NAME

i2cget – read from I2C/SMBus chip registers

SYNOPSIS

i2cget [**-f**] [**-y**] [**-a**] *i2cbus chip-address* [*data-address* [*mode*]]
i2cget -V

DESCRIPTION

i2cget is a small helper program to read registers visible through the I2C bus (or SMBus).

OPTIONS

-V Display the version and exit.

-f Force access to the device even if it is already busy. By default, i2cget will refuse to access a device which is already under the control of a kernel driver. Using this flag is dangerous, it can seriously confuse the kernel driver in question. It can also cause i2cget to return an invalid value. So use at your own risk and only if you know what you're doing.

-y Disable interactive mode. By default, i2cget will wait for a confirmation from the user before messing with the I2C bus. When this flag is used, it will perform the operation directly. This is mainly meant to be used in scripts. Use with caution.

-a Allow using addresses between 0x00 - 0x02 and 0x78 - 0x7f. Not recommended.

There are two required options to i2cget. *i2cbus* indicates the number or name of the I2C bus to be scanned. This number should correspond to one of the busses listed by *i2cdetect -l*. *chip-address* specifies the address of the chip on that bus, and is an integer between 0x03 and 0x77.

data-address specifies the address on that chip to read from, and is an integer between 0x00 and 0xFF. If omitted, the currently active register will be read (if that makes sense for the considered chip).

The *mode* parameter, if specified, is one of the letters **b**, **w** or **c**, corresponding to a read byte data, a read word data or a write byte/read byte transaction, respectively. A **p** can also be appended to the *mode* parameter to enable PEC. If the *mode* parameter is omitted, i2cget defaults to a read byte data transaction, unless *data-address* is also omitted, in which case the default (and only valid) transaction is a single read byte.

WARNING

i2cget can be extremely dangerous if used improperly. I2C and SMBus are designed in such a way that an SMBus read transaction can be seen as a write transaction by certain chips. This is particularly true if setting *mode* to **cp** (write byte/read byte with PEC). Be extremely careful using this program.

EXAMPLES

Get the value of 8-bit register 0x11 of the I2C device at 7-bit address 0x2d on bus 1 (i2c-1), after user confirmation:

```
# i2cget 1 0x2d 0x11
```

Get the value of 16-bit register 0x00 of the I2C device at 7-bit address 0x48 on bus 1 (i2c-1), after user confirmation:

```
# i2cget 1 0x48 0x00 w
```

Set the internal pointer register of a 24C02 EEPROM at 7-bit address 0x50 on bus 9 (i2c-9) to 0x00, then read the first 2 bytes from that EEPROM:

```
# i2cset -y 9 0x50 0x00 ; i2cget -y 9 0x50 ; i2cget -y 9 0x50
```

This assumes that the device automatically increments its internal pointer register on every read, and supports read byte transactions (read without specifying the register address, "Receive Byte" in SMBus terminology.) Most EEPROM devices behave that way. Note that this is only safe as long as nobody else is accessing the I2C device at the same time. A safer approach would be to use a "Read Word" SMBus transaction instead, or an I2C Block Read transaction to read more than 2 bytes.

Set the internal pointer register of a 24C32 EEPROM at 7-bit address 0x53 on bus 9 (i2c-9) to 0x0000, then

read the first 2 bytes from that EEPROM:

```
# i2cset -y 9 0x53 0x00 0x00 ; i2cget -y 9 0x53 ; i2cget -y 9 0x53
```

This again assumes that the device automatically increments its internal pointer register on every read, and supports read byte transactions. While the previous example was for a small EEPROM using 8-bit internal addressing, this example is for a larger EEPROM using 16-bit internal addressing. Beware that running this command on a small EEPROM using 8-bit internal addressing would actually *write* 0x00 to the first byte of that EEPROM. The safety concerns raised above still stand, however in this case there is no SMBus equivalent, so this is the only way to read data from a large EEPROM if your master isn't fully I2C capable. With a fully I2C capable master, you would use *i2ctransfer* to achieve the same in a safe and faster way.

SEE ALSO

i2cdetect(8), i2cdump(8), i2cset(8), i2ctransfer(8)

AUTHOR

Jean Delvare

This manual page was strongly inspired from those written by David Z Maze for i2cset.