2.1 Option Descriptions

AVRDUDE is a command line tool, used as follows:

```
avrdude -p partno options …
```

Command line options are used to control AVRDUDE’s behaviour. The following options are recognized:

`-p partno`

This is the only mandatory option and it tells AVRDUDE what type of part (MCU) that is connected to the programmer. The *partno* parameter is the part’s id listed in the configuration file. Specify -p ? to list all parts in the configuration file. If a part is unknown to AVRDUDE, it means that there is no config file entry for that part, but it can be added to the configuration file if you have the Atmel datasheet so that you can enter the programming specifications. Currently, the following MCU types are understood:

`uc3a0512`	AT32UC3A0512
`c128`	AT90CAN128
`c32`	AT90CAN32
`c64`	AT90CAN64
`pwm2`	AT90PWM2
`pwm2b`	AT90PWM2B
`pwm3`	AT90PWM3
`pwm316`	AT90PWM316
`pwm3b`	AT90PWM3B
`1200`	AT90S1200 (****)
`2313`	AT90S2313
`2333`	AT90S2333
`2343`	AT90S2343 (*)
`4414`	AT90S4414
`4433`	AT90S4433
`4434`	AT90S4434
`8515`	AT90S8515
`8535`	AT90S8535
`usb1286`	AT90USB1286
`usb1287`	AT90USB1287
`usb162`	AT90USB162
`usb646`	AT90USB646

usb647	AT90USB647
usb82	AT90USB82
m103	ATmega103
m128	ATmega128
m1280	ATmega1280
m1281	ATmega1281
m1284p	ATmega1284P
m1284rfr2	ATmega1284RFR2
m128rfa1	ATmega128RFA1
m128rfr2	ATmega128RFR2
m16	ATmega16
m161	ATmega161
m162	ATmega162
m163	ATmega163
m164p	ATmega164P
m168	ATmega168
m168p	ATmega168P
m169	ATmega169
m16u2	ATmega16U2
m2560	ATmega2560 (**)
m2561	ATmega2561 (**)
m2564rfr2	ATmega2564RFR2
m256rfr2	ATmega256RFR2
m32	ATmega32
m324p	ATmega324P
m324pa	ATmega324PA
m325	ATmega325
m3250	ATmega3250
m328	ATmega328
m328p	ATmega328P
m329	ATmega329
m3290	ATmega3290
m3290p	ATmega3290P
m329p	ATmega329P
m32u2	ATmega32U2
m32u4	ATmega32U4
m406	ATMEGA406
m48	ATmega48
m48p	ATmega48P
m64	ATmega64
m640	

	ATmega640
m644	ATmega644
m644p	ATmega644P
m644rfr2	ATmega644RFR2
m645	ATmega645
m6450	ATmega6450
m649	ATmega649
m6490	ATmega6490
m64rfr2	ATmega64RFR2
m8	ATmega8
m8515	ATmega8515
m8535	ATmega8535
m88	ATmega88
m88p	ATmega88P
m8u2	ATmega8U2
t10	ATtiny10
t11	ATtiny11
t12	ATtiny12
t13	ATtiny13
t15	ATtiny15
t1634	ATtiny1634
t20	ATtiny20
t2313	ATtiny2313
t24	ATtiny24
t25	ATtiny25
t26	ATtiny26
t261	ATtiny261
t4	ATtiny4
t40	ATtiny40
t4313	ATtiny4313
t43u	ATtiny43u
t44	ATtiny44
t45	ATtiny45
t461	ATtiny461
t5	ATtiny5
t84	ATtiny84
t85	ATtiny85
t861	ATtiny861
t88	ATtiny88
t9	ATtiny9
x128a1	

	ATxmega128A1
x128a1d	ATxmega128A1revD
x128a1u	ATxmega128A1U
x128a3	ATxmega128A3
x128a3u	ATxmega128A3U
x128a4	ATxmega128A4
x128a4u	ATxmega128A4U
x128b1	ATxmega128B1
x128b3	ATxmega128B3
x128c3	ATxmega128C3
x128d3	ATxmega128D3
x128d4	ATxmega128D4
x16a4	ATxmega16A4
x16a4u	ATxmega16A4U
x16c4	ATxmega16C4
x16d4	ATxmega16D4
x16e5	ATxmega16E5
x192a1	ATxmega192A1
x192a3	ATxmega192A3
x192a3u	ATxmega192A3U
x192c3	ATxmega192C3
x192d3	ATxmega192D3
x256a1	ATxmega256A1
x256a3	ATxmega256A3
x256a3b	ATxmega256A3B
x256a3bu	ATxmega256A3BU
x256a3u	ATxmega256A3U
x256c3	ATxmega256C3
x256d3	ATxmega256D3
x32a4	ATxmega32A4
x32a4u	ATxmega32A4U
x32c4	ATxmega32C4
x32d4	ATxmega32D4
x32e5	ATxmega32E5
x384c3	ATxmega384C3
x384d3	ATxmega384D3
x64a1	ATxmega64A1
x64a1u	ATxmega64A1U
x64a3	ATxmega64A3
x64a3u	ATxmega64A3U
x64a4	

	ATxmega64A4
x64a4u	ATxmega64A4U
x64b1	ATxmega64B1
x64b3	ATxmega64B3
x64c3	ATxmega64C3
x64d3	ATxmega64D3
x64d4	ATxmega64D4
x8e5	ATxmega8E5
ucr2	deprecated,

(*) The AT90S2323 and ATtiny22 use the same algorithm.

(**) Flash addressing above 128 KB is not supported by all programming hardware. Known to work are jtag2, stk500v2, and bit-bang programmers.

(***) The ATtiny11 can only be programmed in high-voltage serial mode.

(****) The ISP programming protocol of the AT90S1200 differs in subtle ways from that of other AVRs. Thus, not all programmers support this device. Known to work are all direct bitbang programmers, and all programmers talking the STK500v2 protocol.

-b *baudrate*

Override the RS-232 connection baud rate specified in the respective programmer’s entry of the configuration file.

-B *bitclock*

Specify the bit clock period for the JTAG interface or the ISP clock (JTAG ICE only). The value is a floating-point number in microseconds. The default value of the JTAG ICE results in about 1 microsecond bit clock period, suitable for target MCUs running at 4 MHz clock and above. Unlike certain parameters in the STK500, the JTAG ICE resets all its parameters to default values when the programming software signs off from the ICE, so for MCUs running at lower clock speeds, this parameter must be specified on the command-line. It can also be set in the configuration file by using the ’default_bitclock’ keyword.

-c *programmer-id*

Specify the programmer to be used. AVRDUDE knows about several common programmers. Use this option to specify which one to use. The *programmer-id* parameter is the programmer’s id listed in the configuration file. Specify -c ? to list all programmers in the configuration file. If you have a programmer that is unknown to AVRDUDE, and the programmer is controlled via the PC parallel port, there’s a good chance that it can be easily added to the configuration file without any code changes to AVRDUDE. Simply copy an existing entry and change the pin definitions to match that of the unknown programmer. Currently, the following programmer ids are understood and supported:

2232HIO	FT2232H based generic programmer
4232h	FT4232H based generic programmer

89isp	Atmel at89isp cable
abcmini	ABCmini Board, aka Dick Smith HOTCHIP
alf	Nightshade ALF-PgmAVR, http://nightshade.homeip.net/
arduino	Arduino
arduino-ft232r	Arduino: FT232R connected to ISP
atisp	AT-ISP V1.1 programming cable for AVR-SDK1 from <http://micro-research.co.th/>
avr109	Atmel AppNote AVR109 Boot Loader
avr910	Atmel Low Cost Serial Programmer
avr911	Atmel AppNote AVR911 AVROSP
avrftdi	FT2232D based generic programmer
avrisp	Atmel AVR ISP
avrisp2	Atmel AVR ISP mkII
avrispmkII	Atmel AVR ISP mkII
avrispv2	Atmel AVR ISP V2
bascom	Bascom SAMPLE programming cable
blaster	Altera ByteBlaster
bsd	Brian Dean’s Programmer, http://www.bsdhome.com/avrdude/
buspirate	The Bus Pirate
buspirate_bb	The Bus Pirate (bitbang interface, supports TPI)
butterfly	Atmel Butterfly Development Board
butterfly_mk	Mikrokopter.de Butterfly
bwmega	BitWizard ftdi_atmega builtin programmer
c2n232i	serial port banging, reset=dtr sck=!rts mosi=!txd miso=!cts
dapa	Direct AVR Parallel Access cable
dasa	serial port banging, reset=rts sck=dtr mosi=txd miso=cts
dasa3	serial port banging, reset=!dtr sck=rts mosi=txd miso=cts
diecimila	alias for arduino-ft232r
dragon_dw	Atmel AVR Dragon in debugWire mode
dragon_hvsp	Atmel AVR Dragon in HVSP mode
dragon_isp	Atmel AVR Dragon in ISP mode
dragon_jtag	Atmel AVR Dragon in JTAG mode
dragon_pdi	Atmel AVR Dragon in PDI mode
dragon_pp	Atmel AVR Dragon in PP mode
dt006	Dontronics DT006
ere-isp-avr	ERE ISP-AVR <http://www.ere.co.th/download/sch050713.pdf>
frank-stk200	Frank STK200
ft232r	FT232R Synchronous BitBang
ft245r	FT245R Synchronous BitBang
futurlec	Futurlec.com programming cable.
jtag1	Atmel JTAG ICE (mkI)

`jtag1slow`	Atmel JTAG ICE (mkI)
`jtag2`	Atmel JTAG ICE mkII
`jtag2avr32`	Atmel JTAG ICE mkII im AVR32 mode
`jtag2dw`	Atmel JTAG ICE mkII in debugWire mode
`jtag2fast`	Atmel JTAG ICE mkII
`jtag2isp`	Atmel JTAG ICE mkII in ISP mode
`jtag2pdi`	Atmel JTAG ICE mkII PDI mode
`jtag2slow`	Atmel JTAG ICE mkII
`jtag3`	Atmel AVR JTAGICE3 in JTAG mode
`jtag3dw`	Atmel AVR JTAGICE3 in debugWIRE mode
`jtag3isp`	Atmel AVR JTAGICE3 in ISP mode
`jtag3pdi`	Atmel AVR JTAGICE3 in PDI mode
`jtagkey`	Amontec JTAGKey, JTAGKey-Tiny and JTAGKey2
`jtagmkI`	Atmel JTAG ICE (mkI)
`jtagmkII`	Atmel JTAG ICE mkII
`jtagmkII_avr32`	Atmel JTAG ICE mkII im AVR32 mode
`lm3s811`	Luminary Micro LM3S811 Eval Board (Rev. A)
`mib510`	Crossbow MIB510 programming board
`mkbutterfly`	Mikrokopter.de Butterfly
`nibobee`	NIBObee
`o-link`	O-Link, OpenJTAG from www.100ask.net
`openmoko`	Openmoko debug board (v3)
`pavr`	Jason Kyle’s pAVR Serial Programmer
`pickit2`	MicroChip’s PICkit2 Programmer
`picoweb`	Picoweb Programming Cable, http://www.picoweb.net/
`pony-stk200`	Pony Prog STK200
`ponyser`	design ponyprog serial, reset=!txd sck=rts mosi=dtr miso=cts
`siprog`	Lancos SI-Prog <http://www.lancos.com/siprogsch.html>
`sp12`	Steve Bolt’s Programmer
`stk200`	STK200
`stk500`	Atmel STK500
`stk500hvsp`	Atmel STK500 V2 in high-voltage serial programming mode
`stk500pp`	Atmel STK500 V2 in parallel programming mode
`stk500v1`	Atmel STK500 Version 1.x firmware
`stk500v2`	Atmel STK500 Version 2.x firmware
`stk600`	Atmel STK600
`stk600hvsp`	Atmel STK600 in high-voltage serial programming mode
`stk600pp`	Atmel STK600 in parallel programming mode
`usbasp`	USBasp, http://www.fischl.de/usbasp/
`usbasp-clone`	Any usbasp clone with correct VID/PID

`usbtiny`	USBtiny simple USB programmer, http://www.ladyada.net/make/usbtinyisp/
`wiring`	Wiring
`xil`	Xilinx JTAG cable

`-C` *`config-file`*

Use the specified config file for configuration data. This file contains all programmer and part definitions that AVRDUDE knows about. If you have a programmer or part that AVRDUDE does not know about, you can add it to the config file (be sure and submit a patch back to the author so that it can be incorporated for the next version). If not specified, AVRDUDE reads the configuration file from /usr/local/etc/avrdude.conf (FreeBSD and Linux). See Appendix A for the method of searching for the configuration file for Windows.

If *config-file* is written as *+filename* then this file is read after the system wide and user configuration files. This can be used to add entries to the configuration without patching your system wide configuration file. It can be used several times, the files are read in same order as given on the command line.

`-D`

Disable auto erase for flash. When the -U option with flash memory is specified, avrdude will perform a chip erase before starting any of the programming operations, since it generally is a mistake to program the flash without performing an erase first. This option disables that. Auto erase is not used for ATxmega devices as these devices can use page erase before writing each page so no explicit chip erase is required. Note however that any page not affected by the current operation will retain its previous contents.

`-e`

Causes a chip erase to be executed. This will reset the contents of the flash ROM and EEPROM to the value ‘0xff’, and clear all lock bits. Except for ATxmega devices which can use page erase, it is basically a prerequisite command before the flash ROM can be reprogrammed again. The only exception would be if the new contents would exclusively cause bits to be programmed from the value ‘1’ to ‘0’. Note that in order to reprogram EERPOM cells, no explicit prior chip erase is required since the MCU provides an auto-erase cycle in that case before programming the cell.

`-E` *`exitspec`*`[,…]`

By default, AVRDUDE leaves the parallel port in the same state at exit as it has been found at startup. This option modifies the state of the ‘/RESET’ and ‘Vcc’ lines the parallel port is left at, according to the exitspec arguments provided, as follows:

`reset`

The ‘/RESET’ signal will be left activated at program exit, that is it will be held low, in order to keep the MCU in reset state afterwards. Note in particular that the programming algorithm for the AT90S1200 device mandates that the ‘/RESET’ signal is active before powering up the MCU, so in case an external power supply is used for this MCU type, a previous invocation of AVRDUDE with this option specified is one of the possible ways to guarantee this condition.

`noreset`

The '/RESET' line will be deactivated at program exit, thus allowing the MCU target program to run while the programming hardware remains connected.

`vcc`

This option will leave those parallel port pins active (i. e. high) that can be used to supply 'Vcc' power to the MCU.

`novcc`

This option will pull the 'Vcc' pins of the parallel port down at program exit.

`d_high`

This option will leave the 8 data pins on the parallel port active (i. e. high).

`d_low`

This option will leave the 8 data pins on the parallel port inactive (i. e. low).

Multiple *exitspec* arguments can be separated with commas.

`-F`

Normally, AVRDUDE tries to verify that the device signature read from the part is reasonable before continuing. Since it can happen from time to time that a device has a broken (erased or overwritten) device signature but is otherwise operating normally, this options is provided to override the check. Also, for programmers like the Atmel STK500 and STK600 which can adjust parameters local to the programming tool (independent of an actual connection to a target controller), this option can be used together with '`-t`' to continue in terminal mode.

`-i` *`delay`*

For bitbang-type programmers, delay for approximately *delay* microseconds between each bit state change. If the host system is very fast, or the target runs off a slow clock (like a 32 kHz crystal, or the 128 kHz internal RC oscillator), this can become necessary to satisfy the requirement that the ISP clock frequency must not be higher than 1/4 of the CPU clock frequency. This is implemented as a spin-loop delay to allow even for very short delays. On Unix-style operating systems, the spin loop is initially calibrated against a system timer, so the number of microseconds might be rather realistic, assuming a constant system load while AVRDUDE is running. On Win32 operating systems, a preconfigured number of cycles per microsecond is assumed that might be off a bit for very fast or very slow machines.

`-l` *`logfile`*

Use *logfile* rather than *stderr* for diagnostics output. Note that initial diagnostic messages (during option parsing) are still written to *stderr* anyway.

`-n`

No-write - disables actually writing data to the MCU (useful for debugging AVRDUDE).

`-O`

Perform a RC oscillator run-time calibration according to Atmel application note AVR053. This is only supported on the STK500v2, AVRISP mkII, and JTAG ICE mkII hardware. Note that the result will be stored in the EEPROM cell at address 0.

`-P` *`port`*

Use port to identify the device to which the programmer is attached. Normally, the default parallel port is used, but if the programmer type normally connects to the serial port, the default serial port will be used. See Appendix A, Platform Dependent Information, to find out the default port names for your platform. If you need to use a different parallel or serial port, use this option to specify the alternate port name.

On Win32 operating systems, the parallel ports are referred to as lpt1 through lpt3, referring to the addresses 0x378, 0x278, and 0x3BC, respectively. If the parallel port can be accessed through a different address, this address can be specified directly, using the common C language notation (i. e., hexadecimal values are prefixed by *0x*).

For the JTAG ICE mkII, if AVRDUDE has been built with libusb support, *port* may alternatively be specified as `usb`[:*serialno*]. In that case, the JTAG ICE mkII will be looked up on USB. If *serialno* is also specified, it will be matched against the serial number read from any JTAG ICE mkII found on USB. The match is done after stripping any existing colons from the given serial number, and right-to-left, so only the least significant bytes from the serial number need to be given. For a trick how to find out the serial numbers of all JTAG ICEs attached to USB, see Example Command Line Invocations.

As the AVRISP mkII device can only be talked to over USB, the very same method of specifying the port is required there.

For the USB programmer "AVR-Doper" running in HID mode, the port must be specified as *avrdoper*. Libusb support is required on Unix but not on Windows. For more information about AVR-Doper see http://www.obdev.at/avrusb/avrdoper.html.

For the USBtinyISP, which is a simplicistic device not implementing serial numbers, multiple devices can be distinguished by their location in the USB hierarchy. See section Troubleshooting, for examples.

For programmers that attach to a serial port using some kind of higher level protocol (as opposed to bit-bang style programmers), *port* can be specified as `net`:*host*:*port*. In this case, instead of trying to open a local device, a TCP network connection to (TCP) *port* on *host* is established. The remote endpoint is assumed to be a terminal or console server that connects the network stream to a local serial port where the actual programmer has been attached to. The port is assumed to be properly configured, for example using a transparent 8-bit data connection without parity at 115200 Baud for a STK500.

This feature is currently not implemented for Win32 systems.

`-q`

Disable (or quell) output of the progress bar while reading or writing to the device. Specify it a second time for even quieter operation.

`-u`

Disables the default behaviour of reading out the fuses three times before programming, then verifying at the end of programming that the fuses have not changed. If you want to change fuses you will need to specify this option, as avrdude will see the fuses have changed (even though you wanted to) and will change them back for your "safety". This option was designed to prevent cases of fuse bits magically changing (usually called *safemode*).

If one of the configuration files contains a line

```
default_safemode = no;
```

safemode is disabled by default. The '`-u`' option's effect is negated in that case, i. e. it *enables* safemode.

Safemode is always disabled for AVR32, Xmega and TPI devices.

`-s`

Disable safemode prompting. When safemode discovers that one or more fuse bits have unintentionally changed, it will prompt for confirmation regarding whether or not it should attempt to recover the fuse bit(s). Specifying this flag disables the prompt and assumes that the fuse bit(s) should be recovered without asking for confirmation first.

`-t`

Tells AVRDUDE to enter the interactive "terminal" mode instead of up- or downloading files. See below for a detailed description of the terminal mode.

`-U` *`memtype`*`:`*`op`*`:`*`filename`*`[:`*`format`*`]`

Perform a memory operation. Multiple '`-U`' options can be specified in order to operate on multiple memories on the same command-line invocation. The *memtype* field specifies the memory type to operate on. Use the '`-v`' option on the command line or the `part` command from terminal mode to display all the memory types supported by a particular device. Typically, a device's memory configuration at least contains the memory types `flash` and `eeprom`. All memory types currently known are:

`calibration`

One or more bytes of RC oscillator calibration data.

`eeprom`

The EEPROM of the device.

`efuse`

The extended fuse byte.

`flash`

> The flash ROM of the device.

`fuse`

> The fuse byte in devices that have only a single fuse byte.

`hfuse`

> The high fuse byte.

`lfuse`

> The low fuse byte.

`lock`

> The lock byte.

`signature`

> The three device signature bytes (device ID).

`fuse`*`N`*

> The fuse bytes of ATxmega devices, *N* is an integer number for each fuse supported by the device.

`application`

> The application flash area of ATxmega devices.

`apptable`

> The application table flash area of ATxmega devices.

`boot`

> The boot flash area of ATxmega devices.

`prodsig`

> The production signature (calibration) area of ATxmega devices.

`usersig`

> The user signature area of ATxmega devices.

The *op* field specifies what operation to perform:

`r`

> read the specified device memory and write to the specified file

`w`

read the specified file and write it to the specified device memory

`v`

read the specified device memory and the specified file and perform a verify operation

The *filename* field indicates the name of the file to read or write. The *format* field is optional and contains the format of the file to read or write. Possible values are:

`i`

Intel Hex

`s`

Motorola S-record

`r`

raw binary; little-endian byte order, in the case of the flash ROM data

`e`

ELF (Executable and Linkable Format), the final output file from the linker; currently only accepted as an input file

`m`

immediate mode; actual byte values specified on the command line, separated by commas or spaces in place of the *filename* field of the '`-U`' option. This is useful for programming fuse bytes without having to create a single-byte file or enter terminal mode. If the number specified begins with `0x`, it is treated as a hex value. If the number otherwise begins with a leading zero (`0`) it is treated as octal. Otherwise, the value is treated as decimal.

`a`

auto detect; valid for input only, and only if the input is not provided at stdin.

`d`

decimal; this and the following formats are only valid on output. They generate one line of output for the respective memory section, forming a comma-separated list of the values. This can be particularly useful for subsequent processing, like for fuse bit settings.

`h`

hexadecimal; each value will get the string *0x* prepended.

`o`

octal; each value will get a *0* prepended unless it is less than 8 in which case it gets no prefix.

`b`

binary; each value will get the string *0b* prepended.

The default is to use auto detection for input files, and raw binary format for output files.

Note that if *filename* contains a colon, the *format* field is no longer optional since the filename part following the colon would otherwise be misinterpreted as *format*.

As an abbreviation, the form `-U` *filename* is equivalent to specifying `-U` *flash:w:filename:a*. This will only work if *filename* does not have a colon in it.

`-v`

Enable verbose output. More `-v` options increase verbosity level.

`-V`

Disable automatic verify check when uploading data.

`-x` *`extended_param`*

Pass *extended_param* to the chosen programmer implementation as an extended parameter. The interpretation of the extended parameter depends on the programmer itself. See below for a list of programmers accepting extended parameters.

[<] [>]   [<<] [Up] [>>]

This document was generated by *Joerg Wunsch* on *September 17, 2013* using *texi2html 1.82*.