: get string sets related to the device
- NLM_F_DUMP, no device: get device related string sets for all devices

If there is no ETHTOOL_A_STRSET_STRINGSETS array, all string sets of requested type are returned, otherwise only those specified in the request. Flag ETHTOOL_A_STRSET_COUNTS_ONLY tells kernel to only return string counts of the sets, not the actual strings.

LINKINFO_GET

Requests link settings as provided by ETHTOOL_GLINKSETTINGS except for link modes and autonegotiation related information. The request does not use any attributes.

Request contents:

ETHTOOL_A_LINKINFO_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_LINKINFO_HEADER	nested	reply header
ETHTOOL_A_LINKINFO_PORT	u8	physical port
ETHTOOL_A_LINKINFO_PHYADDR	u8	phy MDIO address
ETHTOOL_A_LINKINFO_TP_MDIX	u8	MDI(-X) status
ETHTOOL_A_LINKINFO_TP_MDIX_CTRL	u8	MDI(-X) control
ETHTOOL_A_LINKINFO_TRANSCEIVER	u8	transceiver

Attributes and their values have the same meaning as matching members of the corresponding ioctl structures.

LINKINFO_GET allows dump requests (kernel returns reply message for all devices supporting the request).

LINKINFO_SET

LINKINFO_SET request allows setting some of the attributes reported by LINKINFO_GET.

Request contents:

ETHTOOL_A_LINKINFO_HEADER	nested	request header
ETHTOOL_A_LINKINFO_PORT	u8	physical port
ETHTOOL_A_LINKINFO_PHYADDR	u8	phy MDIO address
ETHTOOL_A_LINKINFO_TP_MDIX_CTRL	u8	MDI(-X) control

MDI(-X) status and transceiver cannot be set, request with the corresponding attributes is rejected.

LINKMODES_GET

Requests link modes (supported, advertised and peer advertised) and related information (autonegotiation status, link speed and duplex) as provided by ETHTOOL_GLINKSETTINGS. The request does not use any attributes.

Request contents:

ETHTOOL_A_LINKMODES_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_LINKMODES_HEADER	nested	reply header
ETHTOOL_A_LINKMODES_AUTONEG	u8	autonegotiation status
ETHTOOL_A_LINKMODES_OURS	bitset	advertised link modes
ETHTOOL_A_LINKMODES_PEER	bitset	partner link modes
ETHTOOL_A_LINKMODES_SPEED	u32	link speed (Mb/s)
ETHTOOL_A_LINKMODES_DUPLEX	u8	duplex mode
ETHTOOL_A_LINKMODES_MASTER_SLAVE_CFG	u8	Master/slave port mode
ETHTOOL_A_LINKMODES_MASTER_SLAVE_STATE	u8	Master/slave port state

For ETHTOOL_A_LINKMODES_OURS, value represents advertised modes and mask represents supported modes. ETHTOOL_A_LINKMODES_PEER in the reply is a bit list.

LINKMODES_GET allows dump requests (kernel returns reply messages for all devices supporting the request).

LINKMODES_SET

Request contents:

ETHTOOL_A_LINKMODES_HEADER	nested	request header
ETHTOOL_A_LINKMODES_AUTONEG	u8	autonegotiation status
ETHTOOL_A_LINKMODES_OURS	bitset	advertised link modes
ETHTOOL_A_LINKMODES_PEER	bitset	partner link modes
ETHTOOL_A_LINKMODES_SPEED	u32	link speed (Mb/s)
ETHTOOL_A_LINKMODES_DUPLEX	u8	duplex mode
ETHTOOL_A_LINKMODES_MASTER_SLAVE_CFG	u8	Master/slave port mode
ETHTOOL_A_LINKMODES_LANES	u32	lanes

ETHTOOL_A_LINKMODES_OURS bit set allows setting advertised link modes. If autonegotiation is on (either set now or kept from before), advertised modes are not changed (no ETHTOOL_A_LINKMODES_OURS attribute) and at least one of speed, duplex and lanes is specified, kernel adjusts advertised modes to all supported modes matching speed, duplex, lanes or all (whatever is specified). This autoselection is done on ethtool side with ioctl interface, netlink interface is supposed to allow requesting changes without knowing what exactly kernel supports.

LINKSTATE_GET

Requests link state information. Link up/down flag (as provided by `ETHTOOL_GLINK` ioctl command) is provided. Optionally, extended state might be provided as well. In general, extended state describes reasons for why a port is down, or why it operates in some non-obvious mode. This request does not have any attributes.

Request contents:

`ETHTOOL_A_LINKSTATE_HEADER`	nested	request header

Kernel response contents:

`ETHTOOL_A_LINKSTATE_HEADER`	nested	reply header
`ETHTOOL_A_LINKSTATE_LINK`	bool	link state (up/down)
`ETHTOOL_A_LINKSTATE_SQI`	u32	Current Signal Quality Index
`ETHTOOL_A_LINKSTATE_SQI_MAX`	u32	Max support SQI value
`ETHTOOL_A_LINKSTATE_EXT_STATE`	u8	link extended state
`ETHTOOL_A_LINKSTATE_EXT_SUBSTATE`	u8	link extended substate

For most NIC drivers, the value of `ETHTOOL_A_LINKSTATE_LINK` returns carrier flag provided by `netif_carrier_ok()` but there are drivers which define their own handler.

`ETHTOOL_A_LINKSTATE_EXT_STATE` and `ETHTOOL_A_LINKSTATE_EXT_SUBSTATE` are optional values. ethtool core can provide either both `ETHTOOL_A_LINKSTATE_EXT_STATE` and `ETHTOOL_A_LINKSTATE_EXT_SUBSTATE`, or only `ETHTOOL_A_LINKSTATE_EXT_STATE`, or none of them.

`LINKSTATE_GET` allows dump requests (kernel returns reply messages for all devices supporting the request).

Link extended states:

`ETHTOOL_LINK_EXT_STATE_AUTONEG`	States relating to the autonegotiation or issues therein
`ETHTOOL_LINK_EXT_STATE_LINK_TRAINING_FAILURE`	Failure during link training
`ETHTOOL_LINK_EXT_STATE_LINK_LOGICAL_MISMATCH`	Logical mismatch in physical coding sublayer or forward error correction sublayer
`ETHTOOL_LINK_EXT_STATE_BAD_SIGNAL_INTEGRITY`	Signal integrity issues
`ETHTOOL_LINK_EXT_STATE_NO_CABLE`	No cable connected
`ETHTOOL_LINK_EXT_STATE_CABLE_ISSUE`	Failure is related to cable, e.g., unsupported cable
`ETHTOOL_LINK_EXT_STATE_EEPROM_ISSUE`	Failure is related to EEPROM, e.g., failure during reading or parsing the data
`ETHTOOL_LINK_EXT_STATE_CALIBRATION_FAILURE`	Failure during calibration algorithm
`ETHTOOL_LINK_EXT_STATE_POWER_BUDGET_EXCEEDED`	The hardware is not able to provide the power required from cable or module
`ETHTOOL_LINK_EXT_STATE_OVERHEAT`	The module is overheated
`ETHTOOL_LINK_EXT_STATE_MODULE`	Transceiver module issue

Link extended substates:

Autoneg substates:

`ETHTOOL_LINK_EXT_SUBSTATE_AN_NO_PARTNER_DETECTED`	Peer side is down
`ETHTOOL_LINK_EXT_SUBSTATE_AN_ACK_NOT_RECEIVED`	Ack not received from peer side
`ETHTOOL_LINK_EXT_SUBSTATE_AN_NEXT_PAGE_EXCHANGE_FAILED`	Next page exchange failed
`ETHTOOL_LINK_EXT_SUBSTATE_AN_NO_PARTNER_DETECTED_FORCE_MODE`	Peer side is down during force mode or there is no agreement of speed
`ETHTOOL_LINK_EXT_SUBSTATE_AN_FEC_MISMATCH_DURING_OVERRIDE`	Forward error correction modes in both sides are mismatched
`ETHTOOL_LINK_EXT_SUBSTATE_AN_NO_HCD`	No Highest Common Denominator

Link training substates:

`ETHTOOL_LINK_EXT_SUBSTATE_LT_KR_FRAME_LOCK_NOT_ACQUIRED`	Frames were not recognized, the lock failed
`ETHTOOL_LINK_EXT_SUBSTATE_LT_KR_LINK_INHIBIT_TIMEOUT`	The lock did not occur before timeout
`ETHTOOL_LINK_EXT_SUBSTATE_LT_KR_LINK_PARTNER_DID_NOT_SET_RECEIVER_READY`	Peer side did not send ready signal after training process

ETHTOOL_LINK_EXT_SUBSTATE_LT_REMOTE_FAULT	Remote side is not ready yet

Link logical mismatch substates:

ETHTOOL_LINK_EXT_SUBSTATE_LLM_PCS_DID_NOT_ACQUIRE_BLOCK_LOCK	Physical coding sublayer was not locked in first phase - block lock
ETHTOOL_LINK_EXT_SUBSTATE_LLM_PCS_DID_NOT_ACQUIRE_AM_LOCK	Physical coding sublayer was not locked in second phase - alignment markers lock
ETHTOOL_LINK_EXT_SUBSTATE_LLM_PCS_DID_NOT_GET_ALIGN_STATUS	Physical coding sublayer did not get align status
ETHTOOL_LINK_EXT_SUBSTATE_LLM_FC_FEC_IS_NOT_LOCKED	FC forward error correction is not locked
ETHTOOL_LINK_EXT_SUBSTATE_LLM_RS_FEC_IS_NOT_LOCKED	RS forward error correction is not locked

Bad signal integrity substates:

ETHTOOL_LINK_EXT_SUBSTATE_BSI_LARGE_NUMBER_OF_PHYSICAL_ERRORS	Large number of physical errors
ETHTOOL_LINK_EXT_SUBSTATE_BSI_UNSUPPORTED_RATE	The system attempted to operate the cable at a rate that is not formally supported, which led to signal integrity issues
ETHTOOL_LINK_EXT_SUBSTATE_BSI_SERDES_REFERENCE_CLOCK_LOST	The external clock signal for SerDes is too weak or unavailable.
ETHTOOL_LINK_EXT_SUBSTATE_BSI_SERDES_ALOS	The received signal for SerDes is too weak because analog loss of signal.

Cable issue substates:

ETHTOOL_LINK_EXT_SUBSTATE_CI_UNSUPPORTED_CABLE	Unsupported cable
ETHTOOL_LINK_EXT_SUBSTATE_CI_CABLE_TEST_FAILURE	Cable test failure

Transceiver module issue substates:

ETHTOOL_LINK_EXT_SUBSTATE_MODULE_CMIS_NOT_READY	The CMIS Module State Machine did not reach the ModuleReady state. For example, if the module is stuck at ModuleFault state

DEBUG_GET

Requests debugging settings of a device. At the moment, only message mask is provided.

Request contents:

ETHTOOL_A_DEBUG_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_DEBUG_HEADER	nested	reply header
ETHTOOL_A_DEBUG_MSGMASK	bitset	message mask

The message mask (ETHTOOL_A_DEBUG_MSGMASK) is equal to message level as provided by ETHTOOL_GMSGLVL and set by ETHTOOL_SMSGLVL in ioctl interface. While it is called message level there for historical reasons, most drivers and almost all newer drivers use it as a mask of enabled message classes (represented by NETIF_MSG_* constants); therefore netlink interface follows its actual use in practice.

DEBUG_GET allows dump requests (kernel returns reply messages for all devices supporting the request).

DEBUG_SET

Set or update debugging settings of a device. At the moment, only message mask is supported.

Request contents:

ETHTOOL_A_DEBUG_HEADER	nested	request header
ETHTOOL_A_DEBUG_MSGMASK	bitset	message mask

ETHTOOL_A_DEBUG_MSGMASK bit set allows setting or modifying mask of enabled debugging message types for the device.

WOL_GET

Query device wake-on-lan settings. Unlike most "GET" type requests, `ETHTOOL_MSG_WOL_GET` requires (netns) `CAP_NET_ADMIN` privileges as it (potentially) provides SecureOn(tm) password which is confidential.

Request contents:

`ETHTOOL_A_WOL_HEADER`	nested	request header

Kernel response contents:

`ETHTOOL_A_WOL_HEADER`	nested	reply header
`ETHTOOL_A_WOL_MODES`	bitset	mask of enabled WoL modes
`ETHTOOL_A_WOL_SOPASS`	binary	SecureOn(tm) password

In reply, `ETHTOOL_A_WOL_MODES` mask consists of modes supported by the device, value of modes which are enabled. `ETHTOOL_A_WOL_SOPASS` is only included in reply if `WAKE_MAGICSECURE` mode is supported.

WOL_SET

Set or update wake-on-lan settings.

Request contents:

`ETHTOOL_A_WOL_HEADER`	nested	request header
`ETHTOOL_A_WOL_MODES`	bitset	enabled WoL modes
`ETHTOOL_A_WOL_SOPASS`	binary	SecureOn(tm) password

`ETHTOOL_A_WOL_SOPASS` is only allowed for devices supporting `WAKE_MAGICSECURE` mode.

FEATURES_GET

Gets netdev features like `ETHTOOL_GFEATURES` ioctl request.

Request contents:

`ETHTOOL_A_FEATURES_HEADER`	nested	request header

Kernel response contents:

`ETHTOOL_A_FEATURES_HEADER`	nested	reply header
`ETHTOOL_A_FEATURES_HW`	bitset	dev->hw_features
`ETHTOOL_A_FEATURES_WANTED`	bitset	dev->wanted_features
`ETHTOOL_A_FEATURES_ACTIVE`	bitset	dev->features
`ETHTOOL_A_FEATURES_NOCHANGE`	bitset	NETIF_F_NEVER_CHANGE

Bitmaps in kernel response have the same meaning as bitmaps used in ioctl interference but attribute names are different (they are based on corresponding members of struct net_device). Legacy "flags" are not provided, if userspace needs them (most likely only ethtool for backward compatibility), it can calculate their values from related feature bits itself. ETHA_FEATURES_HW uses mask consisting of all features recognized by kernel (to provide all names when using verbose bitmap format), the other three use no mask (simple bit lists).

FEATURES_SET

Request to set netdev features like `ETHTOOL_SFEATURES` ioctl request.

Request contents:

`ETHTOOL_A_FEATURES_HEADER`	nested	request header
`ETHTOOL_A_FEATURES_WANTED`	bitset	requested features

Kernel response contents:

`ETHTOOL_A_FEATURES_HEADER`	nested	reply header
`ETHTOOL_A_FEATURES_WANTED`	bitset	diff wanted vs. result
`ETHTOOL_A_FEATURES_ACTIVE`	bitset	diff old vs. new active

Request constains only one bitset which can be either value/mask pair (request to change specific feature bits and leave the rest) or

only a value (request to set all features to specified set).

As request is subject to netdev_change_features() sanity checks, optional kernel reply (can be suppressed by `ETHTOOL_FLAG_OMIT_REPLY` flag in request header) informs client about the actual result. `ETHTOOL_A_FEATURES_WANTED` reports the difference between client request and actual result: mask consists of bits which differ between requested features and result (dev->features after the operation), value consists of values of these bits in the request (i.e. negated values from resulting features). `ETHTOOL_A_FEATURES_ACTIVE` reports the difference between old and new dev->features: mask consists of bits which have changed, values are their values in new dev->features (after the operation).

`ETHTOOL_MSG_FEATURES_NTF` notification is sent not only if device features are modified using `ETHTOOL_MSG_FEATURES_SET` request or on of ethtool ioctl request but also each time features are modified with netdev_update_features() or netdev_change_features().

PRIVFLAGS_GET

Gets private flags like `ETHTOOL_GPFLAGS` ioctl request.

Request contents:

`ETHTOOL_A_PRIVFLAGS_HEADER`	nested	request header

Kernel response contents:

`ETHTOOL_A_PRIVFLAGS_HEADER`	nested	reply header
`ETHTOOL_A_PRIVFLAGS_FLAGS`	bitset	private flags

`ETHTOOL_A_PRIVFLAGS_FLAGS` is a bitset with values of device private flags. These flags are defined by driver, their number and names (and also meaning) are device dependent. For compact bitset format, names can be retrieved as `ETH_SS_PRIV_FLAGS` string set. If verbose bitset format is requested, response uses all private flags supported by the device as mask so that client gets the full information without having to fetch the string set with names.

PRIVFLAGS_SET

Sets or modifies values of device private flags like `ETHTOOL_SPFLAGS` ioctl request.

Request contents:

`ETHTOOL_A_PRIVFLAGS_HEADER`	nested	request header
`ETHTOOL_A_PRIVFLAGS_FLAGS`	bitset	private flags

`ETHTOOL_A_PRIVFLAGS_FLAGS` can either set the whole set of private flags or modify only values of some of them.

RINGS_GET

Gets ring sizes like `ETHTOOL_GRINGPARAM` ioctl request.

Request contents:

`ETHTOOL_A_RINGS_HEADER`	nested	request header

Kernel response contents:

`ETHTOOL_A_RINGS_HEADER`	nested	reply header
`ETHTOOL_A_RINGS_RX_MAX`	u32	max size of RX ring
`ETHTOOL_A_RINGS_RX_MINI_MAX`	u32	max size of RX mini ring
`ETHTOOL_A_RINGS_RX_JUMBO_MAX`	u32	max size of RX jumbo ring
`ETHTOOL_A_RINGS_TX_MAX`	u32	max size of TX ring
`ETHTOOL_A_RINGS_RX`	u32	size of RX ring
`ETHTOOL_A_RINGS_RX_MINI`	u32	size of RX mini ring
`ETHTOOL_A_RINGS_RX_JUMBO`	u32	size of RX jumbo ring
`ETHTOOL_A_RINGS_TX`	u32	size of TX ring
`ETHTOOL_A_RINGS_RX_BUF_LEN`	u32	size of buffers on the ring
`ETHTOOL_A_RINGS_TCP_DATA_SPLIT`	u8	TCP header / data split
`ETHTOOL_A_RINGS_CQE_SIZE`	u32	Size of TX/RX CQE

`ETHTOOL_A_RINGS_TCP_DATA_SPLIT` indicates whether the device is usable with page-flipping TCP zero-copy receive (`getsockopt(TCP_ZEROCOPY_RECEIVE)`). If enabled the device is configured to place frame headers and data into separate buffers. The device configuration must make it possible to receive full memory pages of data, for example because MTU is high

enough or through HW-GRO.

RINGS_SET

Sets ring sizes like `ETHTOOL_SRINGPARAM` ioctl request.

Request contents:

`ETHTOOL_A_RINGS_HEADER`	nested	reply header
`ETHTOOL_A_RINGS_RX`	u32	size of RX ring
`ETHTOOL_A_RINGS_RX_MINI`	u32	size of RX mini ring
`ETHTOOL_A_RINGS_RX_JUMBO`	u32	size of RX jumbo ring
`ETHTOOL_A_RINGS_TX`	u32	size of TX ring
`ETHTOOL_A_RINGS_RX_BUF_LEN`	u32	size of buffers on the ring
`ETHTOOL_A_RINGS_CQE_SIZE`	u32	Size of TX/RX CQE

Kernel checks that requested ring sizes do not exceed limits reported by driver. Driver may impose additional constraints and may not suspport all attributes.

`ETHTOOL_A_RINGS_CQE_SIZE` specifies the completion queue event size. Completion queue events(CQE) are the events posted by NIC to indicate the completion status of a packet when the packet is sent(like send success or error) or received(like pointers to packet fragments). The CQE size parameter enables to modify the CQE size other than default size if NIC supports it. A bigger CQE can have more receive buffer pointers inturn NIC can transfer a bigger frame from wire. Based on the NIC hardware, the overall completion queue size can be adjusted in the driver if CQE size is modified.

CHANNELS_GET

Gets channel counts like `ETHTOOL_GCHANNELS` ioctl request.

Request contents:

`ETHTOOL_A_CHANNELS_HEADER`	nested	request header

Kernel response contents:

`ETHTOOL_A_CHANNELS_HEADER`	nested	reply header
`ETHTOOL_A_CHANNELS_RX_MAX`	u32	max receive channels
`ETHTOOL_A_CHANNELS_TX_MAX`	u32	max transmit channels
`ETHTOOL_A_CHANNELS_OTHER_MAX`	u32	max other channels
`ETHTOOL_A_CHANNELS_COMBINED_MAX`	u32	max combined channels
`ETHTOOL_A_CHANNELS_RX_COUNT`	u32	receive channel count
`ETHTOOL_A_CHANNELS_TX_COUNT`	u32	transmit channel count
`ETHTOOL_A_CHANNELS_OTHER_COUNT`	u32	other channel count
`ETHTOOL_A_CHANNELS_COMBINED_COUNT`	u32	combined channel count

CHANNELS_SET

Sets channel counts like `ETHTOOL_SCHANNELS` ioctl request.

Request contents:

`ETHTOOL_A_CHANNELS_HEADER`	nested	request header
`ETHTOOL_A_CHANNELS_RX_COUNT`	u32	receive channel count
`ETHTOOL_A_CHANNELS_TX_COUNT`	u32	transmit channel count
`ETHTOOL_A_CHANNELS_OTHER_COUNT`	u32	other channel count
`ETHTOOL_A_CHANNELS_COMBINED_COUNT`	u32	combined channel count

Kernel checks that requested channel counts do not exceed limits reported by driver. Driver may impose additional constraints and may not suspport all attributes.

COALESCE_GET

Gets coalescing parameters like `ETHTOOL_GCOALESCE` ioctl request.

Request contents:

`ETHTOOL_A_COALESCE_HEADER`	nested	request header

Kernel response contents:

ETHTOOL_A_COALESCE_HEADER	nested	reply header
ETHTOOL_A_COALESCE_RX_USECS	u32	delay (us), normal Rx
ETHTOOL_A_COALESCE_RX_MAX_FRAMES	u32	max packets, normal Rx
ETHTOOL_A_COALESCE_RX_USECS_IRQ	u32	delay (us), Rx in IRQ
ETHTOOL_A_COALESCE_RX_MAX_FRAMES_IRQ	u32	max packets, Rx in IRQ
ETHTOOL_A_COALESCE_TX_USECS	u32	delay (us), normal Tx
ETHTOOL_A_COALESCE_TX_MAX_FRAMES	u32	max packets, normal Tx
ETHTOOL_A_COALESCE_TX_USECS_IRQ	u32	delay (us), Tx in IRQ
ETHTOOL_A_COALESCE_TX_MAX_FRAMES_IRQ	u32	IRQ packets, Tx in IRQ
ETHTOOL_A_COALESCE_STATS_BLOCK_USECS	u32	delay of stats update
ETHTOOL_A_COALESCE_USE_ADAPTIVE_RX	bool	adaptive Rx coalesce
ETHTOOL_A_COALESCE_USE_ADAPTIVE_TX	bool	adaptive Tx coalesce
ETHTOOL_A_COALESCE_PKT_RATE_LOW	u32	threshold for low rate
ETHTOOL_A_COALESCE_RX_USECS_LOW	u32	delay (us), low Rx
ETHTOOL_A_COALESCE_RX_MAX_FRAMES_LOW	u32	max packets, low Rx
ETHTOOL_A_COALESCE_TX_USECS_LOW	u32	delay (us), low Tx
ETHTOOL_A_COALESCE_TX_MAX_FRAMES_LOW	u32	max packets, low Tx
ETHTOOL_A_COALESCE_PKT_RATE_HIGH	u32	threshold for high rate
ETHTOOL_A_COALESCE_RX_USECS_HIGH	u32	delay (us), high Rx
ETHTOOL_A_COALESCE_RX_MAX_FRAMES_HIGH	u32	max packets, high Rx
ETHTOOL_A_COALESCE_TX_USECS_HIGH	u32	delay (us), high Tx
ETHTOOL_A_COALESCE_TX_MAX_FRAMES_HIGH	u32	max packets, high Tx
ETHTOOL_A_COALESCE_RATE_SAMPLE_INTERVAL	u32	rate sampling interval
ETHTOOL_A_COALESCE_USE_CQE_TX	bool	timer reset mode, Tx
ETHTOOL_A_COALESCE_USE_CQE_RX	bool	timer reset mode, Rx

Attributes are only included in reply if their value is not zero or the corresponding bit in `ethtool_ops::supported_coalesce_params` is set (i.e. they are declared as supported by driver).

Timer reset mode (`ETHTOOL_A_COALESCE_USE_CQE_TX` and `ETHTOOL_A_COALESCE_USE_CQE_RX`) controls the interaction between packet arrival and the various time based delay parameters. By default timers are expected to limit the max delay between any packet arrival/departure and a corresponding interrupt. In this mode timer should be started by packet arrival (sometimes delivery of previous interrupt) and reset when interrupt is delivered. Setting the appropriate attribute to 1 will enable `CQE` mode, where each packet event resets the timer. In this mode timer is used to force the interrupt if queue goes idle, while busy queues depend on the packet limit to trigger interrupts.

COALESCE_SET

Sets coalescing parameters like `ETHTOOL_SCOALESCE` ioctl request.

Request contents:

ETHTOOL_A_COALESCE_HEADER	nested	request header
ETHTOOL_A_COALESCE_RX_USECS	u32	delay (us), normal Rx
ETHTOOL_A_COALESCE_RX_MAX_FRAMES	u32	max packets, normal Rx
ETHTOOL_A_COALESCE_RX_USECS_IRQ	u32	delay (us), Rx in IRQ
ETHTOOL_A_COALESCE_RX_MAX_FRAMES_IRQ	u32	max packets, Rx in IRQ
ETHTOOL_A_COALESCE_TX_USECS	u32	delay (us), normal Tx
ETHTOOL_A_COALESCE_TX_MAX_FRAMES	u32	max packets, normal Tx
ETHTOOL_A_COALESCE_TX_USECS_IRQ	u32	delay (us), Tx in IRQ
ETHTOOL_A_COALESCE_TX_MAX_FRAMES_IRQ	u32	IRQ packets, Tx in IRQ
ETHTOOL_A_COALESCE_STATS_BLOCK_USECS	u32	delay of stats update
ETHTOOL_A_COALESCE_USE_ADAPTIVE_RX	bool	adaptive Rx coalesce
ETHTOOL_A_COALESCE_USE_ADAPTIVE_TX	bool	adaptive Tx coalesce
ETHTOOL_A_COALESCE_PKT_RATE_LOW	u32	threshold for low rate
ETHTOOL_A_COALESCE_RX_USECS_LOW	u32	delay (us), low Rx
ETHTOOL_A_COALESCE_RX_MAX_FRAMES_LOW	u32	max packets, low Rx
ETHTOOL_A_COALESCE_TX_USECS_LOW	u32	delay (us), low Tx
ETHTOOL_A_COALESCE_TX_MAX_FRAMES_LOW	u32	max packets, low Tx
ETHTOOL_A_COALESCE_PKT_RATE_HIGH	u32	threshold for high rate
ETHTOOL_A_COALESCE_RX_USECS_HIGH	u32	delay (us), high Rx
ETHTOOL_A_COALESCE_RX_MAX_FRAMES_HIGH	u32	max packets, high Rx

ETHTOOL_A_COALESCE_TX_USECS_HIGH	u32	delay (us), high Tx
ETHTOOL_A_COALESCE_TX_MAX_FRAMES_HIGH	u32	max packets, high Tx
ETHTOOL_A_COALESCE_RATE_SAMPLE_INTERVAL	u32	rate sampling interval
ETHTOOL_A_COALESCE_USE_CQE_TX	bool	timer reset mode, Tx
ETHTOOL_A_COALESCE_USE_CQE_RX	bool	timer reset mode, Rx

Request is rejected if it attributes declared as unsupported by driver (i.e. such that the corresponding bit in `ethtool_ops::supported_coalesce_params` is not set), regardless of their values. Driver may impose additional constraints on coalescing parameters and their values.

PAUSE_GET

Gets pause frame settings like `ETHTOOL_GPAUSEPARAM` ioctl request.

Request contents:

ETHTOOL_A_PAUSE_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_PAUSE_HEADER	nested	request header
ETHTOOL_A_PAUSE_AUTONEG	bool	pause autonegotiation
ETHTOOL_A_PAUSE_RX	bool	receive pause frames
ETHTOOL_A_PAUSE_TX	bool	transmit pause frames
ETHTOOL_A_PAUSE_STATS	nested	pause statistics

`ETHTOOL_A_PAUSE_STATS` are reported if `ETHTOOL_FLAG_STATS` was set in `ETHTOOL_A_HEADER_FLAGS`. It will be empty if driver did not report any statistics. Drivers fill in the statistics in the following structure:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\[linux-master][Documentation][networking]ethtool-netlink.rst`, line 1076)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/ethtool.h
    :identifiers: ethtool_pause_stats
```

Each member has a corresponding attribute defined.

PAUSE_SET

Sets pause parameters like `ETHTOOL_GPAUSEPARAM` ioctl request.

Request contents:

ETHTOOL_A_PAUSE_HEADER	nested	request header
ETHTOOL_A_PAUSE_AUTONEG	bool	pause autonegotiation
ETHTOOL_A_PAUSE_RX	bool	receive pause frames
ETHTOOL_A_PAUSE_TX	bool	transmit pause frames

EEE_GET

Gets Energy Efficient Ethernet settings like `ETHTOOL_GEEE` ioctl request.

Request contents:

ETHTOOL_A_EEE_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_EEE_HEADER	nested	request header
ETHTOOL_A_EEE_MODES_OURS	bool	supported/advertised modes
ETHTOOL_A_EEE_MODES_PEER	bool	peer advertised link modes
ETHTOOL_A_EEE_ACTIVE	bool	EEE is actively used
ETHTOOL_A_EEE_ENABLED	bool	EEE is enabled

ETHTOOL_A_EEE_TX_LPI_ENABLED	bool	Tx lpi enabled
ETHTOOL_A_EEE_TX_LPI_TIMER	u32	Tx lpi timeout (in us)

In ETHTOOL_A_EEE_MODES_OURS, mask consists of link modes for which EEE is enabled, value of link modes for which EEE is advertised. Link modes for which peer advertises EEE are listed in ETHTOOL_A_EEE_MODES_PEER (no mask). The netlink interface allows reporting EEE status for all link modes but only first 32 are provided by the ethtool_ops callback.

EEE_SET

Sets Energy Efficient Ethernet parameters like ETHTOOL_SEEE ioctl request.

Request contents:

ETHTOOL_A_EEE_HEADER	nested	request header
ETHTOOL_A_EEE_MODES_OURS	bool	advertised modes
ETHTOOL_A_EEE_ENABLED	bool	EEE is enabled
ETHTOOL_A_EEE_TX_LPI_ENABLED	bool	Tx lpi enabled
ETHTOOL_A_EEE_TX_LPI_TIMER	u32	Tx lpi timeout (in us)

ETHTOOL_A_EEE_MODES_OURS is used to either list link modes to advertise EEE for (if there is no mask) or specify changes to the list (if there is a mask). The netlink interface allows reporting EEE status for all link modes but only first 32 can be set at the moment as that is what the ethtool_ops callback supports.

TSINFO_GET

Gets timestamping information like ETHTOOL_GET_TS_INFO ioctl request.

Request contents:

ETHTOOL_A_TSINFO_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_TSINFO_HEADER	nested	request header
ETHTOOL_A_TSINFO_TIMESTAMPING	bitset	SO_TIMESTAMPING flags
ETHTOOL_A_TSINFO_TX_TYPES	bitset	supported Tx types
ETHTOOL_A_TSINFO_RX_FILTERS	bitset	supported Rx filters
ETHTOOL_A_TSINFO_PHC_INDEX	u32	PTP hw clock index

ETHTOOL_A_TSINFO_PHC_INDEX is absent if there is no associated PHC (there is no special value for this case). The bitset attributes are omitted if they would be empty (no bit set).

CABLE_TEST

Start a cable test.

Request contents:

ETHTOOL_A_CABLE_TEST_HEADER	nested	request header

Notification contents:

An Ethernet cable typically contains 1, 2 or 4 pairs. The length of the pair can only be measured when there is a fault in the pair and hence a reflection. Information about the fault may not be available, depending on the specific hardware. Hence the contents of the notify message are mostly optional. The attributes can be repeated an arbitrary number of times, in an arbitrary order, for an arbitrary number of pairs.

The example shows the notification sent when the test is completed for a T2 cable, i.e. two pairs. One pair is OK and hence has no length information. The second pair has a fault and does have length information.

ETHTOOL_A_CABLE_TEST_HEADER	nested	reply header
ETHTOOL_A_CABLE_TEST_STATUS	u8	completed
ETHTOOL_A_CABLE_TEST_NTF_NEST	nested	all the results
ETHTOOL_A_CABLE_NEST_RESULT	nested	cable test result
ETHTOOL_A_CABLE_RESULTS_PAIR	u8	pair number
ETHTOOL_A_CABLE_RESULTS_CODE	u8	result code
ETHTOOL_A_CABLE_NEST_RESULT	nested	cable test results

ETHTOOL_A_CABLE_RESULTS_PAIR	u8	pair number
ETHTOOL_A_CABLE_RESULTS_CODE	u8	result code
ETHTOOL_A_CABLE_NEST_FAULT_LENGTH	nested	cable length
ETHTOOL_A_CABLE_FAULT_LENGTH_PAIR	u8	pair number
ETHTOOL_A_CABLE_FAULT_LENGTH_CM	u32	length in cm

CABLE_TEST TDR

Start a cable test and report raw TDR data

Request contents:

ETHTOOL_A_CABLE_TEST_TDR_HEADER	nested	reply header
ETHTOOL_A_CABLE_TEST_TDR_CFG	nested	test configuration
ETHTOOL_A_CABLE_STEP_FIRST_DISTANCE	u32	first data distance
ETHTOOL_A_CABLE_STEP_LAST_DISTANCE	u32	last data distance
ETHTOOL_A_CABLE_STEP_STEP_DISTANCE	u32	distance of each step
ETHTOOL_A_CABLE_TEST_TDR_CFG_PAIR	u8	pair to test

The ETHTOOL_A_CABLE_TEST_TDR_CFG is optional, as well as all members of the nest. All distances are expressed in centimeters. The PHY takes the distances as a guide, and rounds to the nearest distance it actually supports. If a pair is passed, only that one pair will be tested. Otherwise all pairs are tested.

Notification contents:

Raw TDR data is gathered by sending a pulse down the cable and recording the amplitude of the reflected pulse for a given distance.

It can take a number of seconds to collect TDR data, especial if the full 100 meters is probed at 1 meter intervals. When the test is started a notification will be sent containing just ETHTOOL_A_CABLE_TEST_TDR_STATUS with the value ETHTOOL_A_CABLE_TEST_NTF_STATUS_STARTED.

When the test has completed a second notification will be sent containing ETHTOOL_A_CABLE_TEST_TDR_STATUS with the value ETHTOOL_A_CABLE_TEST_NTF_STATUS_COMPLETED and the TDR data.

The message may optionally contain the amplitude of the pulse send down the cable. This is measured in mV. A reflection should not be bigger than transmitted pulse.

Before the raw TDR data should be an ETHTOOL_A_CABLE_TDR_NEST_STEP nest containing information about the distance along the cable for the first reading, the last reading, and the step between each reading. Distances are measured in centimeters. These should be the exact values the PHY used. These may be different to what the user requested, if the native measurement resolution is greater than 1 cm.

For each step along the cable, a ETHTOOL_A_CABLE_TDR_NEST_AMPLITUDE is used to report the amplitude of the reflection for a given pair.

ETHTOOL_A_CABLE_TEST_TDR_HEADER	nested	reply header
ETHTOOL_A_CABLE_TEST_TDR_STATUS	u8	completed
ETHTOOL_A_CABLE_TEST_TDR_NTF_NEST	nested	all the results
ETHTOOL_A_CABLE_TDR_NEST_PULSE	nested	TX Pulse amplitude
ETHTOOL_A_CABLE_PULSE_mV	s16	Pulse amplitude
ETHTOOL_A_CABLE_NEST_STEP	nested	TDR step info
ETHTOOL_A_CABLE_STEP_FIRST_DISTANCE	u32	First data distance
ETHTOOL_A_CABLE_STEP_LAST_DISTANCE	u32	Last data distance
ETHTOOL_A_CABLE_STEP_STEP_DISTANCE	u32	distance of each step
ETHTOOL_A_CABLE_TDR_NEST_AMPLITUDE	nested	Reflection amplitude
ETHTOOL_A_CABLE_RESULTS_PAIR	u8	pair number
ETHTOOL_A_CABLE_AMPLITUDE_mV	s16	Reflection amplitude
ETHTOOL_A_CABLE_TDR_NEST_AMPLITUDE	nested	Reflection amplitude
ETHTOOL_A_CABLE_RESULTS_PAIR	u8	pair number
ETHTOOL_A_CABLE_AMPLITUDE_mV	s16	Reflection amplitude
ETHTOOL_A_CABLE_TDR_NEST_AMPLITUDE	nested	Reflection amplitude
ETHTOOL_A_CABLE_RESULTS_PAIR	u8	pair number
ETHTOOL_A_CABLE_AMPLITUDE_mV	s16	Reflection amplitude

TUNNEL_INFO

Gets information about the tunnel state NIC is aware of.

Request contents:

ETHTOOL_A_TUNNEL_INFO_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_TUNNEL_INFO_HEADER	nested	reply header
ETHTOOL_A_TUNNEL_INFO_UDP_PORTS	nested	all UDP port tables
ETHTOOL_A_TUNNEL_UDP_TABLE	nested	one UDP port table
ETHTOOL_A_TUNNEL_UDP_TABLE_SIZE	u32	max size of the table
ETHTOOL_A_TUNNEL_UDP_TABLE_TYPES	bitset	tunnel types which table can hold
ETHTOOL_A_TUNNEL_UDP_TABLE_ENTRY	nested	offloaded UDP port
ETHTOOL_A_TUNNEL_UDP_ENTRY_PORT	be16	UDP port
ETHTOOL_A_TUNNEL_UDP_ENTRY_TYPE	u32	tunnel type

For UDP tunnel table empty ETHTOOL_A_TUNNEL_UDP_TABLE_TYPES indicates that the table contains static entries, hard-coded by the NIC.

FEC_GET

Gets FEC configuration and state like ETHTOOL_GFECPARAM ioctl request.

Request contents:

ETHTOOL_A_FEC_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_FEC_HEADER	nested	request header
ETHTOOL_A_FEC_MODES	bitset	configured modes
ETHTOOL_A_FEC_AUTO	bool	FEC mode auto selection
ETHTOOL_A_FEC_ACTIVE	u32	index of active FEC mode
ETHTOOL_A_FEC_STATS	nested	FEC statistics

ETHTOOL_A_FEC_ACTIVE is the bit index of the FEC link mode currently active on the interface. This attribute may not be present if device does not support FEC.

ETHTOOL_A_FEC_MODES and ETHTOOL_A_FEC_AUTO are only meaningful when autonegotiation is disabled. If ETHTOOL_A_FEC_AUTO is non-zero driver will select the FEC mode automatically based on the parameters of the SFP module. This is equivalent to the ETHTOOL_FEC_AUTO bit of the ioctl interface. ETHTOOL_A_FEC_MODES carry the current FEC configuration using link mode bits (rather than old ETHTOOL_FEC_* bits).

ETHTOOL_A_FEC_STATS are reported if ETHTOOL_FLAG_STATS was set in ETHTOOL_A_HEADER_FLAGS. Each attribute carries an array of 64bit statistics. First entry in the array contains the total number of events on the port, while the following entries are counters corresponding to lanes/PCS instances. The number of entries in the array will be:

0	device does not support FEC statistics
1	device does not support per-lane break down
1 + #lanes	device has full support for FEC stats

Drivers fill in the statistics in the following structure:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\[linux-master][Documentation][networking]ethtool-netlink.rst, line 1404)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/ethtool.h
    :identifiers: ethtool_fec_stats
```

FEC_SET

Sets FEC parameters like ETHTOOL_SFECPARAM ioctl request.

Request contents:

ETHTOOL_A_FEC_HEADER	nested	request header
ETHTOOL_A_FEC_MODES	bitset	configured modes
ETHTOOL_A_FEC_AUTO	bool	FEC mode auto selection

FEC_SET is only meaningful when autonegotiation is disabled. Otherwise FEC mode is selected as part of autonegotiation.

ETHTOOL_A_FEC_MODES selects which FEC mode should be used. It's recommended to set only one bit, if multiple bits are set driver may choose between them in an implementation specific way.

ETHTOOL_A_FEC_AUTO requests the driver to choose FEC mode based on SFP module parameters. This does not mean autonegotiation.

MODULE_EEPROM_GET

Fetch module EEPROM data dump. This interface is designed to allow dumps of at most 1/2 page at once. This means only dumps of 128 (or less) bytes are allowed, without crossing half page boundary located at offset 128. For pages other than 0 only high 128 bytes are accessible.

Request contents:

ETHTOOL_A_MODULE_EEPROM_HEADER	nested	request header
ETHTOOL_A_MODULE_EEPROM_OFFSET	u32	offset within a page
ETHTOOL_A_MODULE_EEPROM_LENGTH	u32	amount of bytes to read
ETHTOOL_A_MODULE_EEPROM_PAGE	u8	page number
ETHTOOL_A_MODULE_EEPROM_BANK	u8	bank number
ETHTOOL_A_MODULE_EEPROM_I2C_ADDRESS	u8	page I2C address

If ETHTOOL_A_MODULE_EEPROM_BANK is not specified, bank 0 is assumed.

Kernel response contents:

ETHTOOL_A_MODULE_EEPROM_HEADER	nested	reply header
ETHTOOL_A_MODULE_EEPROM_DATA	binary	array of bytes from module EEPROM

ETHTOOL_A_MODULE_EEPROM_DATA has an attribute length equal to the amount of bytes driver actually read.

STATS_GET

Get standard statistics for the interface. Note that this is not a re-implementation of ETHTOOL_GSTATS which exposed driver-defined stats.

Request contents:

ETHTOOL_A_STATS_HEADER	nested	request header
ETHTOOL_A_STATS_GROUPS	bitset	requested groups of stats

Kernel response contents:

ETHTOOL_A_STATS_HEADER	nested	reply header
ETHTOOL_A_STATS_GRP	nested	one or more group of stats
ETHTOOL_A_STATS_GRP_ID	u32	group ID - ETHTOOL_STATS_*
ETHTOOL_A_STATS_GRP_SS_ID	u32	string set ID for names
ETHTOOL_A_STATS_GRP_STAT	nested	nest containing a statistic
ETHTOOL_A_STATS_GRP_HIST_RX	nested	histogram statistic (Rx)
ETHTOOL_A_STATS_GRP_HIST_TX	nested	histogram statistic (Tx)

Users specify which groups of statistics they are requesting via the ETHTOOL_A_STATS_GROUPS bitset. Currently defined values are:

ETHTOOL_STATS_ETH_MAC	eth-mac	Basic IEEE 802.3 MAC statistics (30.3.1.1.*)
ETHTOOL_STATS_ETH_PHY	eth-phy	Basic IEEE 802.3 PHY statistics (30.3.2.1.*)
ETHTOOL_STATS_ETH_CTRL	eth-ctrl	Basic IEEE 802.3 MAC Ctrl statistics (30.3.3.*)
ETHTOOL_STATS_RMON	rmon	RMON (RFC 2819) statistics

Each group should have a corresponding ETHTOOL_A_STATS_GRP in the reply. ETHTOOL_A_STATS_GRP_ID identifies which group's statistics nest contains. ETHTOOL_A_STATS_GRP_SS_ID identifies the string set ID for the names of the statistics in the group, if available.

Statistics are added to the ETHTOOL_A_STATS_GRP nest under ETHTOOL_A_STATS_GRP_STAT. ETHTOOL_A_STATS_GRP_STAT should contain single 8 byte (u64) attribute inside - the type of that attribute is the statistic ID and the value is the value of the statistic. Each group has its own interpretation of statistic IDs. Attribute IDs correspond to strings from the string set identified by ETHTOOL_A_STATS_GRP_SS_ID. Complex statistics (such as RMON histogram entries) are also listed inside

ETHTOOL_A_STATS_GRP and do not have a string defined in the string set.

RMON "histogram" counters count number of packets within given size range. Because RFC does not specify the ranges beyond the standard 1518 MTU devices differ in definition of buckets. For this reason the definition of packet ranges is left to each driver.

ETHTOOL_A_STATS_GRP_HIST_RX and ETHTOOL_A_STATS_GRP_HIST_TX nests contain the following attributes:

ETHTOOL_A_STATS_RMON_HIST_BKT_LOW	u32	low bound of the packet size bucket
ETHTOOL_A_STATS_RMON_HIST_BKT_HI	u32	high bound of the bucket
ETHTOOL_A_STATS_RMON_HIST_VAL	u64	packet counter

Low and high bounds are inclusive, for example:

RFC statistic	low	high
etherStatsPkts64Octets	0	64
etherStatsPkts512to1023Octets	512	1023

PHC_VCLOCKS_GET

Query device PHC virtual clocks information.

Request contents:

ETHTOOL_A_PHC_VCLOCKS_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_PHC_VCLOCKS_HEADER	nested	reply header
ETHTOOL_A_PHC_VCLOCKS_NUM	u32	PHC virtual clocks number
ETHTOOL_A_PHC_VCLOCKS_INDEX	s32	PHC index array

MODULE_GET

Gets transceiver module parameters.

Request contents:

ETHTOOL_A_MODULE_HEADER	nested	request header

Kernel response contents:

ETHTOOL_A_MODULE_HEADER	nested	reply header
ETHTOOL_A_MODULE_POWER_MODE_POLICY	u8	power mode policy
ETHTOOL_A_MODULE_POWER_MODE	u8	operational power mode

The optional ETHTOOL_A_MODULE_POWER_MODE_POLICY attribute encodes the transceiver module power mode policy enforced by the host. The default policy is driver-dependent, but "auto" is the recommended default and it should be implemented by new drivers and drivers where conformance to a legacy behavior is not critical.

The optional ETHTHOOL_A_MODULE_POWER_MODE attribute encodes the operational power mode policy of the transceiver module. It is only reported when a module is plugged-in. Possible values are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\[linux-master][Documentation][networking]ethtool-netlink.rst, line 1592)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/uapi/linux/ethtool.h
    :identifiers: ethtool_module_power_mode
```

MODULE_SET

Sets transceiver module parameters.

Request contents:

ETHTOOL_A_MODULE_HEADER	nested	request header
ETHTOOL_A_MODULE_POWER_MODE_POLICY	u8	power mode policy

When set, the optional ETHTOOL_A_MODULE_POWER_MODE_POLICY attribute is used to set the transceiver module power policy enforced by the host. Possible values are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\[linux-master][Documentation][networking]ethtool-netlink.rst, line 1611)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/uapi/linux/ethtool.h
    :identifiers: ethtool_module_power_mode_policy
```

For SFF-8636 modules, low power mode is forced by the host according to table 6-10 in revision 2.10a of the specification.

For CMIS modules, low power mode is forced by the host according to table 6-12 in revision 5.0 of the specification.

Request translation

The following table maps ioctl commands to netlink commands providing their functionality. Entries with "n/a" in right column are commands which do not have their netlink replacement yet. Entries which "n/a" in the left column are netlink only.

ioctl command	netlink command
ETHTOOL_GSET	ETHTOOL_MSG_LINKINFO_GET ETHTOOL_MSG_LINKMODES_GET
ETHTOOL_SSET	ETHTOOL_MSG_LINKINFO_SET ETHTOOL_MSG_LINKMODES_SET
ETHTOOL_GDRVINFO	n/a
ETHTOOL_GREGS	n/a
ETHTOOL_GWOL	ETHTOOL_MSG_WOL_GET
ETHTOOL_SWOL	ETHTOOL_MSG_WOL_SET
ETHTOOL_GMSGLVL	ETHTOOL_MSG_DEBUG_GET
ETHTOOL_SMSGLVL	ETHTOOL_MSG_DEBUG_SET
ETHTOOL_NWAY_RST	n/a
ETHTOOL_GLINK	ETHTOOL_MSG_LINKSTATE_GET
ETHTOOL_GEEPROM	n/a
ETHTOOL_SEEPROM	n/a
ETHTOOL_GCOALESCE	ETHTOOL_MSG_COALESCE_GET
ETHTOOL_SCOALESCE	ETHTOOL_MSG_COALESCE_SET
ETHTOOL_GRINGPARAM	ETHTOOL_MSG_RINGS_GET
ETHTOOL_SRINGPARAM	ETHTOOL_MSG_RINGS_SET
ETHTOOL_GPAUSEPARAM	ETHTOOL_MSG_PAUSE_GET
ETHTOOL_SPAUSEPARAM	ETHTOOL_MSG_PAUSE_SET
ETHTOOL_GRXCSUM	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_SRXCSUM	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_GTXCSUM	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_STXCSUM	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_GSG	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_SSG	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_TEST	n/a
ETHTOOL_GSTRINGS	ETHTOOL_MSG_STRSET_GET
ETHTOOL_PHYS_ID	n/a
ETHTOOL_GSTATS	n/a
ETHTOOL_GTSO	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_STSO	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_GPERMADDR	rtnetlink RTM_GETLINK
ETHTOOL_GUFO	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_SUFO	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_GGSO	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_SGSO	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_GFLAGS	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_SFLAGS	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_GPFLAGS	ETHTOOL_MSG_PRIVFLAGS_GET
ETHTOOL_SPFLAGS	ETHTOOL_MSG_PRIVFLAGS_SET
ETHTOOL_GRXFH	n/a

ioctl command	netlink command
ETHTOOL_SRXFH	n/a
ETHTOOL_GGRO	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_SGRO	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_GRXRINGS	n/a
ETHTOOL_GRXCLSRLCNT	n/a
ETHTOOL_GRXCLSRULE	n/a
ETHTOOL_GRXCLSRLALL	n/a
ETHTOOL_SRXCLSRLDEL	n/a
ETHTOOL_SRXCLSRLINS	n/a
ETHTOOL_FLASHDEV	n/a
ETHTOOL_RESET	n/a
ETHTOOL_SRXNTUPLE	n/a
ETHTOOL_GRXNTUPLE	n/a
ETHTOOL_GSSET_INFO	ETHTOOL_MSG_STRSET_GET
ETHTOOL_GRXFHINDIR	n/a
ETHTOOL_SRXFHINDIR	n/a
ETHTOOL_GFEATURES	ETHTOOL_MSG_FEATURES_GET
ETHTOOL_SFEATURES	ETHTOOL_MSG_FEATURES_SET
ETHTOOL_GCHANNELS	ETHTOOL_MSG_CHANNELS_GET
ETHTOOL_SCHANNELS	ETHTOOL_MSG_CHANNELS_SET
ETHTOOL_SET_DUMP	n/a
ETHTOOL_GET_DUMP_FLAG	n/a
ETHTOOL_GET_DUMP_DATA	n/a
ETHTOOL_GET_TS_INFO	ETHTOOL_MSG_TSINFO_GET
ETHTOOL_GMODULEINFO	ETHTOOL_MSG_MODULE_EEPROM_GET
ETHTOOL_GMODULEEEPROM	ETHTOOL_MSG_MODULE_EEPROM_GET
ETHTOOL_GEEE	ETHTOOL_MSG_EEE_GET
ETHTOOL_SEEE	ETHTOOL_MSG_EEE_SET
ETHTOOL_GRSSH	n/a
ETHTOOL_SRSSH	n/a
ETHTOOL_GTUNABLE	n/a
ETHTOOL_STUNABLE	n/a
ETHTOOL_GPHYSTATS	n/a
ETHTOOL_PERQUEUE	n/a
ETHTOOL_GLINKSETTINGS	ETHTOOL_MSG_LINKINFO_GET ETHTOOL_MSG_LINKMODES_GET
ETHTOOL_SLINKSETTINGS	ETHTOOL_MSG_LINKINFO_SET ETHTOOL_MSG_LINKMODES_SET
ETHTOOL_PHY_GTUNABLE	n/a
ETHTOOL_PHY_STUNABLE	n/a
ETHTOOL_GFECPARAM	ETHTOOL_MSG_FEC_GET
ETHTOOL_SFECPARAM	ETHTOOL_MSG_FEC_SET
n/a	ETHTOOL_MSG_CABLE_TEST_ACT
n/a	ETHTOOL_MSG_CABLE_TEST_TDR_ACT
n/a	ETHTOOL_MSG_TUNNEL_INFO_GET
n/a	ETHTOOL_MSG_PHC_VCLOCKS_GET
n/a	ETHTOOL_MSG_MODULE_GET
n/a	ETHTOOL_MSG_MODULE_SET