Silicom

Bypass / TAP Programmer Guide

-

Linux OS

REVISION HISTORY

Revision	Date	Change description
1.0	29-Jul-05	Linux side driver programmer guide – initial version
1.1	28-Nov-05	Added products PEG2BP, PEG4BP, PEG2BPI, PEG4BPI, PEG2BPFI/LX,PXE2TAPI, PXG2BPFIL. Add Set/Get_WD_AutoReset command.
1.2	05-Jun-06	Added support for bypass_info command
2.1	12-Jul-06	Added support for user-level and kernel-level libraries
2.2	21-Feb-07	Added PXG2BISC1 product
2.3	9-Jan-08	Added DISC and TPL commands and couple of new adapters
2.4	19-May-08	Added 10G adapters
2.5	10-Nov-08	Added support for 10G BCM-based adapters and 1G 82576 family adapters.
2.6	20-Oct-09	Added new supported products
2.7	24-Jan-10	Added support for user-space driver (IOCTL interface)
2.8	7-Feb-10	Added Director products
2.9	17-Feb-10	Added PEG2BPFi6-FLXM-SD-R
3.0	31-May-10	Add link to the Target release section
4.0	17-Jan-11	Added support for force_link
4.1	8-Feb-11	Added description to the TPL
4.2	27-Jun-11	Added support for PROC interface

1 INTRODUCTION

1.1 General

The Silicom Bypass/TAP-Series Server 1 Gigabit/ 10 Gigabit adapters are 64-bit/133Mhz PCI-X / PCI Express network interface cards that contain multiple independent Gigabit ports on one PCI-X/PCI-E adapter.

Silicom's Bypass-Series are designed with a Bypass circuitry in order to provide maximum up time for the network.

Silicom's tapping product series enable in addition to the bypass functionality to TAP pairs of port traffic via the pairs of interfaces.

In Bypass mode, the connections between the two interfaces to their Giga Ethernet ports is disconnected, and a crossed connection loop-back between the two Giga Ethernet port is establish. In this mode all packets received from one port are transmitted to the other port and vice versa. The interfaces are disconnected and no link is established on these interfaces in the bypass mode.

In Tapping mode, a crossed connection loop-back between the two Giga Ethernet port is establish as they are connected together. At the same time the receive side of the pair of interfaces, in conjunction to these two ports, are connected to these two ports to sense their data transfer and can be taped by these two interfaces.

Normal mode is when the ports are connected directly to the MAC controller and operate independently.

In Disconnect mode the adapter simulates switch / router cable disconnection. In Disconnect mode, the switch / router does not detect link partner of the adapter.

1.2 Purpose

This document describe the Bypass functionality of Silicom bypass product line – PXBPI, PXTBI and PEBPI series and provide functional description of the API available for used to control the Bypass interfaces via user-level application or kernel-level driver.

1.3 Scope

This document includes description of the Silicom Bypass function and its operation and describes list of APIs supported by the bypass products and the method of accessing it and responses returned from it. This document appendix provide list of capabilities and supported features for each product and sample commands and tools to operate the different controlling methods.

1.4 Target Release

For more information on which OS each Silicom Bypass/TAP products support refer to the user manual in the introduction page.

1.5 Definitions

BP – Bypass mode: Ethernet ports connections of the interfaces are disconnected from the interfaces and a "short-circuit" between the two ports on the same network adapter is creating a crossed connection loop-back between the ports. See figure 1 for description of the connection in Bypass mode.

TAP mode: In Tapping mode, a crossed connection loop-back between the two Giga Ethernet ports is establish as they are connected together. At the same time the receive side of the pair of interfaces, in conjunction to these two ports, are connected to these two ports to sense their data transfer and can be taped by these two interfaces.

Normal mode: Ethernet ports operate independently and connected directly to their associated Ethernet controllers. BP_SW1 / 2 in figure-1 change their mode to connect port1 / 2 to interfaces1 / 2.

Disconnect mode: In Disconnect mode the adapter simulates switch / router cable disconnection. In Disconnect mode, the switch / router does not detect link partner of the adapter.

Power-off state: state when there is no power supplied to the product (computer is powered off)

Power-up state: state from the moment the power to the product is resumed until the product driver is loaded and operating.

Operating state: state when the product driver is loaded and running. In this state the product can be configured for its Bypass configuration modes and operation.

Bypass product default states: Each Bypass product at its initial manufacturing configuration will operate as follows:

- At Power-off state - Bypass mode.
- At Power-up state - remain in Bypass mode.
- After installation and driver loading will remain in Bypass mode.
- After SET_BYPASS off command (BP_SW1 / 2 changes their mode) and as long as the watchdog time (if enabled) not expires the product will operate in normal mode of operation.

Change from this default states can be done with commands (if supported by the product) to operate the products differently (SET_DIS_BYPASS, SET_PWRUP(off) and other).

TAP product default states:

Each TAP product at its initial manufacturing configuration will operate as follows:

- At Power-off state - Bypass mode.
- At Power-up state - remain in TAP mode.
- After installation and driver loading will remain in TAP mode.
- After SET_TAP off command (BP_SW1 / 2 changes their mode) and as long as the watchdog time (if enabled) not expires the product will operate in normal mode of operation.

Change from this default stat can be done with commands (if supported by the product) to operate the products differently (SET_DIS_BYPASS, SET_PWRUP(off) and other).

Note that PXG2TBFI TAP mode is also Bypass mode

Standard NIC mode: This mode set the product to operate in normal mode at all operation modes (power-off, power-up, driver loaded and running) without having Bypass/TAP/Disconnect mode and enabling direct connection of interfaces to their ports.

Disable Bypass short circuit: Dis_BP_SW in figure-1 can be controlled (if supported by the product) to disable the Bypass/TAP short circuit between the two ports. These switches are latch switches that retain their state even at power off state. This can enable disabling short circuit bypass connection even if a SET_BYPASS(on) command is issued.

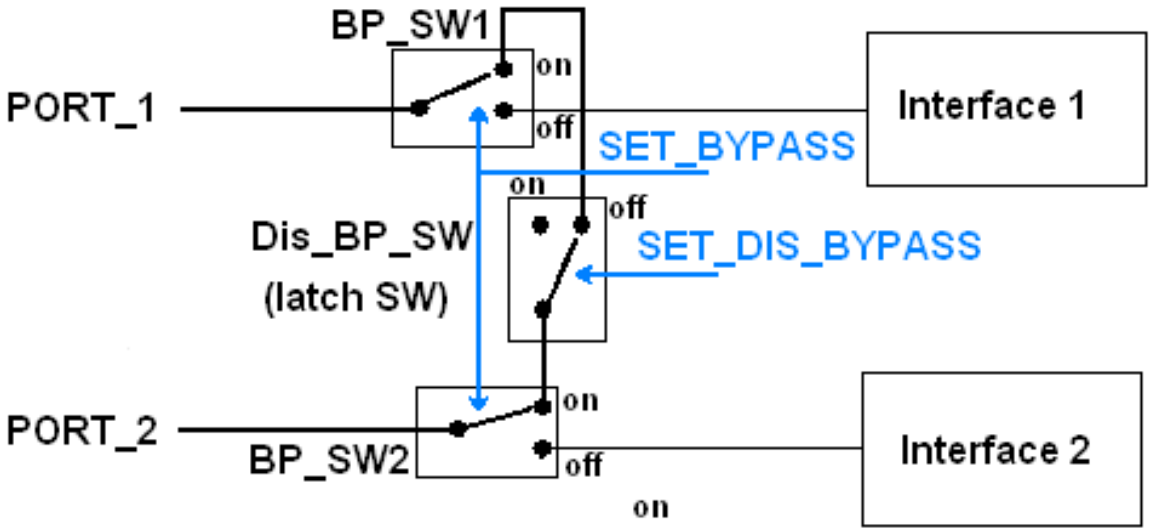


Figure-1

Controlling interface: The interface, of the bypass interface pair, which the bypass functionality is controlled from. From this interface the bypass functionality will be controlled and will affect both interfaces.

Slave interface: The interface, of the bypass interface pair, which is controlled by the controlling interface. Issuing commands to this interface will result with “not supported” results.

1.6 Bypass/TAP features

Basic features that Bypass/TAP interface supports are:

1. Go to Bypass mode on power failure
2. Go to Bypass/TAP/Disconnect mode on software command
3. Go to Bypass/ TAP/Disconnect mode on watchdog timeout.
4. Get status of the different parameters that are supported by the specific product
5. Get information of the product supported capabilities

The adapter may or may not support them depending on the adapter capabilities. All Silicom Bypass products do support these main basic features.

1.7 Operating as Standard NIC

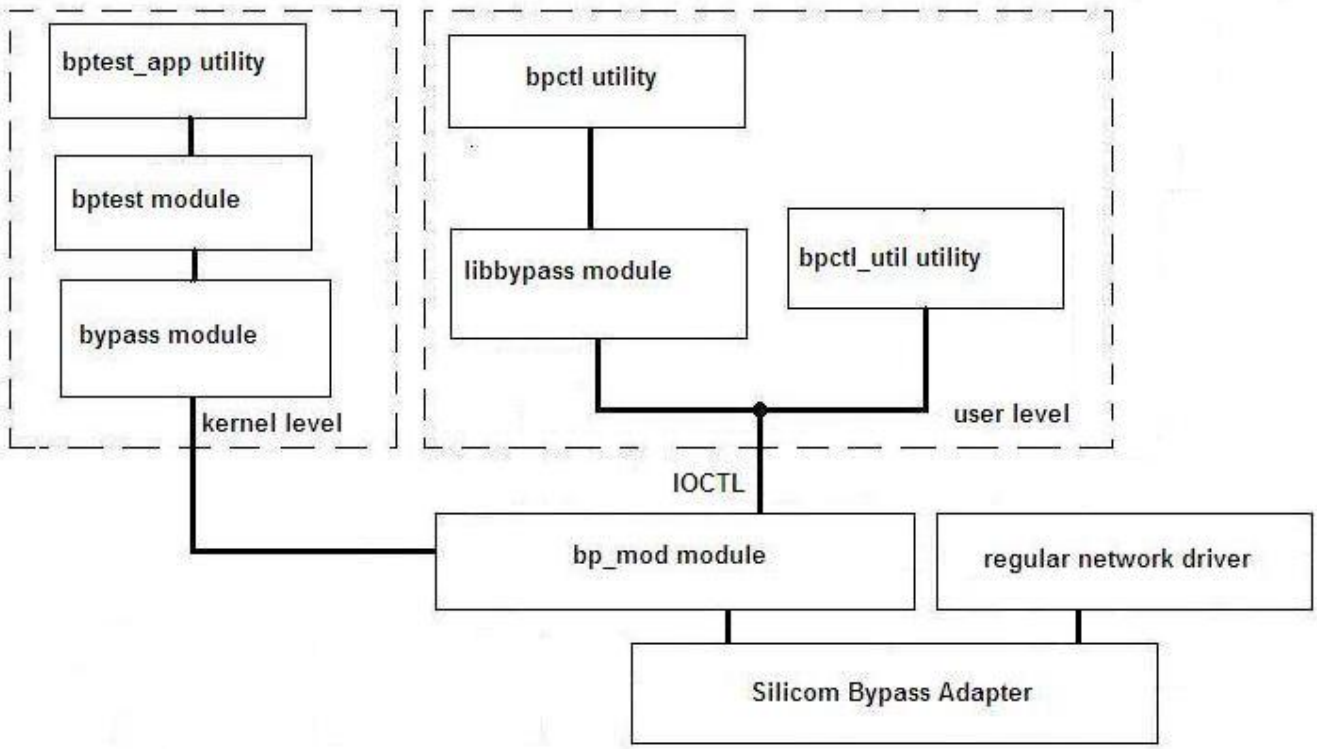
These products are designed to operate as multi port Giga Ethernet adapters with Bypass/TAP/Disconnect capabilities for each pairs of interfaces. Together with this an effort was done to enable operation of the products as standard NIC if needed. This means that the product will automatically work as a NIC without the need to configure any of the bypass parameters. This capability has some limitations due to lack of board space and cost issues and differ between the different products.
For more detailed information see Appendix B – Operating as standard NIC description

1.8 Bypass / TAP software interface

The Bypass software interface has been defined such that it will be able to work with different Bypass products using different chipset and having each one different bypass capabilities and supporting different feature.
The software accessing the product driver via this software interface will have to check, before starting configuring / working with the adapter, its support for Bypass operation and for the different supported features and then after knowing supported features, configure it accordingly and verify that it has configured correctly.

This document defines 2 types of interfaces:
1. Interface via IOCTL calls
2. Function interface (for user and kernel level).
3. PROC interface

See the figure below for detail on the different interfaces.



2 IOCTL API

This section defines the different IOCTL calls to be used for interface with the Bypass functionality.

2.1 Bypass / TAP support detection

This section describes how the software can distinguish between Silicom network interface supporting Bypass/TAP/Disconnect functionality and regular network interface that may be also installed in the system. This is only required when the application accessing to the bypass function do not know in advance if or which interface is Silicom bypass supporting.
The software can, before accessing to the BP functions check if this network interface supports Silicom BP functions. This is done by checking the product PCI-ID (SVID and SPID).
Silicom Bypass products will have SVID (SubVendor ID) of 0x1374 and SPID (SubProduct ID) in the range of 0x0024-0x005F.
Sample user-level code for performing this detection is described in
IOCTL_Bypass_Support_Detection
After detecting that the product is Silicom Bypass product (ID's in the range) use the IS_BYPASS command, as describe below, to verify its bypass support. Silicom Bypass product will respond with success for this command.

- **Note:** When Bypass interfaces are down, IOCTL API can't be used for BCM57710-based adapters.

2.2 IOCTL request to Bypass Driver.

Below is the interface to access the Bypass Driver via IOCTL calls.

```
static enum {
        IF_SCAN,
        GET_DEV_NUM,
        IS_BYPASS,
        GET_BYPASS_SLAVE,          /*Bypass and TAP products*/
        GET_BYPASS_CAPS,           /*Bypass and TAP products*/
        GET_WD_SET_CAPS,           /*Bypass and TAP products*/
        SET_BYPASS,                /*for Bypass support products*/
        GET_BYPASS,                /*for Bypass support products*/
        GET_BYPASS_CHANGE,         /*for Bypass support products*/
        SET_BYPASS_WD,             /*Bypass and TAP products*/
        GET_BYPASS_WD,             /*Bypass and TAP products*/
        GET_WD_EXPIRE_TIME,        /*Bypass and TAP products*/
        RESET_BYPASS_WD_TIMER,     /*Bypass and TAP products*/
        SET_DIS_BYPASS,            /*for Bypass support products */
        GET_DIS_BYPASS,            /*for Bypass support products */
        SET_BYPASS_PWOFF,          /*for Bypass support products*/
        GET_BYPASS_PWOFF,          /*for Bypass support products*/
        SET_BYPASS_PWUP,           /*for Bypass support products*/
```

```
        GET_BYPASS_PWUP,            /*for Bypass support products*/
        SET_STD_NIC,                /*Bypass and TAP products*/
        GET_STD_NIC,                /*Bypass and TAP products*/
        SET_TX,                     /* Bypass and TAP products to control the link*/
        GET_TX,                     /* Bypass and TAP products to control the link*/
                                    /*TAP products commands*/
        SET_TAP,                    /*for TAP support products*/
        GET_TAP,                    /*for TAP support products*/
        GET_TAP_CHANGE,             /*for TAP support products*/
        SET_DIS_TAP,                /*for TAP support products*/
        GET_DIS_TAP,                /*for TAP support products*/
        SET_TAP_PWUP,               /*for TAP support products*/
        GET_TAP_PWUP,               /*for TAP support products*/
        SET_WD_EXP_MODE,            /* for products supporting both TAP and Bypass */
        GET_WD_EXP_MODE,            /*for products supporting both TAP and Bypass*/
        SET_WD_AUTORESET,
        SET_WD_AUTORESET,
        SET_TPL,
        GET_TPL,
        SET_DISC,
        GET_DISC,
        GET_DISC_CHANGE,
        SET_DIS_DISC,
        GET_DIS_DISC,
        GET_DISC_PWUP,
        GET_BYPASS_INFO=100,
        GET_BP_WAIT_AT_PWUP,
         SET_BP_WAIT_AT_PWUP,
         GET_BP_HW_RESET,
         SET_BP_HW_RESET,
         SET_DISC_PORT,
         GET_DISC_PORT,
         SET_DISC_PORT_PWUP,
         GET_DISC_PORT_PWUP,
         SET_BP_FORCE_LINK, /* For Redirector card only */
         GET_BP_FORCE_LINK, /* For Redirector card only */
       #ifdef BP_SELF_TEST
         SET_BP_SELF_TEST=200,
         GET_BP_SELF_TEST,
       #endif
} CMND_TYPE;

/* for passing single values */
struct bpctl_cmd {
   int status;
   int data[8];
   int in_param[8];
   int out_param[8];
```

};

#define MAGIC_NUM 'J'

#define IOCTL_TX_MSG(cmd) _IOWR(MAGIC_NUM, cmd, struct bpctl_cmd)

Note: cmd – from CMND_TYPE enum.

#define DEVICE_NODE "/dev/bpctl0"
#define DEVICE_NAME "bpctl"

Bypass IOCTL API user-level command format:

file_desc = open(DEVICE_NODE, 0);
ioctl(file_desc, IOCTL_TX_MSG(cmd), pbpctl_cmd);

file_desc - open file descriptor;
pbpctl_cmd - pointer to bpctl_cmd structure.

Bypass configuration parameters are passed in the bpctl_cmd structure. The fields of the bpctl_cmd structure are shown below:

Field	Description	Value/Note
bpctl_cmd:status	Used by the IOCTL routine to get command status.	Usually, on success zero or non-zero value. On error, -1 is returned.
bpctl_cmd:data	Used by the IOCTL routine to pass additional parameters.	Depending on command, see below.
bpctl_cmd:in_param	Used by the IOCTL routine to pass input parameters.	**bpctl_cmd:in_param[0…1]** - for all commands (except IF_SCAN and GET_DEV_NUM), if the reference to the device as to the network interface: **bpctl_cmd:in_param[1]** = *ifindex* else: **bpctl_cmd:in_param[0]** = *dev_num* (in this case bpctl_cmd:in_param[1] must be set to 0) *ifindex* – network interface index; to rescan the list of network interfaces, use IF_SCAN command. *dev_num* – bypass device number; to get total number of bypass devices, use GET_DEV_NUM command. **bpctl_cmd:in_param[2…4]** - depending on command, see below. **bpctl_cmd:out_param[5]** – bus **bpctl_cmd:out_param[6]** – slot **bpctl_cmd:out_param[7]** – function
bpctl_cmd:out_param	Used by the IOCTL routine to pass output parameters.	**bpctl_cmd:out_param[0…3]** - for all commands (except IF_SCAN and GET_DEV_NUM), extended information about requested device: **bpctl_cmd:out_param[0]** – bus **bpctl_cmd:out_param[1]** – slot **bpctl_cmd:out_param[2]** – function **bpctl_cmd:out_param[3]** – *ifindex* **bpctl_cmd:in_param[4…7]** - depending on

command, see below.

2.3 Basic Commands

2.3.1 IS_BYPASS

Description - Check if device is a Bypass/TAP controlling device

Command format:

ioctl (file_desc, IOCTL_TX_MSG(IS_BYPASS), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***1*** if the device is bypass/TAP controlling device, ***0*** if the device is bypass/TAP slave device
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.2 GET_BYPASS_SLAVE

Description - Get the second port participate in the Bypass/TAP pair

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_SLAVE), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***-1*** on failure, ***1*** if the device is bypass/TAP controlling device, ***0*** if the device is bypass/TAP slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device. **bpctl_cmd:out_param[4…7]** - extended info about slave of the Bypass/TAP ports pair. This is significant only if **bpctl_cmd:status=1** (requested device is bypass/TAP controlling device). **bpctl_cmd:out_param[4]** – bus **bpctl_cmd:out_param[5]** – function **bpctl_cmd:out_param[6]** – device **bpctl_cmd:out_param[7]** – *ifindex*

2.3.3 SET_BYPASS

Description - Set Bypass On/Off mode

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_BYPASS), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]**= *bypass_mode* (1=on, 0=off)

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***-1*** on failure, if the device is not capable of the operation (TAP only device) or is a slave device. ***0*** on success.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: this command control the state of BP_SW1/2 state without any relation to the Dis_BP_SW switch state (Figure-1).
  In case of SET_DIS_BYPASS or SET_STD_NIC is ON the product will not switch to bypass even if SET_BYPASS ON command will be issued.

2.3.4 GET_BYPASS

Description - Get Bypass state (On/Off)

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***-1*** on failure, if the device is not capable of the operation (TAP only device) or is a slave device. ***0/1*** (off/on) on success.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: Bypass ON will be returned whenever it is in bypass mode not related to the cause of it (command or watchdog timer). In case of SET_DIS_BYPASS or SET_STD_NIC is ON the product will not switch to bypass mode (even if GET_BYPASS command will show that it is ON).

2.3.5 GET_BYPASS_CHANGE

Description - Get change in Bypass mode state from last GET_BYPASS_CHANGE status check

This function enable checking if the device changed to bypass mode, from some reason (WD timeout or any other) from last GET_BYPASS_CHANGE status check. This can determine if there was a situation that the product went to bypass mode and recovered. This function will enable knowing that this situation has occurred.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_CHANGE), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***-1*** on failure, if the device is not capable of the operation (TAP only device) or is a slave device. ***0/1*** (off/on) on success.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.6 SET_BYPASS_WD

Description - Set watchdog timer On/Off and timeout, for Bypass and TAP products.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_BYPASS_WD), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]**=*timeout* (requested timeout (in ms units), 0 for disabling the watchdog timer).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***-1*** on failure, if the device is a slave device. ***0*** on success – if watchdog is disabled, ***ms_timeout_set*** – on success - actual value (in ms units) supported by the interface and that will be used by the watchdog.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: The returned timeout value (in ms units) is for the caller to check and adjust its timeout value in case of mismatch to the requested value. The returned value will be the nearest one available that is equal or larger from the requested value, but not larger then the maximum value it support.
- **Note**: The WDT intend to switch the operation of the product from the defined mode (normal mode or other) to other mode (Bypass or TAP) in case of system failure.
- **Note**: In case of SET_DIS_BYPASS or SET_STD_NIC is ON the WDT will take no effect when the timer expires and the product will remain in its normal mode.

2.3.7 GET_BYPASS_WD

Description - Get watchdog state and timeout setting, for Bypass and TAP products.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_WD), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***-1*** on failure, if the device is a slave device. ***0*** **-** on success.
bpctl_cmd:data	**bpctl_cmd:data[0]** = *wdt_state*, where **0** - watchdog is disabled, ***-1*** - watchdog state is unknown, ***ms_timeout_set*** - watchdog is configured and running with the timeout setting value in ms units.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: unknown is in a case when the software don't know what is the state of the watchdog timer – this state can happen after software reset to the system without providing hardware reset to the interfaces.

2.3.8 GET_WD_EXPIRE_TIME

Description - Get watchdog time to expire, Bypass and TAP products.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_WD_EXPIRE_TIME), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***-1*** on failure, if the device is a slave device or the watchdog state is unknown ***0*** - on success.
bpctl_cmd:data	**bpctl_cmd:data[0]** = *wdt_state*, where ***0*** - watchdog is disabled, ***-1*** - watchdog timer has expired , ***ms_time_left*** – time left (in ms units) till watchdog time expires.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.9 RESET_BYPASS_WD

Description - Reset watchdog timer, Bypass and TAP products.

This command will inform the watchdog that the system is OK and running and will restart the timer to the value defined in Set watchdog state

Command format:

ioctl(file_desc, IOCTL_TX_MSG(RESET_BYPASS_WD), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***1*** – watchdog timer was reset. ***0*** on failure – if watchdog is not configured, ***-1*** on failure - if the device is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.10 SET_WD_AUTORESET

Description - Enable watchdog timer auto reset periodically, Bypass and TAP products.
This command will cause the software driver to perform periodic reset to the hardware watchdog timer with interval defined by the input parameter in mS, value of "0" to stop resetting.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_WD_AUTORESET), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = requested reset period (in ms units), 0 to stop.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***1*** – watchdog timer periodic resetting has started. ***-1*** on failure - if the device is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.11 GET_WD_AUTORESET

Description - Get status of watchdog timer auto reset periodically, Bypass and TAP products. This command reads the status watchdog auto reset state.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_WD_AUTORESET), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***Ms_reset_pr*** – watchdog timer periodic resetting in ms_reset_pr period ***0*** – watchdog auto reset is disabled, ***-1*** on failure - if the device is a slave device or watchdog is not configured.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.12 SET_TAP

Description - Set TAP state (On/Off)

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_TAP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]**=*tap_mode* (1=on,0=off).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation (not a TAP device) or is a slave device. Bypass only device will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: In case the command SET_DIS_TAP or SET_STD_NIC ON was issued this command will take no effect and the product will remain at Normal operating mode (none TAP mode).

2.3.13 GET_TAP

Description - Get TAP state (On/Off)

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_TAP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** (off/on) on success, ***-1*** on failure - if the device is not capable of the operation (not a TAP device) or is a slave device. Bypass only device will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: TAP ON value will be return whenever it is in TAP mode not related to the cause of it (command or watchdog timer).
  In case of SET_DIS_TAP or SET_STD_NIC is ON the result of GET_TAP will be always OFF.

2.3.14 GET_TAP_CHANGE

Description - Get change to TAP state from last GET_TAP_CHANGE status check

This function enable checking if the device changed to TAP mode, from some reason (WD timeout or any other) from last GET_TAP_CHANGE status check. This can determine if there was a situation that the product went to TAP mode and recovered. This function will enable knowing that this situation has occurred.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_TAP_CHANGE), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** (off/on) on success, ***-1*** on failure - if the device is not capable of the operation (not a TAP device) or is a slave device. Bypass only device will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.15 SET_DISC

Description - Set Disconnect state (On/Off)

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_DISC), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]**=disc_*mode* (1=on,0=off).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: In case the command SET_DIS_DISC or SET_STD_NIC ON was issued this command will take no effect and the product will remain at Normal operating mode (none Disconnect mode).

2.3.16 GET_DISC

Description - Get Disconnect state (On/Off)

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_DISC), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description

bpctl_cmd:status	***0/1*** (off/on) on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: DISC ON value will be return whenever it is in DISC mode not related to the cause of it (command or watchdog timer).
  In case of SET_DIS_DISC or SET_STD_NIC is ON the result of GET_DISC will be always OFF.

2.3.17 GET_DISC_CHANGE

Description - Get change to Disconnect state from last GET_DISC_CHANGE status check

This function enable checking if the device changed to Disconnect mode, from some reason (WD timeout or any other) from last GET_DISC_CHANGE check. This can determine if there was a situation that the product went to Disconnect mode and recovered. This function will enable knowing that this situation has occurred.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_DISC_CHANGE), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** (off/on) on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.18 SET_WD_EXP_MODE

Description - Set product Mode when Watchdog Expire

This command defines the state (Bypass/TAP/DISC) to which the product will elapse if and when the watchdog timer expires.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_WD_EXP_MODE), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = Bypass / TAP / DISC mode (0=Bypass, 1=TAP, 2=DISC)

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation (Bypass only device or TAP only device) or is a slave device. Only products having at least two modes, for example, TAP and Bypass, will support this command.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.19 GET_WD_EXP_MODE

Description - Get product Mode if/when Watchdog Expires

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_WD_EXP_MODE), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1/2*** on success - Bypass/TAP/DISC, ***-1*** on failure - if the device is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.3.20 IF_SCAN

Description - Rescan the list of network interfaces.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(IF_SCAN), pbpctl_cmd);

Request structure fields: None

Returned value: *0*

Returned structure fields: None

2.3.21 GET_DEV_NUM

Description - Get total number of bypass/TAP devices.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_DEV_NUM), pbpctl_cmd);

Request structure fields: None

Returned value: *0*

Returned structure fields:

Field	Description
bpctl_cmd:out_param	**bpctl_cmd:out_param[0]** = *total_dev_num* (total number of bypass/TAP devices).

2.4 Advance Commands

2.4.1 GET_BYPASS_CAPS

Description - Obtain product Bypass/TAP capabilities information

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_CAPS), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***-1*** - if the device is not capable of the operation or is a slave device. ***Flags word*** on success - the returned ***flag word*** is a 32-bit mask word with each bit defines different capability as described in Bypass/TAP capabilities table - Appendix A. Value of "1" for supporting this feature, "0" for not supported.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.2 GET_WD_SET_CAPS

Description - Obtain watchdog timer time setting capabilities

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_WD_SET_CAPS), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***Flag word*** on success - set of numbers defining the various parameters of the watchdog capable to be set to as described in WDT Setting Capability table – Appendix A. ***-1*** - if the device is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.3 SET_STD_NIC

Description - Set as a Standard NIC operation mode

This function, if supported, set the interfaces pair to operate as a Standard NIC adapter at all operating modes.
This means that it will Disable the Bypass/TAP during power-off state and disable Bypass/TAP during power-up. In this mode SET_BYPASS ON and WDT will have not effect and the product will remain in the normal mode of operation as long as SET_STD_NIC is ON.
Disabling the STD_NIC to OFF will return the interfaces to their normal Bypass/TAP operation – which means setting back the interfaces to Bypass/TAP mode during power-off and power-up.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_STD_NIC), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = *nic_mode* (1=Standard NIC mode, 0=Default Bypass mode).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.4 GET_STD_NIC

Description - Get Standard NIC mode setting (On/Off)

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_STD_NIC), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - normal Bypass_TAP operation mode / Standard NIC mode is enabled, ***-1*** on failure - if the device is not capable of the operation or is a slave device or not a Bypass/TAP device
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.5 SET_DIS_BYPASS

Description - Set Disable Bypass mode

This function enable the option of disconnect the "short-circuit" connection between the two ports (disconnect Dis_BP_SW in figure-1) at any time (power on or off) even if a bypass command was issued or due to watchdog time out. This can be used (with the Set_bypass OFF) in case wanting to work as standard NIC without the bypass operation and the command SET_STD_NIC is not supported by the product. See description in Appendix B – Operating as standard NIC description

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_DIS_BYPASS), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = *dis_bypass_mode* (1=dis, 0=en).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device. TAP only will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note: after performing this command a delay of 500mS is required before issuing further commands to the interface. This is due to the time it is taking to this latch relay to become stable.**

2.4.6 GET_DIS_BYPASS

Description - Get Disable Bypass mode

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_DIS_BYPASS), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - normal Bypass mode/ Disable bypass, ***-1*** on failure - if the device is not capable of the operation or is a slave device, TAP only device (no separate function for TAP and Bypass) will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.7 SET_BYPASS_PWOFF

Description - Set Bypass mode at power-off state

This function enables the option of change to normal (non bypass) operation at power-off state - BP_SW1/2 switch at Figure-1 to normal operation mode during power-off state. All other states remain unchanged.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_BYPASS_PWOFF), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = *bypass_pwoff_mode* (Define bypass mode setting at power off state 1=BP en, 0=BP Dis).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device. TAP only will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.8 GET_BYPASS_PWOFF

Description - Get Bypass mode state at power-off state

ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_PWOFF), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - Disable bypass at power off state / normal Bypass mode, ***-1*** on failure - if the device is not capable of the operation or is a slave device, TAP only device (no separate function for TAP and Bypass) will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.9 SET_BYPASS_PWUP

Description - Set Bypass mode at power-up state

This function enables the option of change to normal (non bypass) operation at power-up state - BP_SW1/2 switch at Figure-1 to normal operation mode during power-up state. All other states remain unchanged.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_BYPASS_PWUP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. bpctl_cmd:in_param[2] = bypass_pwup_mode (Define bypass mode setting at power up state 1=BP en, 0=BP Dis).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device. TAP only will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.10 GET_BYPASS_PWUP

Description - Get Bypass mode state at power-UP state

ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_PWUP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - Disable bypass at power up state / normal Bypass mode, ***-1*** on failure - if the device is not capable of the operation or is a slave device, TAP only device (no separate function for TAP and Bypass) will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device.

2.4.11 SET_DIS_TAP

Description - Set Disable TAP mode

This function enable the option of disconnect the "short-circuit" connection between the two ports - disconnect Dis_BP_SW in Figure-1 and the TAP mode at any time (power on or off) even if a SET TAP command was issued or due to watchdog time out. This can be used in case wanting to work as standard NIC without the TAP operation and the command SET_STD_NIC is not supported by the product. See description in Appendix B – Operating as standard NIC description

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_DIS_TAP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. bpctl_cmd:in_param[2] = dis_tap_mode (Disable TAP mode 1=dis, 0=en).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device, Bypass only device will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.12 GET_DIS_TAP

Description - Get state of "Disable TAP"

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_DIS_TAP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - normal TAP mode/ Disable TAP, ***-1*** on failure - if the device is not capable of the operation or is a slave device, Bypass only device will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.13 SET_TAP_PWUP

Description - Set TAP mode at power-up state

This function enables the option of change to normal (non TAP) operation at power-up state - BP_SW1/2 switch at Figure-1 to normal operation mode during power-up state. All other states remain unchanged.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_TAP_PWUP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

	bpctl_cmd:in_param[2] = *tap_pwup_mode* (Define TAP mode setting at power up state 1=TAP en, 0=TAP Dis).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device, Bypass only device (no separate function for TAP and Bypass) will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.14 GET_TAP_PWUP

Description - Get TAP state at power-UP state

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_TAP_PWUP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - Disable TAP at power up process / normal TAP mode, ***-1*** on failure - if the device is not capable of the operation or is a slave device, Bypass only device (no separate function for TAP and Bypass) will also return this value.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.15 SET_DIS_DISC

Description - Set Disable Disconnect mode

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_DIS_DISC), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. bpctl_cmd:in_param[2] = dis_disc_mode (Disable Disc mode 1=dis, 0=en).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.16 GET_DIS_DISC

Description - Get state of “Disable Disconnect”

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_DIS_DISC), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - normal DISC mode/ Disable DISC, ***-1*** on failure - if the device is not capable of the operation or is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.17 SET_DISC_PWUP

Description - Set Disconnect mode at power-up state

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_DISC_PWUP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = disc_*pwup_mode* (Define DISC mode setting at power up state 1=DISC en, 0=DISC Dis).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation or is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.18 GET_DISC_PWUP

Description - Get DISC state at power-UP state

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_DISC_PWUP), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - Disable DISC at power up process / set DISC mode at power up, ***-1*** on failure - if the device is not capable of the operation or is a slave device.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.19 SET_TX

Description - Set Network Transmit ON and OFF

This function give the user the option to disable the PHY transmit on and off (for products supporting this feature). Normal operation is when the TX is enabled. If there is a need to stop the LINK for a specific port and stopping this port data flow, disabling the TX can be used.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_TX), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = *tx_mode* (Set PHY transmit on or off (1=TX en, 0=TX Dis)).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: The product PXG2BPFIL, PXG4BPFI, PXG2TBFI has one control for both ports of the bypass. The control of them will be done via the controlling interface. Disabling the TX will stop the link to both ports of the bypass pair.
- **Note**: Some copper products do not support TX_Dis.

- **Note:** The link LED on the adapter may be on even if TX is disabled. This is due to the fact that the other side continues to transmit but the other side will not have link and there will not be any data flow.

2.4.20 GET_TX

Description - Get Network Transmit state

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_TX), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - TX disabled / TX enabled - normal operating mode, ***-1*** on failure - if the device is not capable of the operation.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.21 SET_TPL

Description – set TPL (Two Port Link) ON and OFF

When enabled, if one of the network ports link fails it will drop the link on the other network port as well.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_TPL), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = tpl_*mode* (1=TPL en, 0=TPL Dis).

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description

bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.22 GET_TPL

Description - Get TPL (Two Port Link) state

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_TPL), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0/1*** on success - TPL disabled / TPL enabled - normal operating mode, ***-1*** on failure - if the device is not capable of the operation.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

2.4.23 GET_BYPASS_INFO

Description – Get Bypass Info

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_INFO), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** – if device is bypass/TAP device (to get detailed status, see returned structure fields), otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success ***-1*** on failure - if the device is not Bypass/TAP device or is a slave device.

bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device **bpctl_cmd:out_param[4]** – firmware version
bpctl_cmd:data	**bpctl_cmd:data[0…8] –** product name

2.4.24 SET_BP_FORCE_LINK

Description – Set Force Link ON

This function give the user the option to disable the PHY transmit on and off (for products supporting this feature). Normal operation is when the TX is enabled. If there is a need to stop the LINK for a specific port and stopping this port data flow, disabling the TX can be used.

Command format:

ioctl(file_desc, IOCTL_TX_MSG(SET_BP_FORCE_LINK), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device. **bpctl_cmd:in_param[2]** = *tx_mode* (Set PHY transmit on or off (1=Force Link en, 0=Force Link Dis)).

Returned value: ***0*** –on success otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***0*** on success, ***-1*** on failure - if the device is not capable of the operation
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: Currently supported only for DBI 10G adapters.

2.4.25 GET_BP_FORCE_LINK

Description - Get Force Link On state

Command format:

ioctl(file_desc, IOCTL_TX_MSG(GET_BP_FORCE_LINK), pbpctl_cmd);

Request structure fields:

Field	Description
bpctl_cmd:in_param	**bpctl_cmd:in_param[0…1]** - requested device.

Returned value: ***0*** –on success; otherwise ***-EOPNOTSUPP.***

Returned structure fields:

Field	Description
bpctl_cmd:status	***1/0*** on success - Force Link On - normal operating mode, ***-1*** on failure - if the device is not capable of the operation.
bpctl_cmd:out_param	**bpctl_cmd:out_param[0…3]** - extended info about requested device

- **Note**: Currently supported only for DBI 10G adapters.

3 Function interface.

This section defines the kernel-level and user-level functions interface that can be used to interface the Bypass/TAP functions.
Together with the drivers provided on the product CD we have included libraries, one for kernel-level interface and one for user-level interface. These libraries located on the product CD under Lib folder for the relevant OS.
Description of the function list is provided in Functions / Command List below.

- **Note:** When Bypass interfaces are down, Function Interface can't be used for BCM57710-based adapters.

3.1 Installing the function library module

In order to compile and install this library module follow the steps below:

User level library :

Installing from TAR file

1. un-pack the libbypass archive file located in folder OS/libs/user/
2. change into libbypass-x.x.x/lib folder
3. type: make install

This will compile and install the library in /usr/local/lib and the *libbypass.h* header file in /usr/include. At this time the lib module is ready to be used.

Installing from RPM file

1. Install the source RPM package:
   rpm -ivh libbypass-<version>.src.rpm
2. CD to the RPM path and build the binary driver for your kernel:
   cd /usr/src/{redhat,OpenLinux,turbo,packages,rpm ..}
   rpm -bb SPECS/libbypass-<version>.spec
   or
   rpmbuild -bb SPECS/libbypass-<version>.spec (for RPM version 4.x.x)
   Note that the RPM path is different for different Linux distributions.

3. Install the newly built package (driver and man page):
   rpm -ivh RPMS/i386/libbypass-<version>.i386.rpm

Kernel level library :

1. un-pack the bypass archive file located in folder OS/libs/kernel/
2. change into bypass-x.x.x/lib folder
3. to install it, type: **make install**
   This will compile and install the library in /lib/modules/<kernel_version>/kernel/lib/bypass and the *libbypass.h* header file in /<kernel_source_path>/include
4. to load it, type**: modprobe bypass**

At this time the lib module is ready to be used.

Note: In order to use these libraries need also to install **bp_ctl** package file located in folder OS/BP_Control.

For samples of the function calls please look at Appendix E – User level function - description and examples and Appendix F – Kernel level function - description and examples

3.2 Bypass/TAP support detection

This section describes how the software can distinguish between Silicom network interface supporting Bypass/TAP functionality and regular network interface that may be also installed in the system.
The software can, before accessing to the Bypass/TAP functions check if this network interface supports Silicom Bypass/TAP functions with the IS_BYPASS command as describe below. Silicom Bypass product will respond with success for this command. Standard product (not supporting bypass/TAP) will not recognize this command and respond with “not supported” response. This detection of the Bypass/TAP support is done by the library module that checks the interface device ID to be one of Silicom Bypass/TAP product member.

3.3 Basic Commands

3.3.1 is_bypass()

Description - Check if device is a Bypass/TAP controlling device

Syntax:

int is_bypass(int ifindex);

Input: Kernel network device index - ***ifindex***.
Output: ***1*** if the device is bypass/TAP controlling device,
0 if the device is bypass/TAP slave device.

3.3.2 get_bypass_slave()

Description - Get the second port participate in the Bypass/TAP pair

Syntax:

int get_bypass_slave(int ifindex);

Input: Kernel network device index - ***ifindex***.

Output: ***-1*** on failure - if the device is not capable of the operation or is a slave device or not a Bypass/TAP device,
Kernel network device index on success - ***ifindex*** of the slave device of the Bypass/TAP ports pair.

3.3.3 set_bypass()

Description - Set Bypass On/Off mode

Syntax:

int set_bypass(int ifindex,
int bypass_mode);

Input: Index of the controlling device - ***ifindex***, bypass mode (1=on, 0=off)
Output: ***0*** on success,
-1 on failure - if the device is not capable of the operation or is a slave device or is not a Bypass device. TAP only device (no separate function for TAP and Bypass) will also return this value.

- **Note**: this command control the state of BP_SW1/2 state without any relation to the Dis_BP_SW switch state at Figure-1.
  In case of SET_DIS_BYPASS or SET_STD_NIC is ON the product will not switch to bypass even if SET_BYPASS ON command will be issued.

3.3.4 get_bypass()

Description - Get Bypass state (On/Off)

Syntax:

int get_bypass(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success – Bypass off/on,
-1 if the device is not capable of the operation or is a slave device or is not a Bypass device. TAP only device (no separate function for TAP and Bypass) will also return this value.

- **Note**: Bypass on will return whenever it is in bypass mode not related to the cause of it (command or watchdog timer). In case of SET_DIS_BYPASS or SET_STD_NIC is ON the product will not switch to bypass (even if GET_BYPASS command will show that it is ON).

3.3.5 get_bypass_change()

Description - Get change in Bypass mode state from last bypass_change status check

This function enable checking if the device changed to bypass mode, from some reason (WD timeout or any other) from last bypass_change status check. This can determine if there was a situation that the product went to bypass mode and recovered. This function will enable knowing that this situation has occurred.

Syntax:

int get_bypass_change(int ifindex);

Input: Index of the controlling device - ***ifindex***,

Output: ***0/1*** on success – Bypass change off/on,
-1 on failure - if the device is not capable of the operation or is a slave device or is not a Bypass device. TAP only device (no separate function for TAP and Bypass) will also return this value.

3.3.6 set_bypass_wd()

Description - Set watchdog timer on/Off and timeout, for Bypass and TAP products.

Syntax:

```
int set_bypass_wd(int ifindex,
                  int ms_timeout
                  int * ms_timeout_set);
```

Input: Index of the controlling device - ***ifindex***,
requested timeout (in ms units), 0 for disabling the watchdog timer.
Output: ***-1*** if the device is not capable of the operation or is a slave device or is not a Bypass/TAP device,
0 on success.
Ms_timeout_set:
the actual value (in ms units) that the adapter supports and will be used by the watchdog.
0 – watchdog is disabled.

- **Note**: The returned timeout value (in ms units) is for the caller to check and adjust its timeout value in case of mismatch to the requested value. The returned value will be the nearest one available that is equal or larger from the requested value but not larger then the maximum value it support.

- **Note**: The WDT intend to switch the operation of the product from the defined mode (normal mode or other) to other mode (Bypass or TAP) in case of system failure.
- **Note**: In case of SET_DIS_BYPASS or SET_STD_NIC is ON the WDT will take no effect when the timer expires and the product will remain in its normal mode.

3.3.7 get_bypass_wd()

Description - Get watchdog state and timeout setting, for Bypass and TAP products.

Syntax:

```
int get_bypass_wd (     int ifindex,
                        int * ms_timeout_set);
```

Input: Index of the controlling device - ***ifindex***,
Output: ***-1*** if the device is not capable of the operation or is a slave device or is not a Bypass/TAP device
0 – on success,
1 –watchdog is configured and running.
ms_timeout_set:
watchdog is configured and running with the timeout setting of the int value in ms units.
0 – watchdog is not configured, disabled,
-1 if watchdog state is unknown

- **Note**: unknown is in a case when the software don't know what is the state of the watchdog timer – this state can happen after software reset to the system without providing hardware reset to the interfaces.

3.3.8 get_wd_expire_time()

Description - Get watchdog time to expire, Bypass and TAP products.

Syntax:

int get_wd_expire_time(int ifindex,
int * ms_time_left);

Input: Index of the controlling device - ***ifindex***,
Output: -1 if the device is not capable of the operation or is a slave device or is not a Bypass/TAP device or the WD_Timer state is unknown,
0 – watchdog is disabled,
1 – watchdog is configured and running.
ms_time_left:
time left (in ms units) till watchdog time expire.
-1 – watchdog timer has expired

3.3.9 reset_bypass_wd()

Description - Reset watchdog timer, Bypass and TAP products.

This command will inform the watchdog that the system is OK and running and will restart the timer to the value defined in Set watchdog state

Syntax:

int reset_bypass_wd_timer(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***-1*** if the device is not capable of the operation or is a slave device or is not a Bypass/TAP device
0 – watchdog is not configured
1 – watchdog timer was reset

3.3.10 set_wd_autoreset()

Description - Enable watchdog timer auto reset periodically, Bypass and TAP products.

This command will cause the software driver to perform periodic reset to the hardware watchdog timer with interval defined by the input parameter in mS. Value of “0” will stop the resetting.

Syntax:

int set_wd_autoreset(int ifindex
int ms_reset_pr);

Input: Index of the controlling device - ***ifindex***,
ms_reset_pr - requested reset period (in ms units), value of 0 will stop the WD timer resetting.
Output: ***-1*** if the device is not capable of the operation or is a slave device or is not a Bypass/TAP device
1 – watchdog timer periodic resetting has started

3.3.11 get_wd_autoreset()

Description – Get status of watchdog timer auto reset periodically, Bypass and TAP products.

This command reads the status watchdog auto reset state.

Syntax:

int get_wd_autoreset(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***-1*** if the device is not capable of the operation or is a slave device or is not a Bypass/TAP device.
0 – watchdog auto reset is disabled
ms_reset_pr – watchdog timer periodic resetting in ms_reset_pr period.

3.3.12 set_tap()

Description - Set TAP state (On/Off)

Syntax:

int set_tap(int ifindex,
int TAP_mode);

Input: Index of the controlling device - ***ifindex***, TAP mode (1=on, 0=off)
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device or is not a TAP device. Bypass only device will also return this value.

- **Note**: In Case the command SET_DIS_TAP or SET_STD_NIC ON was issued this command will take no effect and the product will remain at Normal operating mode (none TAP mode).

3.3.13 get_tap()

Description - Get TAP state (On/Off)

Syntax:

int get_tap(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success – TAP off/on,
-1 if the device is not capable of the operation or is a slave device or is not a TAP device. Bypass only device will also return this value.

- **Note**: TAP ON value will be returned whenever it is in TAP mode not related to the cause of it (command or watchdog timer).
In case of SET_DIS_TAP or SET_STD_NIC is ON the result of GET_TAP will be always OFF.

3.3.14 get_tap_change()

Description - Get change to TAP state from last tap_change status check

This function enable checking if the device changed to TAP mode, from some reason (WD timeout or any other) from last tap_change status check. This can determine if there was a

situation that the product went to TAP mode and recovered. This function will enable knowing that this situation has occurred.

Syntax:

int get_tap_change(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success – TAP change off/on,
-1 if the device is not capable of the operation or is a slave device or is not a TAP device. Bypass only device will also return this value.

3.3.15 set_bp_disc()

Description - Set Disconnect state (On/Off)

Syntax:

int set_bp_disc(int ifindex,
int disc_mode);

Input: Index of the controlling device - ***ifindex***, Disconnect mode (1=on, 0=off)
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device.

- **Note**: In case the command SET_DIS_DISC or SET_STD_NIC ON was issued this command will take no effect and the product will remain at Normal operating mode (none DISC mode).

3.3.16 get_bp_disc()

Description - Get Disconnect state (On/Off)

Syntax:

int get_bp_disc(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success – Disconnect off/on,
-1 if the device is not capable of the operation or is a slave device.

- **Note**: DISC ON value will be returned whenever it is in DISC mode not related to the cause of it (command or watchdog timer).
  In case of SET_DIS_DISC or SET_STD_NIC is ON the result of GET_DISC will be always OFF.

3.3.17 get_bp_disc_change()

Description - Get change to Disconnect state from last get_disc_change status check

This function enable checking if the device changed to DISC mode, from some reason (WD timeout or any other) from last get_disc_change check. This can determine if there was a situation that the product went to DISC mode and recovered. This function will enable knowing that this situation has occurred.

Syntax:

int get_bp_disc_change(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success – DISC change off/on,
-1 if the device is not capable of the operation or is a slave device.

3.3.18 set_wd_exp_mode()

Description - Set product Mode when Watchdog Expire

This command defines the state (Bypass/TAP/Disconnect) to which the product will elapse if and when the watchdog timer expires. This feature supported only for products supporting at least two modes, for example TAP and Bypass.

Syntax:

int set_wd_exp_mode(int ifindex,
int wd_exp_mode);

Input: Index of the controlling device - ***ifindex***, Bypass / TAP / DISC mode (0=Bypass, 1=TAP, 2=DISC)
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device. Only products having at least two modes will support this command.

3.3.19 get_wd_exp_mode()

Description - Get product Mode if/when Watchdog Expire

Syntax:

int get_wd_exp_mode(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1/2*** on success – WDT expires to Bypass/TAP/DISC mode,
-1 if the device is not capable of the operation or is a slave device. Only products having at least two modes will support this command.

3.4 Advance Commands

3.4.1 get_bypass_caps()

Description - Obtain product Bypass/TAP capabilities information

Syntax:

int get_bypass_caps(int ifindex);

Input: Kernel network device index - ***ifindex***.
Output: ***-1*** if the device is not capable of the operation or is a slave device or not a Bypass/TAP device.
Flags word on success - The returned ***flag word*** is a 32-bit mask word with each bit defines different capability as described in **Bypass/TAP capabilities table** - Appendix A.
Value of "1" for supporting this feature. "0" for not supported.

3.4.2 get_wd_set_caps()

Description - Obtain watchdog timer time setting capabilities

Syntax:

int get_wd_set_caps (int ifindex);

Input: Kernel network device index - ***ifindex***.
Output: ***-1*** if the device is not capable of the operation or is a slave device or not a Bypass device,
Flag word on success - Set of numbers defining the various parameters of the watchdog capable to be set to as described in **WDT Setting Capability table** – Appendix A.

3.4.3 set_std_nic()

Description - Set as a Standard NIC operation mode

This function, if supported, set the interfaces pair to operate as a Standard NIC adapter at all operating modes.
This means that it will Disable the Bypass/TAP during power-off state and disable Bypass during power-up. In this mode SET_BYPASS ON and WDT will have not effect and the product will remain in the normal mode of operation as long as SET_STD_NIC is ON.
Disabling the STD_NIC to OFF will return the interfaces to their normal Bypass/TAP operation – which means setting back the interfaces to Bypass/TAP mode during power-off and power-up.

Syntax:

int set_std_nic(int ifindex,
int std_nic);

Input: Index of the controlling device - ***ifindex***,
0/1 (Default Bypass mode / Standard NIC mode)
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device or is not a Bypass device

3.4.4 get_std_nic()

Description - Get Standard NIC mode setting (On/Off)

Syntax:

int get_std_nic(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - normal Bypass_TAP mode / Standard NIC is enabled,
-1 on failure - if the device is not capable of the operation or is a slave device or is not a Bypass/TAP device

3.4.5 set_dis_bypass()

Description - Set Disable Bypass mode

This function enable the option of disconnect the "short-circuit" connection between the two ports (disconnect Dis_BP_SW in figure-1) at any time (power on or off) even if a bypass command was issued or due to watchdog time out. This can be used (with the SET_BYPASS OFF) in case wanting to work as standard NIC without the bypass operation and the command SET_STD_NIC is not supported by the product. See description in Appendix B – Operating as standard NIC description

Syntax:

int set_dis_bypass(int ifindex,
int dis_bypass);

Input: Index of the controlling device - ***ifindex***, disable bypass (1=dis, 0=en)
Output: ***0*** on success,
-1 on failure - if the device is not capable of the operation or is a slave device. TAP only device (no separate function for TAP and Bypass) will also return this value.

- **Note: after performing this command a delay of 500mS is required before issuing further commands to the interface. This is due to the time it is taking to this latch relay to become stable.**

3.4.6 get_dis_bypass()

Description - Get Disable Bypass mode

Syntax:

int get_dis_bypass(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success – normal Bypass mode / Disable bypass,
-1 on failure - if the device is not capable of the operation or is a slave device. TAP only device (no separate function for TAP and Bypass) will also return this value.

3.4.7 set_bypass_pwoff()

Description - Set Bypass mode at power-off state

This function enables the option of change to normal (non bypass) operation at power-off state - BP_SW1/2 switch at Figure-1 to normal operation mode during power-off state. All other states remain unchanged.

Syntax:

int set_bypass_pwoff(int ifindex,
int bypass_pwoff);

Input: Index of the controlling device- ***ifindex***, Define bypass mode setting at power off state (1=BP en, 0=BP Dis)
Output: ***0*** on success,
-1 on failure - if the device is not capable of the operation or is a slave device. TAP only device (no separate function for TAP and Bypass) will also return this value.

3.4.8 get_bypass_pwoff()

Description - Get Bypass mode state at power-off state

Syntax:

int get_bypass_pwoff(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - Disable bypass at power off state / normal Bypass mode,
-1 on failure - if the device is not capable of the operation or is a slave device. TAP only device (no separate function for TAP and Bypass) will also return this value.

3.4.9 set_bypass_pwup()

Description - Set Bypass mode at power-up state

This function enables the option of change to normal (non bypass) operation at power-up state - BP_SW1/2 switch at Figure-1 to normal operation mode during power-up state. All other states remain unchanged.

Syntax:

int set_bypass_pwup(int ifindex,
int bypass_pwup);

Input: Index of the controlling device - ***ifindex***,
Enable/disable bypass at power up process (1=en, 0=dis)
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device or is not a Bypass device

3.4.10 get_bypass_pwup()

Description - Get Bypass mode state at power-UP state

Syntax:

int get_bypass_pwup(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - Disable bypass at power up state / normal Bypass mode,

-1 on failure - if the device is not capable of the operation or is a slave device. TAP only device (no separate function for TAP and Bypass) will also return this value.

3.4.11 set_dis_tap()

Description - Set Disable TAP mode

This function enable the option of disconnect the "short-circuit" connection between the two ports -disconnect Dis_BP_SW in figure-1 and the TAP mode at any time (power On or Off) even if a SET TAP command was issued or due to watchdog time out. This can be used in case wanting to work as standard NIC without the TAP operation and the command SET_STD_NIC is not supported by the product. See description in Appendix B – Operating as standard NIC description

Syntax:

int set_dis_tap(int ifindex,
int dis_tap);

Input: Index of the controlling device - ***ifindex***, disable tap (1=dis, 0=en)
Output: ***0*** on success,
-1 on failure - if the device is not capable of the operation or is a slave device. Bypass only device (no separate function for TAP and Bypass) will also return this value.

3.4.12 get_dis_tap()

Description - Get state of "Disable TAP"

Syntax:

int get_dis_tap(int ifindex);
Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - normal TAP mode/ Disable TAP,
-1 if the device is not capable of the operation or is a slave device. Bypass only device (no separate function for TAP and Bypass) will also return this value.

3.4.13 set_tap_pwup()

Description - Set TAP mode at power-up state

This function enables the option of change to normal (non TAP) operation at power-up state - BP_SW1/2 switch at Figure-1 to normal/non-TAP operation mode during power-up state. All other states remain unchanged.

Syntax:

int set_tap_pwup(int ifindex,
int tap_pwup);

Input: Index of the controlling device - ***ifindex***,
Enable/disable TAP at power up process (1=en, 0=dis)
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device or is not a TAP device

3.4.14 get_tap_pwup()

Description - Get TAP state at power-UP state

Syntax:

int get_tap_pwup(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - Disable TAP at power up state / normal TAP mode,
-1 if the device is not capable of the operation or is a slave device. Bypass only device (no separate function for TAP and Bypass) will also return this value.

3.4.15 set_bp_dis_disc()

Description - Set Disable Disconnect mode

Syntax:

int set_bp_dis_disc(int ifindex,
int dis_disc);

Input: Index of the controlling device - ***ifindex***, disable Disconnect (1=dis, 0=en)
Output: ***0*** on success,
-1 on failure - if the device is not capable of the operation or is a slave device.

3.4.16 get_bp_dis_disc()

Description - Get state of "Disable Disconnect"

Syntax:

int get_bp_dis_disc(int ifindex);
Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - enable DISC mode/ Disable DISC,
-1 if the device is not capable of the operation or is a slave device.

3.4.17 set_bp_disc_pwup()

Description - Set Disconnect mode at power-up state

Syntax:

int set_bp_disc_pwup(int ifindex,
int disc_pwup);

Input: Index of the controlling device - ***ifindex***,
Enable/disable Disconnect at power up process (1=en, 0=dis)
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device.

3.4.18 get_bp_disc_pwup()

Description - Get Disconnect state at power-UP state

Syntax:

int get_bp_disc_pwup(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - Disable Disconnect at power up state / Disconnect mode at power up state, ***-1*** if the device is not capable of the operation or is a slave device.

3.4.19 set_tx()

Description - Set Network Transmit ON and OFF

This function give the user the option to disable the PHY transmit on and off (for products supporting this feature). Normal operation is when the TX is enabled. If there is a need to stop the LINK for a specific port and stopping this port data flow, disabling the TX can be used.

Syntax:

int set_tx(int ifindex, int tx);

Input: Index of the controlling device - ***ifindex***,
tx - enable/disable PHY transmit (1=en, 0=dis)
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device or is not a Bypass device
Description - Set Network Transmit ON and OFF

This function give the user the option to disable the PHY transmit on and off (for products supporting this feature). Normal operation is when the TX is enabled. If there is a need to stop the LINK for a specific port and stopping this port data flow, disabling the TX can be used.

- Note: The product PXG4BPFI has one control for both ports of the bypass. The control of them will be done via the controlling interface. Disabling the TX will stop the link to both ports of the bypass pair.
- Note: When "Auto Negotiation" mode is disable (force 10 or 100, half/full duplex) in Copper adapters the link LED on the adapter will be on due to the fact that the other side continues to transmit, but the other side will not have link and there will not be any data flow. The same is with Fiber products.

3.4.20 get_tx()

Description - Get Network Transmit state

Syntax:

int get_tx(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - TX Disabled / TX enabled - normal operation mode,
-1 if the device is not capable of the operation or is a slave device or is not a Bypass device

3.4.21 set_tpl()

Description - Set TPL (Two Port Link) ON and OFF

Syntax:

int set_tpl(int ifindex, int tpl);

Input: Index of the controlling device - ***ifindex***,
tpl - enable/disable TPL (1=en, 0=dis)
Output: ***0*** on success,

-1 if the device is not capable of the operation or is a slave device or is not a Bypass device

3.4.22 get_tpl()

Description - Get TPL (Two Port Link)

Syntax:

int get_tpl(int ifindex);

Input: Index of the controlling device - ***ifindex***,
Output: ***0/1*** on success - TPL Disabled / TPL enabled,
-1 if the device is not capable of the operation or is a slave device.

3.4.23 get_bypass_info()

Description - Get Network Transmit state

Syntax:

int get_bypass_info(int ifindex, struct *bp_info bp_info);

Input: Index of the controlling device - ***ifindex***,
Output: ***0*** on success,
-1 if the device is not capable of the operation or is a slave device or is not a Bypass device;

Bypass Info parameters are passed in the bp_info structure. The fields of the bp_info structure are shown below:

```
struct bp_info {
   char prod_name[14];
   unsigned char fw_ver;
};
```

Field	Description
bp_info:prod_name	Product name
bp_info:fw_ver	Firmware version

4 Proc interface.

This section defines script-level interface for the user to check status and configure the different bypass parameters. It can be done by issuing specific command via Proc interface or with running a script to perform set of commands to the interfaces. Proc interface is in general producing virtual files in a specific folder defined here below. Accessing to these virtual folders, read or write, will send commands to the interface according to the virtual file accessed and will read or write the defined parameter.

4.1 Commands location

After Bypass driver installation and bringing the interface up, Bypass/TAP control commands can be found at:

/proc/net/bypass/bypass_ethX (X –Bypass controlling interface index)

4.2 Bypass/TAP support detection

Proc folder will exist for each controlling Bypass/TAP interface. None Bypass/TAP interfaces or slave interfaces will not have proc folder.
Finding the slave interface for a controlling interface will be done using get bypass slave (bypass_slave) command as described below.

4.3 Basic Commands

4.3.1 BYPASS_SLAVE - Get the second port participate of the Bypass/TAP pair

Command:

 # cat /proc/net/bypass/bypass_ethX/bypass_slave

Output:

"**ethY"** (where Y is the index of the slave interface for ethX controlling interface)
"**fail**" – if device is not capable of the operation

Writing to this function will take no effect.

4.3.2 BYPASS – Set/Get Bypass mode state

With this file you can set the bypass mode on and off and read its state.

Setting Bypass ON
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/bypass

Setting Bypass OFF
Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/bypass

Reading Bypass state:
Command:

 # cat /proc/net/bypass/bypass_ethX/bypass

Output:

"**on**" when Bypass is ON.
"**off**" when Bypass is OFF.

4.3.3 TAP – Set/Get TAP mode state

With this function you can set the TAP mode on and off and read its state.

Setting TAP ON
Command:
 # echo "on" > /proc/net/bypass/bypass_ethX/tap

Setting TAP OFF
Command:
 # echo "off" > /proc/net/bypass/bypass_ethX/tap

Reading TAP mode
Command:
 # cat /proc/net/bypass/bypass_ethX/tap
Output:
"**on**" when TAP is ON.
"**off**" when TAP is OFF.

4.3.4 DISC – Set/Get Disconnect mode state

With this function you can set the Disconnect mode on and off and read its state.

Setting Disconnect ON
Command:
 # echo "on" > /proc/net/bypass/bypass_ethX/disc

Setting Disconnect OFF
Command:
 # echo "off" > /proc/net/bypass/bypass_ethX/disc

Reading Disconnect mode
Command:
 # cat /proc/net/bypass/bypass_ethX/disc
Output:
"**on**" when Disconnect is ON.
"**off**" when Disconnect is OFF.

4.3.5 BYPASS_CHANGE - Get change of Bypass state from last status check

Command:
 # cat /proc/net/bypass/bypass_ethX/bypass_change
Output:
"**on**" when product is in bypass mode.
"**off**" when product is in – non bypass mode.
"**fail**" when the device do no support this function

Description: This function, if supported, enable checking if the device changed to bypass mode, from some reason (WD timeout or any other reason) from last bypass_change status check. This can enable the user to determine if there was a situation that the product went to bypass mode and recovered. This function will enable knowing that this situation has occurred.

*__Note__: this operation has to be performed before performing Bypass read command. Bypass command clear the Bypass_Change status.

Writing to this function will take no effect.

4.3.6 TAP_CHANGE - Get change of TAP state from last status check

Command:
 # cat /proc/net/bypass/bypass_ethX/tap_change
Output:

"**on**" If device was changes to TAP state or it is now in TAP mode.
"off" if the product didn't change to TAP mode from the last read of this command.

Description This function, if supported, enable checking if the device changed to TAP mode, from some reason (WD timeout or any other reason) from last tap_change status check. This can enable the user to determine if there was a situation that the product went to bypass mode and recovered. This function will enable knowing that this situation has occurred.

*__Note__: this operation has to be performed before performing tap read command. tap command clear the tap_Change status.

Writing to this function will take no effect.

4.3.7 DISC_CHANGE - Get change of Disconnect state from last status check

Command:
 # cat /proc/net/bypass/bypass_ethX/disc_change
Output:

"**on**" If device was changes to Disconnect state or it is now in Disconnect mode.
"off" if the product didn't change to Disconnect mode from the last read of this command.

Description This function, if supported, enable checking if the device changed to Disconnect mode, from some reason (WD timeout or any other reason) from last disc_change status check. This can enable the user to determine if there was a situation that the product went to Disconnect mode and recovered. This function will enable knowing that this situation has occurred.

Writing to this function will take no effect.

4.3.8 BYPASS_WD – Configure Bypass/TAP Watchdog timer setting / reading its state

This function enables controlling the Watchdog timer setting and reading its configuration and state.

Setting Watchdog timer OFF:
Command:
 # echo "0" > /proc/net/bypass/bypass_ethX/bypass_wd

Setting Watchdog timer to X ms
Command:
 # echo "X" > /proc/net/bypass/bypass_ethX/bypass_wd

Reading Watchdog timer setting:
Command:
 # cat /proc/net/bypass/bypass_ethX/bypass_wd
Output:
"<**ms_timeout_set**>" actual value (in ms units) the watchdog timer is set to.
"**disable**" when the Watchdog timer is disable
"**unknown**" when the watchdog timer status is unknown.

4.3.9 WD_EXPIRE_TIME - Get watchdog expired time (Bypass and TAP)

Command:
 # cat /proc/net/bypass/bypass_ethX/wd_expire_time
Output:
"**<ms_time_left>**" actual time left (in ms units) till watchdog timer expires.
"**disable**" when watchdog is not configured, disabled,
"**fail**" when the watchdog timer status is unknown.

Writing to this function will take no effect.

4.3.10 RESET_BYPASS_WD - Reset watchdog timer (Bypass and TAP)

Command:
 # cat /proc/net/bypass/bypass_ethX/reset_bypass_wd
Output:
"**success**" when operation succeeded.
"**disable**" when watchdog is not configured, disabled

Writing to this function will take no effect.

4.3.11 WD_AUTO RESET - Reset watchdog timer periodically (Bypass and TAP)

This command will cause the software driver to perform periodic reset to the hardware watchdog timer with interval defined by the input parameter in mS, value of 0 will stop the WDT resetting.

Resetting Watchdog timer with X ms period, 0 will stop WDT resetting.
Command:
 # echo "X" > /proc/net/bypass/bypass_ethX/wd_autoreset

Reading watchdog timer AutoReset state and setting:
Command:
 # cat /proc/net/bypass/bypass_ethX/wd_autoreset
Output:
"**AutoReset period in ms**" when operation succeeded.
"**fail**" on error.

4.3.12 WD_EXP_MODE – Set/Get Adapter state when WD expire

This function Set/read the state to which the product will elapse to when the watchdog timer expires.

Set WD timer to switch to bypass mode when WD timer expires:
Command:

 # echo "bypass" > /proc/net/bypass/bypass_ethX/wd_exp_mode

Set WD timer to switch to TAP mode when WD timer expires:
Command:

 # echo "tap" > /proc/net/bypass/bypass_ethX/wd_exp_mode

Set WD timer to switch to Disconnect mode when WD timer expires:
Command:

 # echo "disc" > /proc/net/bypass/bypass_ethX/wd_exp_mode

Reading wd_exp_mode state:
Command:

 # cat /proc/net/bypass/bypass_ethX/wd_exp_mode

Output:

"**bypass**" when WD is set to expire to Bypass mode.
"**tap**" when WD is set to expire to TAP mode.
"**disc**" when WD is set to expire to Disconnect mode.

4.4 Advance Commands

4.4.1 BYPASS_CAPS - Obtain Bypass/TAP capabilities information

Command:

 # cat /proc/net/bypass/bypass_ethX/bypass_caps

Output:

"<Flags word>" - is a 32-bit mask word with each bit defines different capability. Value of "1" when supporting this feature, "0" when the feature is not supported.

Writing to this function will take no effect.

4.4.2 WD_SET_CAPS - Obtain watchdog timer setting capabilities

Command:

 # cat /proc/net/bypass/bypass_ethX/wd_set_caps

Output:

"<Flags word>" - Set of numbers defining the various parameters of the watchdog capable to be set to as described in WDT Setting Capability table – Appendix A.

Writing to this function will take no effect.

4.4.3 STD_NIC - Set/Get Normal NIC mode

Setting as Standard NIC operation:

Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/std_nic

Setting back to Bypass/TAP operation:

Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/std_nic

Reading STD_NIC state:

Command:

 # cat /proc/net/bypass/bypass_ethX/std_nic

Output:

"**on**" when in Standard NIC mode of operation.
"**off**" when in normal Bypass/TAP operating mode.

Description: This function, if supported, enables setting the interfaces pair to operate as a Standard NIC adapter at all operating modes.
This means that it will disable the Bypass/TAP during power-off state and during power-up. In this mode SET_BYPASS ON and WDT will have not effect and the product will remain in the normal mode of operation as long as SET_STD_NIC is ON. Disabling the STD_NIC to OFF will return the interfaces to their normal Bypass/TAP mode during power-off and power-up.

4.4.4 DIS_BYPASS – Control and display the state of the DIS_BP_SW switches

This function controls the state of the Disable Bypass Switch (see figure-1) and read its state.

Disabling bypass at all state
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/dis_bypass

Enabling back the bypass at all state
Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/dis_bypass

Reading DIS_BYPASS state:
Command:
 # cat /proc/net/bypass/bypass_ethX/dis_bypass
Output:

"**on**" when the "Disable Bypass" Switch is ON.
"**off**" when the "Disable Bypass" Switch is OFF.
"**fail**" when this operation is not supported

Description: This function enable the option of disconnect the "short-circuit" connection between the two ports (disconnect Dis_BP_SW in figure-1) at any time (power is on or off) even if a bypass command was issued or due to watchdog time out. This can be used (with the Set_bypass OFF) in case wanting to work as standard NIC without the bypass operation and the command SET_STD_NIC is not supported by the product. See description in Appendix B – Operating as standard NIC description

- **Note: after performing this command a delay of 500mS is required before issuing further commands to the interface. This is due to the time it is taking to these latch relay to become stable.**

4.4.5 DIS_TAP – Control and display the state of the DIS_BP_SW switches

This function controls the state of the Disable Bypass/TAP Switch (see figure-1) and read its state.

Disabling TAP at all state
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/dis_tap

Enabling back the TAP at all state
Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/dis_tap

Reading DIS_TAP state
Command:
 # cat /proc/net/bypass/bypass_ethX/dis_tap
Output:

"**on**" when the "Disable TAP" Switch is ON.
"**off**" when the "Disable TAP" Switch is OFF.
"**fail**" when this operation is not supported.

Description This function enable the option of disconnect the "short-circuit" connection between the two ports (disconnect Dis_BP_SW in figure-1) at any time (power is on or off) even if a TAP command was issued or due to watchdog time out. This can be used in case

wanting to work as standard NIC without the TAP operation and the command SET_STD_NIC is not supported by the product. See description in Appendix B – Operating as standard NIC description

4.4.6 DIS_DISC – Set/Get Disable Disconnect mode state.

Disabling Disconnect at all state
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/dis_disc

Enabling back the Disconnect at all state
Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/dis_disc

Reading DIS_DISC state
Command:
 # cat /proc/net/bypass/bypass_ethX/dis_disc
Output:

"**on**" when Disconnect mode is disabled.
"**off**" when Disconnect mode is not disabled.
"**fail**" when this operation is not supported.

4.4.7 BYPASS_PWUP - Configure Bypass mode at power-up state

Setting Bypass mode at Power-up state
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/bypass_pwup

Setting normal mode (non Bypass) mode at Power-up state
Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/bypass_pwup

Reading BYPASS_PWUP state
Command:

 # cat /proc/net/bypass/bypass_ethX/bypass_pwup

Output:

"**on**" when configure to be in Bypass mode at power-up state.
"**off**" when configure to be in normal mode at power-up state.
"**fail**" when the device does not support this function

Description: This function, if supported, enables controlling the state of the Bypass switches at power-up state. (controlling the state of BP_SW1/2 switch mode to normal/bypass mode during power-up state) All other states remain unchanged.

4.4.8 BYPASS_PWOFF - Configure Bypass mode at power-off state

Setting Bypass mode at Power-off state
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/bypass_pwoff

Setting normal mode (non Bypass) mode at Power-off state
Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/bypass_pwoff

Reading BYPASS_PWOFF state
Command:

 # cat /proc/net/bypass/bypass_ethX/bypass_pwoff

Output:

"**on**" when configure to be in Bypass mode at power-off state.
"**off**" when configure to be in normal mode at power-off state.
"**fail**" when the device does not support this function

Description: This function, if supported, enables controlling the state of the Bypass switches at power-up state. (controlling the state of BP_SW1/2 switch mode to normal/bypass mode during power-up state) All other states remain unchanged.

4.4.9 TAP_PWUP - Configure TAP mode at power-up state

Setting TAP mode at Power-up state
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/tap_pwup

Setting normal mode (non TAP) mode at Power-up state
Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/tap_pwup

Reading TAP_PWUP state
Command:

 # cat /proc/net/bypass/bypass_ethX/tap_pwup

Output:

"**on**" when configure to be in Bypass mode at power-up state.
"**off**" when configure to be in normal mode at power-up state.

This function, if supported, enables the option of change to normal (non TAP) operation at power-up state - BP_SW1/2 switch at Figure-1 to normal/non TAP operation mode during power-up state. All other states remain unchanged.

4.4.10 DISC_PWUP - Configure Disconnect mode at power-up state

Setting Disconnect mode at Power-up state
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/disc_pwup

Setting non Disconnect mode at Power-up state
Command:

 # echo "off" > /proc/net/bypass/bypass_ethX/disc_pwup

Reading DISC_PWUP state
Command:

 # cat /proc/net/bypass/bypass_ethX/disc_pwup

Output:

"**on**" when configure to be in Disconnect mode at power-up state.
"**off**" when configure to be in non-Disconnect mode at power-up state.

This function, if supported, enables the option of change to non Disconnect operation mode during power-up state. All other states remain unchanged.

4.4.11 TPL – Set/Get TPL mode state

With this function you can set the Two Port Link (TPL) mode on and off and read its state.

Setting TPL ON
Command:

 # echo "on" > /proc/net/bypass/bypass_ethX/tpl

Setting TPL OFF
Command:
 # echo "off" > /proc/net/bypass/bypass_ethX/tpl

Reading TPL mode
Command:
 # cat /proc/net/bypass/bypass_ethX/tpl
Output:

"**on**" when TPL is ON.
"**off**" when TPL is OFF

4.4.12 BYPASS_INFO – Get detailed info about Bypass/TAP device

Command:

 # cat /proc/net/bypass/bypass_ethX/bypass_info

Output:

Displays the product name and firmware version number .

Writing to this function will take no effect.

5 APPENDIX

5.1 Appendix A - Products and their command responses

This section describes how the different Silicom Bypass/TAP products will respond to the different commands and the capabilities each product have and support. If the product support this feature and what are the responses for each command can be.

The responses are for accessing the controlling interface. Responses when accessing Slave interface will be “-1” unless defined otherwise.

Bypass/TAP Command Responses table

Bypass/TAP capabilities table

WDT Setting Capability table

Bit	Feature	description	Products
0-3	WD_MIN_TIME	The interface WD minimal time period in 100mS units	PXG2/4BP – 5 (500mS) All the other products – 1(100mS)
4	WD_STEP_TIME	The steps of the WD timer in 0 – for linear steps (WD_MIN_TIME * X) 1 – for multiply by 2 from previous step (WD_MIN_TIME * 2^X)	PXG2/4BP – 0 All the other products – 1
5-8	WD_STEP_COUNT	Number of register bits available for configuring the WD timer. From that the number of steps the WD timer will have is 2^X (X number of bits available for defining the value)	PXG2/4BP –7 All the other products – 4
9-31	RESERVED	Reserved, should be ignored.	

5.2 Appendix B – Operating as standard NIC

These product series where designed to work as a bypass/TAP products. As so they will be in Bypass/TAP state at the following states: Power-off, Power-up operating state until configured differently.

This will change the operation of the product from Bypass/TAP to normal mode when the product driver is loaded and enable working with the product normally. No additional command (via the different software interfaces) is needed.

To disable the bypass mode at all operating states the use of SET_STD_NIC can be used. If this command is not supported then SET_DIS_BYPASS(on) (or SET_DIS_TAP for TAP products) command need to be issued via the software interface.

5.3 Appendix C – Steps for operating the BYPASS/TAP products

The Bypass/TAP product interfaces will start operating in the "Bypass_product_default_states" as defined in definition section – will be in Bypass/TAP mode at power-off, power-up and during its operating until a change will be done to change it from this state.
The steps for operating the Bypass/TAP interfaces will be as follows:

1 Bypass/TAP interfaces detection

These steps are needed in cases where there are mixed environment of interfaces that are supporting Bypass/TAP and standard interfaces that are not supporting Bypass/TAP. In this case the application/module, if it do not have this info, has to detect and distinguish between them

a. Detect the interfaces which are Bypass/TAP capable.
   IOCTL interface:
   This is done by checking the product ID(PID and VID) for bypass product ID's and IS_BYPASS call. For more details see in IOCTL Bypass/TAP support detection
   Function interface:
   This is done by using **is_bypass()** function. For more details see in Function interface – Bypass/TAP support detection
b. Detect the slave interface - the interface associate with the controlling interface.
   IOCTL interface:
   This is done by GET_BYPASS_SLAVE call issued to the controlling interface. For more details see in IOCTL API - GET_BYPASS_SLAVE
   Function interface:
   This is done by **get_bypass_slave()** function using. For more details in Function interface - get_bypass_slave

2 Bypass/TAP/Disconnect Capabilities detection

These steps are needed in order to find out the capabilities the Bypass/TAP product support. This is needed in cases where there are mix environment of different Bypass/TAP products each supporting different features – see in Appendix A - Products and their command responses for the different

products capabilities. All the product support the basic capabilities (Bypass/TAP/Disconnect on/off, watchdog timer) and if there is no intention using advanced feature then there is no need to read these capabilities detection.

a. Check product capabilities:

   This is done by GET_BYPASS_CAPS call to the controlling interface. This will return a 32-bit mask word with different capabilities, For more details see in GET_BYPASS_CAPS (IOCTL interface) or GET_BYPASS_CAPS (function interface) call.

b. Check product watchdog timer setting capabilities to get the time setting supported by the product. This is done by issuing GET_WD_SET_CAPS. For details see command description at GET_WD_SET_CAPS (IOCTL interface) or GET_WD_SET_CAPS (function interface).

3 Configuring Bypass/TAP/Disconnect operation

These steps will configure the Bypass/TAP interfaces to the desired operation of the bypass/TAP/Disconnect function. The most common use is setting the interfaces to normal operation and enabling watchdog timer to set the product to Bypass/TAP/Disconnect when system fails.

a. Set the product to normal operating mode (no Bypass/TAP/DISC) by issuing SET_BYPASS/TAP/DISC(off) to the controlling interface, verify the return value for success.
b. Set the watchdog timer to the desired value. This is done by issuing SET_BYPASS_WD(required_time). Check the returned value for success and for the real assigned timer value.
c. Start resetting the timer periodically in interval that is shorter then the assigned timer value. This is done by issuing RESET_BYPASS_WD periodically.

4 Monitoring Bypass/TAP/Disconnect operation

These steps will enable monitoring the status of the Bypass/TAP operation and give indication of its status.

a. Get the Bypass/TAP/DISC change status (if the product was in bypass mode last status check). This is done by issuing GET_BYPASS/TAP_CHANGE/DISC_CHANGE command. Return value of 1 will indicate that the product went to Bypass/TAP/Disconnect mode from last status check.
b. Get current Bypass/TAP/DISC mode by issuing GET_BYPASS/TAP/DISC command. Returned value of 1 indicates that the product is currently in bypass mode.
c. Get status of the watchdog timer. This is done by issuing GET_WD_EXPIRE_TIME. The returned value will indicate the time left till the watchdog timer will expire, value of -1 indicates that it is already expired.

The above steps will initialize and configure the Bypass/TAP interface for the basic operation. These steps have to be performed once after system boot-up. The RESET_BYPASS_WD has to be performed periodically as long as the system is working.

5.4 Appendix D - User level IOCTL - description and examples

The IOCTL API is a standard common used API for user level application to access kernel module. Below are several samples of how to implement accesses to this API via user level application.
We have included also sample application describing how these accesses are implemented. Also this sample application can help in testing and understanding the product operation.
This sample application – BPCTL is included with the product software CD under Util folder of the relevant OS.

1 IOCTL Configuration examples

This section gives some simple examples for the code needed to perform IOCTL access to the Bypass/TAP IOCTL API.

```
/* for passing single values */
struct bpctl_cmd {
    int status;
    int data[8];
    int in_param[8];
    int out_param[8];
};

#define MAGIC_NUM 'J'

#define IOCTL_TX_MSG(cmd) _IOWR(MAGIC_NUM, cmd, struct bpctl_cmd *)
```

Note: cmd – from CMND_TYPE enum.

```
#define DEVICE_NODE "/dev/bpctl0"
```

#define DEVICE_NAME "bpctl"

Bypass IOCTL API user-level command format:

```
file_desc = open(DEVICE_NODE, 0);
ioctl(file_desc, IOCTL_TX_MSG(cmd), pbpctl_cmd);
```

fike_desc - control socket descriptor;
pbpctl_cmd - pointer to bpctl_cmd structure.

Common used commands for Bypass/TAP operation will be:

1.1 IS_Bypass

```
int ret_val, dev_num, i;
struct bpctl_cmd bpctl_cmd;

 bpctl_cmd.in_param[1]=if_index;
 ret_val = ioctl(file_desc, IOCTL_TX_MSG(IS_BYPASS), &bpctl_cmd);
 if ((ret_val < 0)||(bpctl_cmd.status <0)) {
       printf(NOT_SUPP_BPT_MSG);
       exit(-1);
 }
 if (bpctl_cmd.status==0)
          printf("interface is not a bypass controlling device\n");
   else  printf("interface is a bypass controlling device\n");
```

1.2 GET_BYPASS_SLAVE

```
int ret_val;
   struct bpctl_cmd bpctl_cmd;
   char if_name[IFNAMSIZ];

   bpctl_cmd.in_param[1]=if_index;

   ret_val = ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_SLAVE), &bpctl_cmd);
   if ((ret_val < 0)||(bpctl_cmd.status<0)) {
      printf(NOT_SUPP_BPT_MSG);
      exit(-1);
   }
   if (bpctl_cmd.status==0) {
      printf(SLAVE_IF_MSG);
      return;
   }
   if ((bpctl_cmd.status==1)&&
      (if_indextoname(bpctl_cmd.out_param[7], (char *)&if_name)!=NULL))
      printf("slave interface is %s", if_name);
   else {
      printf("fail\n");
      return;
   }
```

1.3 GET_BYPASS_CAP

```
int ret_val,i;
struct bpctl_cmd bpctl_cmd;
char if_name[IFNAMSIZ];

bpctl_cmd.in_param[1]=if_index;
ret_val = ioctl(file_desc, IOCTL_TX_MSG(GET_BYPASS_CAPS), &bpctl_cmd);
```

The returned value will define bypass interface different capabilities in a 32-bit mask word as defined in the GET_BYPASS_CAP command returned value.

1.4 SET_BYPASS OFF

```
int ret_val,dev_num,i;
   struct bpctl_cmd bpctl_cmd;

   bpctl_cmd.in_param[1]=if_index;
   bpctl_cmd.in_param[2]=tx_state;

   ret_val = ioctl(file_desc, IOCTL_TX_MSG(SET_BYPASS), &bpctl_cmd);
   if ((ret_val == 0)&&(bpctl_cmd.status==0)) {
      printf(operation succeeded\n);
   } else {
       printf("operation fail, interface is not a bypass interface or is a slave device\n");
   }
```

1.5 SET_BYPASS Watchdog Timer

```
int ret=0, timeout_set=0;
   int ret_val;
   struct bpctl_cmd bpctl_cmd;

   bpctl_cmd.in_param[1]=if_index;
   bpctl_cmd.in_param[2]=timeout;

   ret_val = ioctl(file_desc, IOCTL_TX_MSG(SET_BYPASS_WD), &bpctl_cmd);
   if ((ret_val == 0)&&(bpctl_cmd.status>=0)) {
      if (!bpctl_cmd.status)
         printf("operation succeeded watchdog timer was disabled\n");
      else if (bpctl_cmd.status>0)
          printf(("operation succeeded watchdog timer set to %d ms\n", bpctl_cmd.status);

   } else {
      printf("operation fail, interface is not a bypass interface or is a slave device\n");

   }
```

5.5 Appendix E – User level function - description and examples

The user level function interface is available with the use of the libbypass library module provided with the product software CD. Description of its functionality and its installation is described in Function interface section 3.

NOTE: Please make sure that you installed the function library package before installing this sample application.

Installing the sample application package – bpctl:

Installing from TAR file

To install the bpctl, do the following:

1. un-pack the bpctl archive file located in folder OS/Samples/User/
2. change into bpctl-<version> folder
3. run: make
4. run: make install

Installing from RPM file

1. Install the source RPM package:
   rpm -ivh bpctl-<version>.src.rpm
2. CD to the RPM path and build the binary driver for your kernel:
   cd /usr/src/{redhat,OpenLinux,turbo,packages,rpm ..}
   rpm -bb SPECS/bpctl-<version>.spec
   or
   rpmbuild -bb SPECS/bpctl-<version>.spec (for RPM version 4.x.x)
   Note that the RPM path is different for different Linux distributions.
3. Install the newly built package (driver and man page):
   rpm -ivh RPMS/i386/bpctl-<version>.i386.rpm

Below are samples of how to access this lib with user level application. We have provided also sample application – bpctl provided with the product software CD under /os/samples/user folder of the relevant OS.

```
/*
 * User level function using sample console application
*/
int main(int argc, char **argp, char **envp){
                .
/*
* Command line parsing
*/
char ifname[IFNAMSIZ];
int ifindex = if_nametoindex("eth0"); /* get ifindex for eth0 */

/* Send a specific command by library using*/
```

```
/* IS_BYPASS command:    */
   ret= is_bypass(ifindex);      /* to eth0  */
   if (ret==1)
      printf("interface is bypass controlling device\n");
   else
      printf("interface is not a bypass controlling device\n");


/* GET_BYPASS_SLAVE command:       */
   ret= get_bypass_slave(ifindex);     /* to eth0
   if(ret>0)
       printf("slave interface for eth0 is %s\n",if_indextoname(ret, (char *)&ifname));

/* SET_BYPASS off command:      */
   ret= set_bypass(ifindex,0); /* (interface eth0, 0=off) */
   if (!ret)
      printf("operation succeeded\n");
   else
      printf("operation fail, interface is not a bypass interface or is a slave device\n");

/* SET_BYPASS_WD command:   */
   ret= set_bypass_wd(ifindex,5000,&ms_timeout_set);      /* (interface eth0, 5000ms) */
   if (!ret)
      printf("operation succeeded, watchdog timer set to %d ms\n",ms_timout_set);
   else
      printf("operation fail, interface is not a bypass interface or is a slave device\n");

/* GET_WD_EXPIRE_TIME command:  */
   ret= get_wd_expire_time(ifindex,&ms_time_left);          /* (interface eth0) */
   if (!ret)
      printf("watchdog timer is disabled\n")
   else
   if (ret==1)
      printf("watchdog timer will expire in %d ms\n",ms_time_left);
   else
      printf("operation fail, interface is not a bypass interface or is a slave device\n");

/* RESET_BYPASS_WD_TIMER command:      */
   ret= reset_bypass_wd_timer(ifindex);       /* (interface eth0) */
   if (!ret)
      printf("watchdog timer is not configured\n")
   else if (ret==1)
      printf("watchdog timer was reset successfully\n");
   else
      printf("operation fail, interface is not a bypass interface or is a slave device\n");
}
```

5.6 Appendix F – Kernel level function - description and examples

The kernel level function interface is available with the use of the bypass library module provided with the product software CD. Description of its functionality and its installation is described in Function interface, section 3.

NOTE: Please make sure that you installed the kernel function library package before installing this sample application.

Installing the sample kernel module package – bptest:

NOTE: Please make sure that you installed the kernel function library package before installing this sample module.

To build and install this sample, do the following:

1. un-pack the bptest archive file located in folder OS/Samples/Kernel/
2. change into bptest-x.x.x folder
3. run: make install

To use it please type:
insmod ./bptest.o (2.4.x kernels)
insmod ./bptest.ko (2.6.x kernels)

Installing the sample application package – bptest_app:

NOTE: Please make sure that you installed the kernel function library package and kernel sample kernel module package before installing this sample application.
To build and install this sample, do the following:

1. un-pack the bptest_app archive file located in folder OS/Samples/Kernel/
2. change into bptest_app-x.x.x folder
3. run: make

To use it please type:
bptest_app help

Below are samples of how to access this lib with kernel level application. We have provided also sample application – bptest and bptest_app provided with the product software CD under the \os\samples\kernel folder of the relevant OS.

```
/*
* Send a specific command by library using
*/

struct net_device *dev;
int ifindex;

dev=dev_get_by_name("eth0");
ifindex=dev->ifindex;
```

```
/* IS_BYPASS command:     */

   ret= is_bypass(ifindex);
   if (ret==1)
      printk("interface is bypass controlling device\n");
   else
      printk("interface is not a bypass controlling device\n");

/* GET_BYPASS_SLAVE command:          */
   ret= get_bypass_slave(ifindex);
   if(ret>0)
       printk("slave interface for eth0 is %s\n",(dev_get_by_index(ret))->name);

/* SET_BYPASS off command:        */
   ret= set_bypass(ifindex,0); /* ( 0=off) */
   if (!ret)
      printk("operation succeeded\n");
   else
      printk("operation fail, interface is not a bypass interface or is a slave device\n");

/* SET_BYPASS_WD command:     */
   ret= set_bypass_wd(ifindex,5000,&ms_timeout_set);      /* ( 5000ms) */
   if (!ret)
      printk("operation succeeded, watchdog timer set to %d ms\n",ms_timout_set);
   else
      printk("operation fail, interface is not a bypass interface or is a slave device\n");

/* GET_WD_EXPIRE_TIME command:     */
   ret= get_wd_expire_time(ifindex,&ms_time_left);          /* (interface eth0) */
   if (!ret)
      printk("watchdog timer is disabled\n")
   else
   if (ret==1)
      printk("watchdog timer will expire in %d ms\n",ms_time_left);
   else
      printk("operation fail, interface is not a bypass interface or is a slave device\n");

/* RESET_BYPASS_WD_TIMER command:         */
   ret= reset_bypass_wd_timer(ifindex);
   if (!ret)
      printk("watchdog timer is not configured\n")
   else if (ret==1)
      printk("watchdog timer was reset successfully\n");
   else
      printk("operation fail, interface is not a bypass interface or is a slave device\n");
```

5.7 Appendix G – PROC interface examples

Below are some samples for common used command. This shows how the PROC interface can be used to control the Bypass/TAP operation.

5.7.1 Standard initial configuration example on operating the Bypass/TAP product at interface eth0:

```
#!/bin/bash

cd  /proc/net/bypass/bypass_eth0
echo Disable Watchdog ...
echo "0" >bypass_ wd
 bypass_string=`cat bypass_wd`
 if test $bypass_string != "disable"; then
    echo FAIL
    exit 1
 fi
echo SUCCESS
echo Change Bypass to Normal mode ...
echo "off" >bypass
bypass_string=`cat bypass`
if test $bypass_string != "off";then
    echo FAIL
    exit 1
fi
echo SUCCESS
echo Setting watchdog to 10Sec ...
echo "10000" >bypass_ wd
bypass_string=`cat  bypass_wd`
if test $bypass_string = "fail" -o $bypass_string = "unknown"; then
    echo FAIL
    exit 1
fi
echo SUCCESS
#
# for every ms_time_set*0.7 do reset watchdog
#
timeout=`expr $bypass_string \* 7 / 10 / 1000`
while test 1
do
  echo Reset Bypass watchdog ...
  bypass_string=`cat reset_bypass_wd`
  if test $bypass_string != "success"; then
     echo FAIL
     exit 1
 fi
```

```
  echo SUCCESS
  sleep $timeout
done
```

5.7.2 Disabling watchdog timer of interface eth0:

```
#!/bin/bash

cd  /proc/net/bypass/bypass_eth0
echo Disable Watchdog ...
echo "0" >bypass_ wd
 bypass_string=`cat bypass_wd`
 if test $bypass_string = "disable"; then
   echo SUCCESS
 else
   echo FAIL
   exit 1
 fi
```

5.8 Appendix H – BPCTL_UTIL (IOCTL interface sample program) description

bpctl_util program is a simple Bypass Control utility, which interfaces to Bypass/TAP Driver through IOCTL calls.
This application is included with the product software CD for the relevant OS.
This sample program can be used to help writing customer specific application program that will fit to its own need and will control the Bypass/TAP functionality.
Below is description of how to use and operate this sample program.

1 Compiling and using the software

In order to compile and run this sample program, follow the steps below:

Installing from TAR file

1. Un-pack the bp_ctl-xxx archive file
2. Change into bp_ctl-xxx folder
3. Type: make install
4. Type: bpctl _start

Installing from RPM file

1. Install the source RPM package:
   rpm -ivh bp_ctl-<version>.src.rpm
2. CD to the RPM path and build the binary driver for your kernel:
   cd /usr/src/{redhat,OpenLinux,turbo,packages,rpm ..}
   rpm -bb SPECS/bp_ctl-<version>.spec
   or

rpmbuild -bb SPECS/bp_ctl-<version>.spec (for RPM version 4.x.x)
Note that the RPM path is different for different Linux distributions.

3. Install the newly built package (driver and man page):
   rpm -ivh RPMS/i386/bp_ctl-<version>.i386.rpm
4. Type: bpctl_start

This will compile and place the program in /bin folder so it can be used from any folder.

2 Usage

bpctl_util <if_index|bus:slot.function> <command> [parameters]
bpctl_util <info|help>
<if_index> - interface name, for example, eth0, or all for all Bypass-SD/TAP-SD Control devices
<command> - bypass control command (see Commands List).
[parameters] - set_bypass_wd command:
WDT timeout interval, msec (0 for disabling WDT).
set_bypass/set_bypass_pwoff/set_bypass_pwup/set_dis_bypass commands:
on/off for enable/disable Bypass
set_std_nic command:
on/off for enable/disable Standard NIC mode
set_tx command:
on/off for enable/disable transmit
set_tpl command:
on/off for enable/disable TPL
set_force_link_on command:
on/off for enable/disable force_link_on
set_hw_reset command:
on/off for enable/disable hw_reset
set_tap/set_tap_pwup/set_dis_tap commands:
on/off for enable/disable TAP
set_disc/set_disc_pwup/set_dis_disc commands:
on/off for enable/disable Disc
set_wd_exp_mode command:
bypass/tap/disc for bypass/tap/disc mode
set_wd_autoreset command:
WDT autoreset interval, msec (0 for disabling WDT autoreset).
info - print Program Information.
help - print this message.
Commands List:
is_bypass - check if device is a Bypass/TAP controlling device
get_bypass_slave - get the second port participate in the Bypass/TAP pair
get_bypass_caps - obtain Bypass/TAP capabilities information
get_wd_set_caps - obtain watchdog timer settings capabilities
get_bypass_info - get bypass/TAP info
set_bypass - set Bypass mode state
get_bypass - get Bypass mode state
get_bypass_change - get change of Bypass mode state from last status check

set_dis_bypass - set Disable Bypass mode
get_dis_bypass - get Disable Bypass mode state
set_bypass_pwoff - set Bypass mode at power-off state
get_bypass_pwoff - get Bypass mode at power-off state
set_bypass_pwup - set Bypass mode at power-up state
get_bypass_pwup - get Bypass mode at power-up state
set_std_nic - set Standard NIC mode of operation
get_std_nic - get Standard NIC mode settings
set_bypass_wd - set watchdog state
get_bypass_wd - get watchdog state
get_wd_time_expire - get watchdog expired time
reset_bypass_wd - reset watchdog timer
set_tx - set transmit enable / disable
get_tx - get transmitter state (enabled / disabled)
set_tpl - set TPL enable / disable
get_tpl - get TPL state (enabled / disabled)
set_hw_reset - set hw_reset enable / disable
get_hw_reset - get hw_reset (enabled / disabled)
set_force_link_on - set force_link_on enable / disable
get_force_link_on - get force_link_on (enabled / disabled)
set_tap - set TAP mode state
get_tap - get TAP mode state
get_tap_change - get change of TAP mode state from last status check
set_dis_tap - set Disable TAP mode
get_dis_tap - get Disable TAP mode state
set_tap_pwup - set TAP mode at power-up state
get_tap_pwup - get TAP mode at power-up state
set_disc - set Disc mode state
get_disc - get Disc mode state
get_disc_change - get change of Disc mode state from last status check
set_dis_disc - set Disable Disc mode
get_dis_disc - get Disable Disc mode state
set_disc_pwup - set Disc mode at power-up state
get_disc_pwup - get Disc mode at power-up state
set_wd_exp_mode - set adapter state when WDT expired
get_wd_exp_mode - get adapter state when WDT expired
set_wd_autoreset - set WDT autoreset mode
get_wd_autoreset - get WDT autoreset mode

Example: bpctl_util eth0 set_bypass_wd 5000
bpctl_util all set_bypass on
bpctl_util eth0 set_wd_exp_mode tap
bpctl_util 0b:00.0 get_bypass_info

Sample commands:

2.1 Set bypass on

bpctl_util eth0 set_bypass on

2.2 Set bypass off for all Bypass/Tap devices

bpctl_util all set_bypass off

2.3 Enable watchdog with 10Sec refresh timeout

bpctl_util eth0 set_bypass_wd 10000

2.4 Refresh active watchdog timer

bpctl_util 0b:00.0 reset_bypass_wd