

AVRDUDE

A program for downloading/uploading AVR microcontroller flash, EEPROM and more
for AVRDUDE, Version 8.2, 01 September 2026

by Hans Eirik Bull, Brian S. Dean, Stefan Rüger and
Jörg Wunsch

Use <https://github.com/avrdudes/avrdude/issues> to report bugs and ask questions.
Copyright © Hans Eirik Bull, Brian S. Dean, Stefan Rüger and Jörg Wunsch

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

Table of Contents

1	Introduction	1
1.1	History and Credits	4
2	Command Line Options	6
2.1	Option Descriptions	6
2.2	Programmers Accepting Exitspec Parameters	16
2.3	Programmers Accepting Extended Parameters	17
2.4	Example Command Line Invocations	28
3	Terminal Mode Operation	36
3.1	Terminal Mode Commands	36
3.2	Terminal Mode Examples	46
4	Configuration Files	53
4.1	AVRDUDE Defaults	53
4.2	Programmer Definitions	54
4.3	Serial Adapter Definitions	55
4.4	Part Definitions	56
4.4.1	Parent Part	58
4.4.2	Instruction Format	59
4.5	Other Notes	59
5	Autogenerated files	61
5.1	Test Files <code>dry:</code>	61
5.2	Bootloader Files <code>urboot:</code>	63
6	Programmer-Specific Information	69
6.1	Atmel STK600	69
6.2	DFU Bootloader Using FLIP Version 1	71
6.3	SerialUPDI Programmer	71
6.4	Programmer LED Management	73
Appendix A	Platform Dependent Information	74
A.1	Unix	74
A.1.1	Unix Installation	74
A.1.2	Unix Configuration Files	74
A.1.2.1	FreeBSD Configuration Files	74
A.1.2.2	Linux Configuration Files	74
A.1.3	Unix Port Names	74
A.1.4	Unix USB Permissions	74

A.1.4.1	FreeBSD USB Permissions	75
A.1.4.2	Linux USB Permissions	75
A.1.5	Unix Documentation	76
A.2	Windows	76
A.2.1	Installation	76
A.2.2	Windows Configuration Files	76
A.2.2.1	Windows Configuration File Names	77
A.2.2.2	Windows Configuration File Location	77
A.2.3	Windows Port Names	77
A.2.3.1	Windows Serial Ports	77
A.2.3.2	Windows Parallel Ports	77
Appendix B	Troubleshooting	78
Appendix C	List of Programmers	83
Appendix D	List of Parts	88
Appendix E	List of Memories	97
E.1	Classic parts	97
E.2	ATxmegas	97
E.3	Modern AVR Parts	98
Concept Index		100

1 Introduction

AVRDUDE - AVR Downloader Uploader - is a program for downloading and uploading the on-chip memories of Atmel's AVR microcontrollers. It can program the Flash, EEPROM, and where supported by the programmer, lock bits, fuses that hold the microcontroller's configuration and other memories that the part might have.

AVRDUDE can be used via the command line to read or write chip memories (eeprom, flash, fuses, lock bits) and read memories such as signature or calibration bytes; the same can be achieved via an interactive terminal mode. Using AVRDUDE from the command line works well for programming the entire memory of the chip from the contents of a file, while interactive mode is useful for exploring memory contents, modifying individual bytes of eeprom, programming fuse/lock bits, etc.

Programming a microcontroller either requires a physical programmer that sits between the target chip and the PC running AVRDUDE, or a bootloader program on the target chip that is then directly connected to the PC to be served by AVRDUDE. Currently, AVRDUDE knows about 0 parts and 0 programmers, though not every programmer can deal with every part. One noteworthy programmer is **dryrun**, which allows one to explore the AVRDUDE command-line and terminal without needing to have, or connect, a real physical programmer. Similarly, **dryboot** allows exploring how to communicate with a bootloader without connecting an AVR part.

AVRDUDE supports the following basic programmer types: Atmel's STK500, Atmel's AVRISP and AVRISP mkII devices, Atmel's STK600, Atmel's JTAG ICE (the original one, mkII, and 3), appnote avr910, appnote avr109 (including the AVR Butterfly), serial bit-bang adapters, and, on Linux and BSD systems, the PPI (parallel port interface). PPI represents a class of simple programmers where the programming lines are directly connected to the PC parallel port. Several pin configurations exist for several variations of the PPI programmers, and AVRDUDE can be configured to work with them by either specifying the appropriate programmer on the command line or by creating a new entry in its configuration file. All that's usually required for a new entry is to tell AVRDUDE which pins to use for each programming function.

A number of equally simple bit-bang programming adapters that connect to a serial port are supported as well, among them the popular Ponyprog serial adapter, and the DASA and DASA3 adapters that used to be supported by uisp(1). Note that these adapters are meant to be attached to a physical serial port. Connecting to a serial port emulated on top of USB is likely to not work at all, or to work abysmally slow.

If you happen to have a Linux system with at least 4 hardware GPIOs available (like almost all embedded Linux boards) you can do without any additional hardware - just connect them to the SDO, SDI, RESET and SCK pins of the AVR's SPI interface and use the linuxgpio programmer type. Older boards might use the labels MOSI for SDO and MISO for SDI. It bitbangs the lines using the Linux sysfs GPIO interface. Of course, care should be taken about voltage level compatibility. Also, although not strictly required, it is strongly advisable to protect the GPIO pins from overcurrent situations in some way. The simplest would be to just put some resistors in series or better yet use a 3-state buffer driver like the 74HC244.

Under a Linux installation with direct access to the SPI bus and GPIO pins, such as would be found on a Raspberry Pi, the "linuxspi" programmer type can be used to

directly connect to and program a chip using the built in interfaces on the computer. The requirements to use this type are that an SPI interface is exposed along with one GPIO pin. The GPIO serves as the reset output since the Linux SPI drivers do not hold chip select down when a transfer is not occurring and thus it cannot be used as the reset pin. A readily available level translator should be used between the SPI bus/reset GPIO and the chip to avoid potentially damaging the computer's SPI controller in the event that the chip is running at 5 V and the SPI runs at 3.3 V. The GPIO chosen for reset can be configured in the avrdude configuration file using the `reset` entry under the `linuxspi` programmer, or directly in the port specification. An external pull-up resistor should be connected between the AVR's reset pin and Vcc. If Vcc is not the same as the SPI voltage, this should be done on the AVR side of the level translator to protect the hardware from damage.

On a Raspberry Pi, header J8 provides access to the SPI and GPIO lines.

Typically, pins 19, 21, and 23 are SPI SDO, SDI, and SCK, while pins 24 and 26 would serve as CE outputs. So, close to these pins is pin 22 as GPIO25 which can be used as /RESET, and pin 25 can be used as GND.

A typical programming cable would then look like:

J8 pin	ISP pin	Name
21	1	SDI
-	2	Vcc - leave open
23	3	SCK
19	4	SDO
22	5	/RESET
25	6	GND

The `-P port` option for this programmer typically has the format `/dev/spidev0.0:/dev/gpiochip0`. And, mind the 3.3 V voltage level of the Raspberry Pi!

The STK500, JTAG ICE, avr910, and avr109/butterfly use the serial port to communicate with the PC. The STK600, JTAG ICE mkII/3, AVRISP mkII, USBasp, avrftdi (and derivatives), and USBtinyISP programmers communicate through the USB, using `libusb` as a platform abstraction layer. The avrftdi adds support for the FT2232C/D, FT2232H, and FT4232H devices. These all use the MPSSE mode, which has a specific pin mapping. Bit 0 (the lsb of the byte in the config file) is SCK. Bit 1 is SDO, and Bit 2 is SDI. Bit 3 usually reset. The 2232C/D parts are only supported on interface A, but the H parts can be either A or B (specified by the `usbdev` config parameter). The STK500, STK600, JTAG ICE, and avr910 contain on-board logic to control the programming of the target device. The avr109 bootloader implements a protocol similar to avr910, but is actually implemented in the boot area of the target's flash, as opposed to being an external device. The fundamental difference between the two types lies in the protocol used to control the programmer. The avr910 protocol is very simplistic and can easily be used as the basis for a simple, home made programmer since the firmware is available online. On the other hand, the STK500 protocol is more robust and complicated and the firmware is not openly available. The JTAG ICE also uses a serial communication protocol which is similar to the STK500 firmware version 2 one. However, as the JTAG ICE is intended to allow on-chip debugging as well as memory programming, the protocol is more sophisticated. (The JTAG ICE mkII protocol can also be run on top of USB.) Only the memory programming functionality of the JTAG ICE is supported by AVRDUDE. For the JTAG ICE mkII/3, JTAG,

debugWIRE and ISP mode are supported, provided it has a firmware revision of at least 4.14 (decimal). See below for the limitations of debugWIRE. For ATxmega devices, the JTAG ICE mkII/3 is supported in PDI mode (Xmega parts), provided it has a revision 1 hardware and firmware version of at least 5.37 (decimal).

The Atmel-ICE (ARM/AVR) is supported (JTAG, PDI, debugWIRE, ISP, UPDI).

Atmel's XplainedPro boards, using EDBG protocol (CMSIS-DAP compliant), are supported by the "jtag3" programmer type.

Atmel's XplainedMini boards, using mEDBG protocol, are also supported by the "jtag3" programmer type.

The AVR Dragon is supported in all modes (ISP, JTAG, PDI, HVSP, PP, debugWIRE). When used in JTAG and debugWIRE mode, the AVR Dragon behaves similar to a JTAG ICE mkII, so all device-specific comments for that device will apply as well. When used in ISP and PDI mode, the AVR Dragon behaves similar to an AVRISP mkII (or JTAG ICE mkII in ISP mode), so all device-specific comments will apply there. In particular, the Dragon starts out with a rather fast ISP clock frequency, so the `-B bitclock` option might be required to achieve a stable ISP communication. For ATxmega devices, the AVR Dragon is supported in PDI mode, provided it has a firmware version of at least 6.11 (decimal).

Wiring boards (e.g. Arduino Mega 2560 Rev3) are supported, utilizing STK500 V2.x protocol, but a simple DTR/RTS toggle to set the boards into programming mode. The programmer type is "wiring". Note that the `-D` option will likely be required in this case, because the bootloader will rewrite the program memory, but no true chip erase can be performed.

Serial bootloaders that run a skeleton of the STK500 1.x protocol are supported via their own programmer type specification "arduino". This programmer works for the Arduino Uno Rev3 or any AVR that runs the Optiboot bootloader. The number of connection retry attempts can be specified as an extended parameter. See the section on *extended parameters* below for details.

Urprotocol is a leaner version of the STK500 1.x protocol that is designed to be backwards compatible with STK500 v1.x; it allows bootloaders to be much smaller, e.g., as implemented in the urboot project <https://github.com/stefanrueger/urboot>. The programmer type "urclock" caters for these urboot bootloaders. Owing to its backward compatibility, bootloaders that can be served by the arduino programmer can normally also be served by the urclock programmer. This may require specifying the size of (to AVR-DUDE) *unknown* bootloaders in bytes using the `-x bootsize=n` option, which is necessary for the urclock programmer to enable it to protect the bootloader from being overwritten. If an unknown bootloader has EEPROM read/write capability then the option `-x eepromrw` informs avrdude `-c urclock` of that capability.

The BusPirate is a versatile tool that can also be used as an AVR programmer. A single BusPirate can be connected to up to 3 independent AVRs. See the section on *extended parameters* below for details.

The USBasp ISP, USBtinyISP and CH341A adapters are also supported, provided AVR-DUDE has been compiled with libusb support. The former two feature simple firmware-only USB implementations, running on an ATmega8 (or ATmega88), or ATtiny2313, respectively. CH341A programmers connect directly to the AVR target. Their SPI bit clock is approximately 1.7 MHz and cannot be changed. As a consequence, the AVR target must

have a CPU frequency of 6.8 MHz or more: factory-set AVR parts, which typically run on an internal oscillator between 1 MHz and 1.6 MHz, cannot be programmed using `-c ch341a`.

The Atmel DFU bootloader is supported in both, FLIP protocol version 1 (AT90USB* and ATmega*U* devices), as well as version 2 (Xmega devices). See below for some hints about FLIP version 1 protocol behaviour.

The MPLAB(R) PICkit 4/5/Basic and MPLAB(R) SNAP are supported in JTAG, TPI, ISP, PDI and UPDI mode. The Curiosity Nano board is supported in UPDI mode. It is dubbed “PICkit on Board”, thus the name `pkobn_updi`.

SerialUPDI programmer implementation is based on Microchip’s *pymcuprog* (<https://github.com/microchip-pic-avr-tools/pymcuprog>) utility, but it also contains some performance improvements included in Spence Konde’s *DxCore* Arduino core (<https://github.com/SpenceKonde/DxCore>). In a nutshell, this programmer consists of simple USB-to-UART adapter, diode and couple of resistors. It uses serial connection to provide UPDI interface. See Section 6.3 [SerialUPDI Programmer], page 71, for more details and known issues.

The `jtag2updi` programmer is supported, and can program AVRs with a UPDI interface. `Jtag2updi` is just a firmware that can be loaded onto an AVR, which enables it to interface with avrdude using the `jtagice mkii` protocol via a serial link (<https://github.com/ElTangas/jtag2updi>).

The Micronucleus bootloader is supported for both protocol version V1 and V2. As the bootloader does not support reading from flash memory, use the `-V` option to prevent AVRDUDE from verifying the flash memory. See the section on *extended parameters* below for Micronucleus specific options.

The Teensy bootloader is supported for all AVR boards. As the bootloader does not support reading from flash memory, use the `-V` option to prevent AVRDUDE from verifying the flash memory. See the section on *extended parameters* below for Teensy specific options.

Appendix C [List of Programmers], page 83, holds a full listing of known programmers.

1.1 History and Credits

AVRDUDE was written by Brian S. Dean under the name of AVRPROG to run on the FreeBSD Operating System. Brian renamed the software to be called AVRDUDE when interest grew in a Windows port of the software so that the name did not conflict with AVRPROG.EXE which is the name of Atmel’s Windows programming software.

For many years, the AVRDUDE source resided in public repositories on the now read-only savannah.nongnu.org, where it continued to be enhanced and ported to other systems. In addition to FreeBSD, AVRDUDE now runs on Linux, macOS and Windows. The developers behind the porting effort primarily were Ted Roth, Eric Weddington, and Jörg Wunsch.

In 2022, the project moved to Github (<https://github.com/avrdudes/avrdude/>).

And in the spirit of many open source projects, this manual also draws on the work of others. The initial revision was composed of parts of the original Unix manual page written by Jörg Wunsch, the original web site documentation by Brian Dean, and from the

comments describing the fields in the AVRDUDE configuration file by Brian Dean. The texi formatting was modeled after that of the Simulavr documentation by Ted Roth.

2 Command Line Options

2.1 Option Descriptions

AVRDUDE is a command line tool, used as follows:

```
avrdude -p partname options ...
```

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

-p *partname*

--part *partname*

This option tells AVRDUDE what part (MCU) is connected to the programmer. The *partname* parameter is the part's id listed in the configuration file. To see a list of currently supported MCUs use ? as partname, which will, for each part, print its id; its official part name; alternative names, if any; and the list of available programming interfaces. Depending on the used shell, ? may need to be quoted as in "?" or \?. In connection with -v, this will also print a table of variant part names with the package code and some absolute maximum ratings. The part id, their official part name, the listed alternative names or any of the full variant part names can be used to specify a part with the -p option. 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. If -p ? is specified with a specific programmer, see -c below, then only those parts are output that the programmer expects to be able to handle, together with the programming interface(s) that can be used in that combination. In reality there can be deviations from this list, particularly if programming is directly via a bootloader. See Appendix D [List of Parts], page 88, for a full and detailed listing of supported parts.

-p *wildcard/flags*

--part *wildcard/flags*

Run developer options for MCUs that are matched by *wildcard*. Whilst their main use is for developers some *flags* can be of utility for users, e.g., `avrdude -p m328p/S` outputs AVRDUDE's understanding of ATmega328P MCU properties; for more information run `avrdude -p x/h`.

-b *baudrate*

--baud *baudrate*

Override the RS-232 connection baud rate specified in the respective programmer's **baudrate** entry of the configuration file or defined by the **default_baudrate** entry in your `~/.config/avrdude/avrdude.rc` or `~/.avrduderc` configuration file if no **baudrate** entry was provided for this programmer.

-B *bitclock*

--bitclock *bitclock*

Specify the bit clock period for the JTAG, PDI, TPI, UPDI, or ISP interface. The value is a floating-point number in microseconds. Alternatively,

the value might be suffixed with Hz, kHz or MHz in order to specify the bit clock frequency rather than a period. Some programmers default their bit clock value to a 1 microsecond bit clock period, suitable for target MCUs running at 4 MHz clock and above. Slower MCUs need a correspondingly higher bit clock period. Some programmers reset their bit clock value to the default value when the programming software signs off, whilst others store the last used bit clock value. It is recommended to always specify the bit clock if read/write speed is important. You can use the 'default_bitclock' keyword in your `~/.config/avrdude/avrdude.rc` or `~/.avrduderc` configuration file to assign a default value to keep from having to specify this option on every invocation.

Note that some official Microchip programmers store the bitclock setting and will continue to use it until a different value is provided. This applies to 2nd generation programmers (AVRISPmkII, AVR Dragon, JTAG ICE mkII, STK600) and 3rd generation programmers (JTAGICE3, Atmel ICE, Power Debugger). 4th generation programmers (PICKit 4, MPLAB(R) SNAP) will store the last user-specified bitclock until the programmer is disconnected from the computer.

`-c programmer-id`

`--programmer programmer-id`

Specify the programmer to be used. AVRDUDE knows about quite a few programmers. The *programmer-id* parameter is the programmer's id listed in the configuration file. Specify `-c ?` to list all programmers in the configuration file. Depending on the used shell, `?` may need to be quoted as in `"?"` or `\?`. If you have a programmer that is unknown to AVRDUDE but related to a known programmer there is some chance that it can be added to the configuration file without any code changes to AVRDUDE: copy a similar entry and change those features that differ to match that of the unknown programmer. If `-c ?` is specified with a specific part, see `-p` above, then only those programmers are output that expect to be able to handle this part, together with the programming interface(s) that can be used in that combination. In reality there can be deviations from this list, particularly if programming is directly via a boot-loader. See Appendix C [List of Programmers], page 83, for a full and detailed listing of known programmers.

`-c wildcard/flags`

`--programmer wildcard/flags`

Run developer options for programmers that are matched by *wildcard*. Whilst their main use is for developers some *flags* can be of utility for users, e.g., `avrdude -c usbtiny/S` shows AVRDUDE's understanding of usbtiny's properties; for more information run `avrdude -c x/h`.

`-C config-file`

`--config config-file`

Use the specified config file for configuration data. This file contains all programmer and part definitions that AVRDUDE knows about. If not specified, AVRDUDE looks for the configuration file in the following two locations:

1. *directory from which application loaded/../../etc/avrdude.conf*

2. *directory from which application loaded/avrdude.conf*

If not found there, the lookup procedure becomes platform dependent. On FreeBSD and Linux, AVRDUDE looks at `/usr/local/etc/avrdude.conf`. See Appendix A for the method of searching on 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.

-N

--noconfig

Do not load the personal configuration file that is usually located at `~/.config/avrdude/avrdude.rc`, `~/.avrduderc` or in the same directory as the avrdude executable.

-A

--keep-trailing-0xff

Disable the automatic removal of trailing-0xFF sequences in file input that is to be programmed to flash and in AVR reads from flash memory. Normally, trailing 0xFFs can be discarded, as flash programming requires the memory be erased to 0xFF beforehand. **-A** should be used when the programmer hardware, or bootloader software for that matter, does not carry out chip erase and instead handles the memory erase on a page level. The popular Arduino bootloader exhibits this behaviour; for this reason **-A** is engaged by default when specifying **-c arduino**.

-D

--noerase

Disable auto-erase for flash. When the **-U** option for writing to any 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 parts nor for the UPDI (AVR8X family) parts as these can use page erase before writing each page so no explicit chip erase is required. Note, however, that any flash page not affected by the current operation will retain its previous contents. Setting **-D** implies **-A**.

-e

--erase

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 and UPDI (AVR8X family) devices, all of 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. This option carries out the chip erase at the beginning, before any of the **-U**, **-T** or **-t** options are processed. If a chip erase is required in at a certain position within the sequence of **-U**, **-T** or

`-t` options it is recommended to use `-T` erase instead which is processed in the given command line order.

In absence of an explicit `-e` or `-D` option `avrdude` tries to augur from the command line whether or not the chip should be auto-erased at the beginning. If `avrdude` detects a `-U` command that writes to flash then auto-erase will be carried out before any other programming unless a `-T` erase command has been detected beforehand and unless flash is read before writing to it. For the purpose of this analysis any terminal command is considered to possibly read flash.

Note that for reprogramming EEPROM 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[,...]`

`--exitspecs exitspec[,...]`

Pass *exitspec* to the programmer. The interpretation of the *exitspec* parameter depends on the programmer itself. See below for a list of programmers accepting *exitspec* parameter options or issue `avrdude -E help ...` to see the options for the programmer.

Multiple *exitspec* options can be separated with commas.

`-F`

`--force` 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. Moreover, the option allows to continue despite failed initialization of connection between a programmer and a target.

`-i delay`

`--isp-clock-delay 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`

`--logfile logfile`

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

`-n`

`--test-memory`

No-write: disables writing data to the MCU whilst processing `-U` (useful for debugging AVRDUDE). The terminal mode continues to write to the device.

`-O`

`--osccal` 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`

`--port port`

Use *port* to identify the connection through which the programmer is attached. This can be a serial, spi, linuxgpio or, on a Linux or BSD system, parallel connection. The programmer configuration normally only ever specifies the connection type but an optional port entry can be added in the per-user configuration file `.avrduderc`, which is then used in absence of `-P`. If the programmer configuration does not specify the *port*, system-dependent default values `default_serial`, `default_spi`, `default_linuxgpio` or `default_parallel` from the configuration file are used. These, too, can be overwritten in the per-user configuration file.

USB-only programmers normally do not need a *port* name as they are automatically identified via their vendor and product IDs from `avrdude.conf` or `.avrduderc`. Only when there are multiple programmers of the same type plugged in is the programmer's port configuration or the `-P` option needed, see below. Some `-c` programmers, e.g., pickit2, ignore the `-P` option altogether; these cannot distinguish multiple plugged-in programmers.

Most USB programmers, however, support the port name syntax `usb[:vid:pid][:serialno]`, which allows the user to override the vendor and product IDs with hexadecimal numbers *vid* and *pid* and/or request a match of the desired device serial number with *serialno*. The match is done after stripping any existing colons from the given serial number on the command line, and right-to-left, so only the least significant bytes from the serial number need to be given. The JTAG ICE mkII, JTAGICE3, SNAP, PICKit5, CH341A, teensy and avrftdi programmers are examples for this `-P` port syntax. Some of these in turn, e.g. the CH341A and teensy, are not capable of matching serial numbers.

If avrdude has been configured with libserialport support, a serial port can be specified using a predefined serial adapter type in `avrdude.conf` or `.avrduderc`, e.g., `ch340` or `ft232r`. If more than one serial adapter of the same type is connected, they can be distinguished by appending a serial number, e.g.,

`ft232r:12345678`. The USB to serial chip has to have a serial number for this to work. Note that serial number matches for serial adapters (in contrast to matches for USB programmers above) is from left-to-right. For matches from right-to-left, serial adapters can use the ellipses. In the above example, `ft232r:1234` would also result in a match, and so would `ft232r:...5678`. If the USB to serial chip is not known to avrdude, it can be specified using the hexadecimal USB vendor ID, hexadecimal product ID and an optional serial number, following the serial number matching rules described above, e.g., `usb:0x2341:0x0043` or `usb:2341:0043:12345678`. To see a list of currently plugged-in serial ports use `-P ?s`. In order to see a list of all possible serial adapters known to avrdude use `-P ?sa`. Depending on the used shell, `?` may need to be quoted as in `"?"` or `\?`.

For the JTAG ICE mkII, if AVRDUDE has been built with libusb support, the port can 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. `avrdude -v -P usb:xyz -c jtag2 -p ... 2>&1 | grep ^Found` lists all JTAG ICEs attached to USB, see Section 2.4 [Example Command Line Invocations], page 28.

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 `-P avrdoper`. Libhidapi support is required on Unix and Mac OS but not on Windows. For more information about AVR-Doper see <https://www.obdev.at/products/vusb/avrdoper.html>.

For the USBtinyISP, which is a simplistic device not implementing serial numbers, multiple devices can be distinguished by their location in the USB hierarchy using `-P usb:busdir:devicefile`.

For USBasp, multiple devices can also be distinguished using `-P usb:busdir:devicefile` or using the serial number `-P usb:serialno`. The USBasp serial number is matched from the end, so only the unique least significant bytes are needed. For examples, see the respective entry in Appendix B [Troubleshooting], page 78.

For the XBee programmer the target MCU is to be programmed wirelessly over a ZigBee mesh using the XBeeBoot bootloader. The ZigBee 64-bit address for the target MCU's own XBee device must be supplied as a 16-character hexadecimal value as a port prefix, followed by the `@` character, and the serial device to connect to a second directly contactable XBee device associated with the same mesh (with a default baud rate of 9600). This may look similar to: `0013a20000000001dev/tty.serial`.

For diagnostic purposes, if the target MCU with an XBeeBoot bootloader is connected directly to the serial port, the 64-bit address field can be omitted. In this mode the default baud rate will be 19200.

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. Square brackets may be placed around *host* to improve readability for numeric IPv6 addresses (e.g. **net:[2001:db8::42]:1337**). 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.

Note: IPv6 hostnames and addresses are limited to Posix systems.

-r

--reconnect

Opens the serial port at 1200 baud and immediately closes it, waits 400 ms for each **-r** on the command line and then establishes communication with the programmer. This is commonly known as a "1200 bps touch", and is used to trigger programming mode for certain boards like Arduino Leonardo, Arduino Micro/Pro Micro and the Arduino Nano Every. Longer waits, and therefore multiple **-r** options, are sometimes needed for slower, less powerful hosts.

-q

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

-T cmd

--command cmd

Run terminal line *cmd* when it is its turn in relation to other **-t** interactive terminals, **-T** terminal commands and **-U** memory operations. Except for the simplest of terminal commands the argument *cmd* will most likely need to be set in quotes, see your OS shell manual for details. See below for a detailed description of all terminal commands.

-t

--terminal

Tells AVRDUDE to run an interactive terminal when it is its turn in relation to other **-t** interactive terminals, **-T** terminal commands and **-U** memory operations.

-U mem:op:file[:fmt]

--memory mem:op:fil[:fmt]

Perform a memory operation when it is its turn in relation to other **-t** interactive terminals, **-T** terminal commands and **-U** memory operations. The *mem* field specifies the memory type to operate on. From version 8.0 the memory field can also be a comma-separated list of memories, eg, **flash,eeeprom**; also, Intel Hex or Motorola S-Record files generated by AVRDUDE can store multiple memories. The special memory **ALL** expands to all memories that a part has while **all** expands to all memories with exception of sub-memories. **etc** is the

same as **all**; this can be used to change the order in which memories are written to or read from file, eg, **signature,etc** is a list of all memories such that the **signature** memory comes first. It is possible to remove a memory from the list so far by preceding a minus or backslash, eg, **all,-calibration**. Use the **-T part** option on the command line or the **part** command in the interactive terminal to display all the memories supported by a particular device.

Typically, a device's memory configuration at least contains the memory types **flash**, **eeprom**, **signature** and **lock**, which is sometimes known as **lockbits**. The signature memory contains the three device signature bytes, which should be, but not always are, unique for the part. The **lock** memory of one or four bytes typically details whether or not external reading/writing of the flash memory, or parts of it, is allowed. After restricting access via the lock memory, often the only way to unlock memory is via a chip erase. Parts will also typically have fuse bytes, which are read/write memories for configuration of the device and calibration memories that typically contain read-only factory calibration values.

The flash memory, being physically implemented as NOR-memory, is special in the sense that it is normally only possible to program bits to change from 1 to 0. Before reprogramming takes place normally flash memory has to be erased. Older parts would only offer a chip erase to do so, which also erases EEPROM unless a fuse configuration preserves its contents. If AVRDUDE detects a **-U** option that writes to a flash memory it might automatically trigger a chip erase for these older parts. See the description of auto-erase under the **-e** option above. ATxmegas or UPDI parts (AVR8X family) offer a page erase, and AVRDUDE takes advantage of that by erasing pages before programming them unless **-e** (chip erase) or **-D** (do not erase before writing) was requested. It should be noted that in absence of the **-e** chip erase option any ATxmega or UPDI flash pages not affected by the programming will retain their previous content.

See Appendix E [List of Memories], page 97, for a complete list of memories that AVR devices can have.

The *op* field specifies what operation to perform:

- r** read device memories and write to the specified file
- w** read data from the specified file and write to the device memories in the list; read-only memories in a memory list are skipped, as are fuses and lock bits when the programmer is a bootloader; writing to single read-only memories fails only if the contents differs between the file and memory
- v** read data from both the device and the specified file and perform a verify

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

- i** Intel Hex

I	Intel Hex with comments on reading from, and tolerance of checksum errors, writing to the AVR
s	Motorola S-Record
r	raw binary; little-endian byte order, in the case of the flash 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 are specified on the command line, separated by commas or spaces in place of the <i>file</i> field of the -U option. This is useful for programming fuse bytes without having to create a single-byte file or enter terminal mode.
a	auto detect; valid for input only, and only if the input is not provided at stdin.
d	decimal; this and the following formats 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 <i>0x</i> prepended.
o	octal; each value will get a <i>0</i> prepended unless it is less than 8 in which case it gets no prefix.
b	binary; each value will get the string <i>0b</i> prepended.

When used as input, the **m**, **d**, **h**, **o** and **b** formats will use the same code for reading lists of numbers separated by white space and/or commas. The read routine handles decimal, hexadecimal, octal or binary numbers on a number-by-number basis, and the list of numbers can therefore be of mixed type. In fact the syntax, is the same as for data used by the terminal write command, i.e., the file's input data can also be 2-byte short integers, 4-byte long integers or 8-byte long long integers, 4-byte floating point numbers, 8-byte double precision numbers, C-type strings with a terminating nul or C-like characters such as `'\t'`. Numbers are written as little endian to memory. When using **0x** hexadecimal or **0b** binary input leading zeros are used to determine the size of the integer, e.g., **0x002a** will occupy two bytes and write a **0x2a** to memory followed by **0x00**, while **0x01234** will occupy 4 bytes. See the description of the terminal write command for more details.

In absence of an explicit file format, the default is to use auto detection for input files, raw binary format for output files from a single memory read and Intel Hex with comments when an output file is generated from a list of memories. Note that while AVRDUDE will generate a single output file from a memory list for all formats with the exception of elf (**:e**) it only recognises Intel hex (**:I** or **:i**), Motorola S-Record (**:s**) or elf files (**:e**, generated by the compiler) as valid multi-memory files when reading a file for verifying or writing memories. Note also that if a file name contains a colon as penultimate character the *fmt* field is no longer optional since the last character would otherwise be misinterpreted as *fmt*.

When reading any kind of flash memory area (including the various sub-areas in Xmega devices), the resulting output file will be truncated to not contain trailing 0xFF bytes which indicate unprogrammed (erased) memory. Thus, if the entire memory is unprogrammed, this will result in an output file that has no contents at all. This behaviour can be overridden with the `-A` option.

As an abbreviation, the form `-U file` is equivalent to specifying `-U flash:w:file:a` or `-U application:w:file:a` for ATxmegas. This will only work if *file* does not have a pair of colons in it that sandwich a single character as otherwise the first part might be interpreted as memory, and the single character as memory operation.

A file name used for writing to flash that starts with `urboot:` autogenerates a read-only urboot bootloader file. Try for example `-c dryrun -U urboot:help` for a list of features that determine the contents of the bootloader. See Section 5.2 [Bootloader Files], page 63, for detailed documentation. Writing `urboot:...` files to flash using `-U` has the desired side-effect of also writing all necessary fuse configurations for the bootloader to work.

A file name used for writing to the device that starts with `dry:` autogenerates a read-only multi-memory input file for testing purposes. See Section 5.1 [Test Files], page 61, for detailed documentation.

`-v`

`--verbose`

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

`-V`

`--noverify-memory`

Disable automatic verify check when writing data to the AVR with `-U`.

`--version`

Print avrdude version and exit

`-x parameter`

`--extended parameter`

Pass *parameter* 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 or issue `avrdude -x help ...` to see the extended options of the chosen programmer. This option can be used several times on the command line.

`-h`

`--help` Show a short help text and exit

2.2 Programmers Accepting Exitspec Parameters

Currently, the `flip2`, `linuxspi`, `linuxgpio`, `raspberrypi_gpio`, `pickit4_mplab`, `pickit5` and old-school parallel port programmers such as `stk200` and `dapa` support `-E` exitspec parameter options. These let the user decide in which state the programmer pins after ended programming session. AVRDUDE only allows one `-E` option. However, multiple exitspec parameters can be specified as one comma-separated list.

`flip2`

`linuxspi`

`linuxgpio`

`raspberrypi_gpio`

Parallel port programmers

<code>help</code>	Show help menu and exit.
<code>reset</code>	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. <code>flip2</code> will not exit bootloader mode at program exit if <code>reset</code> is used.
<code>noreset</code>	The '/RESET' line will be deactivated at program exit, thus allowing the MCU target program to run while the programming hardware remains connected. <code>flip2</code> will exit bootloader mode at program exit and start the application if <code>noreset</code> is used, and this is the default behaviour for this bootloader.

Parallel port programmers

<code>vcc</code>	This option will leave those parallel port pins active (i. e. high) that can be used to supply 'Vcc' power to the MCU.
<code>novcc</code>	This option will pull the 'Vcc' pins of the parallel port down at program exit.
<code>d_high</code>	This option will leave the 8 data pins on the parallel port active (i.e. high).
<code>d_low</code>	This option will leave the 8 data pins on the parallel port inactive (i.e. low).

Pickit 4 (MPLAB)

Pickit 5

<code>vcc</code>	This option will leave the power supply from the programmer enabled after avrdude finished the operation. Disabled by default.
------------------	--

2.3 Programmers Accepting Extended Parameters

Extended parameters are programmer-specific options; they all start with `-x`. Generally, each programmer will allow `-x help`, which will show a help menu of known extended parameters for this programmer, if any, and exit. The extended parameters below are all shown without the necessary `-x` option lead-in. AVRDUDE allows any number of `-x` extended parameters to be specified on the command line.

`dryrun`
`dryboot`

Dryrun emulates external programming without the need to connect a programmer or a part while dryboot emulates bootloader programming without the need to connect the target part.

`lock` and `fuse` memories are initialised with with factory values as far as known, `0xff` otherwise. The `signature` memory is set from the configuration file; the `calibration` memory is filled with U (for uncalibrated); `osc16err` with e and `osc20err` with E (for error); `osccal16` with o and `osccal20` with 0; `sib` with S; `tempsense` with T; `sernum` with the downward letter sequence UTSRQP...; and the volatile `io` memory with reset values if known, `0x00` otherwise.

If either the `init` or `random` parameters are set, then the `flash` memory is randomly configured in terms of bootloader sections, code and application data sections, and the fuses updated accordingly. In either case, `flash` (including ATxmega submemories of `application`, `apptable` and `boot`), `eeprom`, and all other existing memories such as `prodsig/sigrow`, `userrow/usersig` and `bootrow` are updated with random data. `flash` is always initialised with benign code, that is, its opcodes will not access I/O memories, SRAM or flash.

If none of `init` or `random` parameters are set, these memories are initialised with `0xff`. Note that `init` and `random` are not meant to be both set at the same time.

The dryrun and dryboot programmers accept the following parameters:

`init`

The patterns that are used for initialising memories as detailed above are human-readable. These patterns can best be seen with a fixed-width font and the `:I` format by inspecting the generated hex file or by using, eg, `-U flash:r:-:I` to dump the patterns on screen. `eeprom`, `userrow/usersig` and `bootrow` memories are filled with pangrams such as The quick brown fox jumps over the lazy dog. Choose a fixed seed for reproducible results.

`init=n` Shortcut for `-x init -x seed=n` (see below)

`random` Initialise `flash` with random opcodes and, if applicable, random application table data. The `sernum` memory, if it exists, will be initialised with a random upper-letter sequence. Other memories are initialised with a random sequence of at-signs and spaces. Choose a fixed seed for reproducible results.

`random=n` Shortcut for `-x random -x seed=n`

- holes** Put holes into larger memories, ie, longer sequences of 0xff, and add small islands of code or data. Some of these holes can pose problems for programmers that do not anticipate them. As such these can be used for hardened testing, which is the main purpose of the dryrun programmers.
- seed=*n*** Seed random number generator with *n*; the default is `time(NULL)`. Setting this option with a fixed positive *n* will make the random choices reproducible, ie, they will stay the same between different avrdude runs.

JTAG ICE mkII/3

Atmel-ICE

PICKit 4

MPLAB(R) SNAP

Power Debugger

AVR Dragon

When using the JTAG ICE mkII, JTAGICE3, Atmel-ICE, PICKit 4, MPLAB(R) SNAP, Power Debugger or AVR Dragon in JTAG mode, the following extended parameter is accepted:

jtagchain=UB,UA,BB,BA

Setup the JTAG scan chain for *UB* units before, *UA* units after, *BB* bits before, and *BA* bits after the target AVR, respectively. Each AVR unit within the chain shifts by 4 bits. Other JTAG units might require a different bit shift count.

hvupdi *Power Debugger and Pickit 4 only*

High-voltage UPDI programming is used to enable a UPDI pin that has previously been set to RESET or GPIO mode. Use `-x hvupdi` to enable high-voltage UPDI initialization for supported targets.

vtarg Read the target voltage.

vtarg=VALUE

Power Debugger only

The voltage generator can be enabled by setting a target voltage.

PICKit 4

MPLAB(R) SNAP

The PICKit 4 and MPLAB(R) SNAP programmers accept the following extended parameters:

vtarg Read the target voltage.

mode=avr Switch programmer to AVR mode, then exit unless it was already in AVR mode

mode=<mplab|pic>

Switch programmer to MPLAB aka PIC mode, then exit

Use `-x mode=avr` for switching to AVR mode, or `-x mode=mplab` for switching (back) to MPLAB mode.

PICkit 5**PICkit 4 (PIC Mode)**

The PICkit 5 and PICkit 4 (MPLAB Mode) programmer can accept following extended parameters

vtarg Read the target voltage.

vtarg=VALUE

Specify a voltage between 1.8 and 5.5 V that the programmer should supply to the target. If there is already a valid voltage applied to the VTG Pin, this setting will be ignored. When AVRDUDE detects an external voltage outside of this range, it will terminate the operation. You can disable this check by setting the voltage to 0 V. If an XMEGA part was selected, a requested voltage above 3.49 V will lead to an abort of operation. Usually, the programmer will stop providing power when the session ends. To continue to power the target you can use the **-E vcc** option.

hvupdi High-voltage UPDI programming is used to enable a UPDI pin that has previously been set to RESET or GPIO mode. Use **-x hvupdi** to enable high-voltage UPDI initialization for supported targets. Depending on the target, the HV pulse will be applied either on the RST pin, or the UPDI pin.

Xplained Mini

The Xplained Mini/Nano programmer (ISP or UPDI, not TPI) type accepts the following extended parameters:

suffer=VALUE, suffer

The SUFFER register allows the user to modify the behavior of the on-board mEDBG. The current state can be read by **-x suffer** alone.

Bit 7 ARDUINO:

Adds control of extra LEDs when set to 0

Bit 6..3: Reserved (must be set to 1)

Bit 2 EOF:

Aggressive power-down, sleep after 5 seconds if no USB enumeration when set to 0

Bit 1 LOWP:

forc running the mEDBG at 1 MHz when bit set to 0

Bit 0 FUSE:

Fuses are safe-masked when bit sent to 1. Fuses are unprotected when set to 0

vtarg_switch=VALUE, vtarg_switch

The on-board target voltage switch can be turned on or off by writing a 1 or a 0. The current state can be read by **-x vtarg_switch** alone. Note that the target power switch will always be

on after a power cycle. Also note that the smaller Xplained Nano boards does not have a target power switch.

Curiosity Nano

The Curiosity Nano board accepts the following extended parameter:

vtarg=VALUE, vtarg

The generated on-board target voltage can be changed by specifying a new voltage. The current set-voltage can be read by **-x vtarg** alone.

STK500

STK600

The STK500 and STK600 boards accept the following extended parameters:

vtarg=VALUE, vtarg

The generated on-board target voltage can be changed by specifying a new voltage. The current set-voltage can be read by **-x vtarg** alone.

fosc=VALUE[MHz|M|kHz|k|Hz|H] , fosc

Set the programmable oscillator frequency in MHz, kHz or Hz. The current frequency can be read by **-x fosc** alone.

varef=VALUE, varef

The generated on-board analog reference voltage can be changed by specifying a new reference voltage. The current reference voltage can be read by **-x varef** alone.

varef[0,1]=VALUE, varef[0,1]

STK600 only

The generated on-board analog reference voltage for channel 0 or channel 1 can be changed by specifying a new reference voltage. The current reference voltage can be read by **-x varef0** or **-x varef1** alone.

attempts[=1..99]

STK500V1 only

Specify how many connection retry attempts to perform before exiting. Defaults to 10 if not specified.

xtal=VALUE[MHz|M|kHz|k|Hz|H]

Defines the XTAL frequency of the programmer if it differs from 7.3728 MHz of the original STK500. Used by avrdude for the correct calculation of fosc and sck.

AVR109

The AVR109 programmer type accepts the following extended parameter:

autoreset

Toggle RTS/DTR lines on port open to issue a hardware reset.

AVR910

The Atmel low-cost AVR910 programmer type accepts the following extended parameter:

devcode=VALUE

Override the device code selection by using *VALUE* as the device code. The programmer is not queried for the list of supported device codes, and the specified *VALUE* is not verified but used directly within the **T** command sent to the programmer. *VALUE* can be specified using the conventional number notation of the C programming language.

no_blockmode

Disables the default checking for block transfer capability. Use **no_blockmode** only if your AVR910 programmer creates errors during initial sequence.

Arduino

The Arduino programmer type accepts the following extended parameter:

attempts[=1..99]

Specify how many connection retry attempts to perform before exiting. Defaults to 10 if not specified.

noautoreset

Do not toggle RTS/DTR lines on port open to prevent a hardware reset.

Urclock

The urclock programmer type accepts the following extended parameters:

showall Show all info for the connected part, then exit. The **-x show...** options below can be used to assemble a bespoke response consisting of a subset (or only one item) of all available relevant information about the connected part and bootloader.

showid Show a unique Urclock ID stored in either flash or EEPROM of the MCU, then exit.

id=E|F.addr.len

Historically, the Urclock ID was a six-byte unique little-endian number stored in Urclock boards at EEPROM address 257. The location of this number can be set by the **-x id=E|F.addr.len** extended parameter. **E** stands for EEPROM and **F** stands for flash. A negative address *addr* counts from the end of EEPROM and flash, respectively. The length *len* of the Urclock ID can be between 1 and 8 bytes.

showdate Show the last-modified date of the input file for the flash application, then exit. If the input file was stdin, the date will be that of the programming. Date and filename are part of the metadata that the urclock programmer stores by default in high flash just under the bootloader; see also **-x nometadata**.

showfilename	Show the input filename (or title) of the last flash writing session, then exit.
title=string	When set, <i>string</i> will be used in lieu of the input filename. The maximum string length for the title/filename field is 254 bytes including terminating nul.
showapp	Show the size of the programmed application, then exit.
showstore	Show the size of the unused flash between the application and metadata, then exit.
showmeta	Show the size of the metadata just below the bootloader, then exit.
showboot	Show the size of the bootloader, then exit.
showversion	Show bootloader version and capabilities, then exit.
showvector	Show the vector number and name of the interrupt table vector used by the bootloader for starting the application, then exit. For hardware-supported bootloaders this will be vector 0 (Reset), and for vector bootloaders this will be any other vector number of the interrupt vector table or the slot just behind the vector table with the name VBL_ADDITIONAL_VECTOR.
showpart	Show the part for which the bootloader was compiled, then exit.
bootsize=size	Manual override for bootloader size. Urboot bootloaders put the number of used bootloader pages into a table at the top of the bootloader section, i.e., typically top of flash, so the urclock programmer can look up the bootloader size itself. In backward-compatibility mode, when programming via other bootloaders, this option can be used to tell the programmer the size, and therefore the location, of the bootloader.
vectornum=n	Manual override for vector number. Urboot bootloaders put the vector number used by a vector bootloader into a table at the top of flash, so this option is normally not needed for urboot bootloaders. However, it is useful in backward-compatibility mode (or when the urboot bootloader does not offer flash read). Specifying a vector number in these circumstances implies a vector bootloader whilst the default assumption would be a hardware-supported bootloader.
eeepromrw	Manual override for asserting EEPROM read/write capability. Not normally needed for urboot bootloaders, but useful for in backward-

compatibility mode if the bootloader offers EEPROM read/write.

emulate_ce

If an urboot bootloader does not offer a chip erase command it will tell the urclock programmer so during handshake. In this case the urclock programmer emulates a chip erase, if warranted by user command line options, by filling the remainder of unused flash below the bootloader with 0xff. If this option is specified, the urclock programmer will assume that the bootloader cannot erase the chip itself. The option is useful for backwards-compatible bootloaders that do not implement chip erase.

restore

Write unchanged flash input files to the AVR and trim below the bootloader if needed. This is most useful when one has a backup of the full flash and wants to play that back onto the device. No metadata are written in this case and no vector patching happens either if it is a vector bootloader. However, for vector bootloaders, even under the option **-x restore** an input file will not be written to the AVR for which the reset vector does not point to the vector bootloader. This is to avoid loading an input file onto the device that would render the vector bootloader becoming unreachable after reset.

initstore

On writing to flash fill the store space between the flash application and the metadata section with 0xff.

nofilename

On writing to flash do not store the application input filename (nor a title).

nodate

On writing to flash do not store the application input filename (nor a title) and no date either.

nostore

On writing to flash do not store metadata except the metadata code byte 0xff saying there are no metadata. In particular, no data store frame is programmed.

nometadata

Do not support any metadata. The full flash besides the bootloader is available for the application. If the application is smaller than the available space then a metadata code byte 0xff is stored nevertheless to indicate there are no further metadata available. In absence of **-x nometadata**, the default for the urclock programmer is to write as much metadata (filename, data and store information) as the size of the application and the other extended options allow. The subtle difference between **-x nometadata** and **-x nostore** is that the latter always explicitly stores in flash that no further metadata are available, so that a such prepared flash can always be queried with **avrdude -x showall**. In contrast to this, it

cannot be guaranteed that a `-x showall` query on flash prepared with `-x nometadata` yields useful results.

`noautoreset`

Do not toggle RTS/DTR lines on port open to prevent a hardware reset.

`delay=n` Add a n ms delay after reset. This can be useful if a board takes a particularly long time to exit from external reset. n can be negative, in which case the default 120 ms delay after issuing reset will be shortened accordingly.

`strict` Urlock has a faster, but slightly different strategy than `-c arduino` to synchronise with the bootloader; some `stk500v1` bootloaders cannot cope with this, and they need the `-x strict` option.

BusPirate

The BusPirate programmer type accepts the following extended parameters:

`reset=cs,aux,aux2`

The default setup assumes the BusPirate's CS output pin connected to the RESET pin on AVR side. It is however possible to have multiple AVRs connected to the same BP with SDI, SDO and SCK lines common for all of them. In such a case one AVR should have its RESET connected to BusPirate's *CS* pin, second AVR's RESET connected to BusPirate's *AUX* pin and if your BusPirate has an *AUX2* pin (only available on BusPirate version v1a with firmware 3.0 or newer) use that to activate RESET on the third AVR.

It may be a good idea to decouple the BusPirate and the AVR's SPI buses from each other using a 3-state bus buffer. For example 74HC125 or 74HC244 are some good candidates with the latches driven by the appropriate reset pin (*cs*, *aux* or *aux2*). Otherwise the SPI traffic in one active circuit may interfere with programming the AVR in the other design.

`spifreq=0..7`

- 0 30 kHz (default)
- 1 125 kHz
- 2 250 kHz
- 3 1 MHz
- 4 2 MHz
- 5 2.6 MHz
- 6 4 MHz
- 7 8 MHz

`rawfreq=0..3`

Sets the SPI speed and uses the Bus Pirate's binary "raw-wire" mode instead of the default binary SPI mode:

- 0 5 kHz
- 1 50 kHz

- 2 100 kHz (Firmware v4.2+ only)
- 3 400 kHz (v4.2+)

The only advantage of the “raw-wire” mode is that different SPI frequencies are available. Paged writing is not implemented in this mode.

- pullups** Enable the Bus Pirate’s built-in pull-up resistors. These resistors are useful when working with different voltage levels. VPU pin of the Bus Pirate must be connected to an external voltage. For example: connect VPU pin to the +5V pin or an external power supply.
- hiz** Enable the Bus Pirate’s HiZ mode on SPI, allowing it to work as an open-collector and interface with external pull-up circuits. If the external target circuit does not have pull-ups, the Bus Pirate will not be able to send data.
- ascii** Attempt to use ASCII mode even when the firmware supports BinMode (binary mode). BinMode is supported in firmware 2.7 and newer, older FW’s either don’t have BinMode or their BinMode is buggy. ASCII mode is slower and makes the above `reset=`, `spifreq=` and `rawfreq=` parameters unavailable. Be aware that ASCII mode is not guaranteed to work with newer firmware versions, and is retained only to maintain compatibility with older firmware versions.
- nopagedwrite** Firmware versions 5.10 and newer support a binary mode SPI command that enables whole pages to be written to AVR flash memory at once, resulting in a significant write speed increase. If use of this mode is not desirable for some reason, this option disables it.
- nopagedread** Newer firmware versions support in binary mode SPI command some AVR Extended Commands. Using the “Bulk Memory Read from Flash” results in a significant read speed increase. If use of this mode is not desirable for some reason, this option disables it.
- cpufreq=125..4000** This sets the *AUX* pin to output a frequency of *n* kHz. Connecting the *AUX* pin to the XTAL1 pin of your MCU, you can provide it a clock, for example when it needs an external clock because of wrong fuses settings. Make sure the CPU frequency is at least four times the SPI frequency.
- serial_recv_timeout=1...** This sets the serial receive timeout to the given value. The timeout happens every time avrdude waits for the BusPirate prompt. Especially in ascii mode this happens very often, so setting a smaller value can speed up programming a lot. The default value is 100 ms. Using 10 ms might work in most cases.

Micronucleus bootloader

The Micronucleus programmer type accepts the following extended parameter:

wait=timeout

If the device is not connected, wait for the device to be plugged in. The optional *timeout* specifies the connection time-out in seconds. If no time-out is specified, AVRDUDE will wait indefinitely until the device is plugged in.

Teensy bootloader

The Teensy programmer type accepts the following extended parameter:

wait=timeout

If the device is not connected, wait for the device to be plugged in. The optional *timeout* specifies the connection time-out in seconds. If no time-out is specified, AVRDUDE will wait indefinitely until the device is plugged in.

Wiring

The Wiring programmer type accepts the following extended parameters:

snooze=n After performing the port open phase, AVRDUDE will wait/snooze for *snooze* milliseconds before continuing to the protocol sync phase. No toggling of DTR/RTS is performed if *snooze* > 0.

delay=n Add a *n* milliseconds delay after reset. This can be useful if a board takes a particularly long time to exit from external reset. *n* can be negative, in which case the default 100 ms delay after issuing reset will be shortened accordingly.

PICkit2

Connection to the PICkit2 programmer:

AVR	PICkit2
RST	VPP/MCLR (1)
VDD	VDD Target (2) – optional if AVR self powered
GND	GND (3)
SDI	PGD (4)
SCLK	PDC (5)
SDO	AUX (6)

The PICkit2 programmer type accepts the following extended parameters:

clockrate=rate

Sets the SPI clocking rate in Hz (default is 100 kHz). Alternately the -B or -i options can be used to set the period.

timeout=usb-transaction-timeout

Sets the timeout for USB reads and writes in milliseconds (default is 1500 ms).

USBasp

The USBasp programmer type accepts the following extended parameter:

section_config

Programmer will erase configuration section with option '-e' (chip erase), rather than entire chip. Only applicable to TPI devices (ATtiny 4/5/9/10/20/40).

xbee

The xbee programmer type accepts the following extended parameter:

xbeeresetpin=1..7

Select the XBee pin **DI01..7** that is connected to the MCU's /RESET line. The programmer needs to know which DIO pin to use to reset into the bootloader. The default (3) is the **DI03** pin (XBee pin 17), but some commercial products use a different XBee pin.

The remaining two necessary XBee-to-MCU connections are not selectable - the XBee **DOUT** pin (pin 2) must be connected to the MCU's **RXD** line, and the XBee **DIN** pin (pin 3) must be connected to the MCU's **TXD** line.

jtag2updi

serialupdi

The jtag2updi and serialupdi programmer types accept the following extended parameters:

rtsdtr=low,high

Forces RTS/DTR lines to assume low or high state during the whole programming session. Some programmers might use this signal to indicate UPDI programming state, but this is strictly hardware specific.

When not provided, driver/OS default value will be used.

linuxspi

The linuxspi programmer type accepts the following extended parameter:

disable_no_cs

Ensures the programmer does not use the **SPI_NO_CS** bit for the SPI driver. This parameter is useful for kernels that do not support the CS line being managed outside the application.

serprog

The serprog programmer type accepts the following extended parameter:

cs

Sets the chip select (**CS**) to use on supported programmers. Programmers supporting the 0x16 serprog command can have more than the default **CS** (0). This option allows to choose these additional **CSes** (1, 2, ...) for programming the AVR.

2.4 Example Command Line Invocations

AVRDUDE error messages, warnings and progress reports are generally written to stderr which can, in bash, be turned off by `2>/dev/null` or by using increasingly more `-q` options to suppress them. Terminal output of commands or that of the `-U` command with an output file named `-` are written to stdout. In some examples empty lines are shown for clarity that are not printed by AVRDUDE or the shell.

Write the file `diag.hex` to the ATmega128 chip using the STK500 programmer connected to the default serial port:

```
$ avrdude -p m128 -c stk500 -e -U flash:w:diag.hex

Reading 19278 bytes for flash from input file diag.hex
Writing 19278 bytes to flash
Writing | ##### | 100% 7.60 s
Reading | ##### | 100% 6.83 s
19278 bytes of flash verified

Avrdude done. Thank you.
```

Same but in **quell-progress-reporting (silent) mode** `-qq`:

```
$ avrdude -qq -p m128 -c stk500 -e -U flash:w:diag.hex
```

Using `&&` to confirm that the silent AVRDUDE command went OK:

```
$ avrdude -qq -p m128 -c stk500 -e -U flash:w:diag.hex && echo OK
OK
```

Save flash memory in raw binary format to the file named `c:/diag flash.bin`:

```
$ avrdude -p m128 -c stk500 -U flash:r:"c:/diag flash.bin":r

Reading flash memory ...
Reading | ##### | 100% 6.90 s
Writing 19278 bytes to output file diag flash.bin

Avrdude done. Thank you.
```

Read the fuses and print their values in different formats (hexadecimal, binary and octal):

```
$ avrdude -cusbasp -patmega128 -qq -Ulfuse:r:-:h -Uhfuse:r:-:b -Uefuse:r:-:o

0xbf
0b11000110
0377
```

Using the default programmer, write the file `diag.hex` to flash, the file `eeeprom.hex` to EEPROM, and set the extended, high, and low fuse bytes to 0xff, 0x89, and 0x2e respectively:

```
$ avrdude -p m128 -U flash:w:diag.hex \
          -U eeprom:w:eeeprom.hex \
          -U efuse:w:0xff:m \
          -U hfuse:w:0x89:m \
          -U lfuse:w:0x2e:m

Processing -U flash:w:diag.hex:i
Reading 19278 bytes for flash from input file diag.hex
Writing 19278 bytes to flash
Writing | ##### | 100% 7.60 s
Reading | ##### | 100% 6.81 s
19278 bytes of flash verified

Processing -U eeprom:w:eeeprom.hex:i
Reading 3328 bytes for eeprom from input file eeeprom.hex
Writing 3328 bytes to eeprom
Writing | ##### | 100% 1.20 s
Reading | ##### | 100% 0.70 s
3328 bytes of eeprom verified

Processing -U efuse:w:0xff:m
Reading 1 byte for efuse from input file 0xff
Writing 1 byte (0xFF) to efuse, 1 byte written, 1 verified

Processing -U hfuse:w:0x89:m
Reading 1 byte for hfuse from input file 0x89
Writing 1 byte (0x89) to hfuse, 1 byte written, 1 verified

Processing -U lfuse:w:0x2e:m
Reading 1 byte for lfuse from input file 0x2e
Writing 1 byte (0x2E) to lfuse, 1 byte written, 1 verified

Avrdude done. Thank you.
```

Write data from stdin (standard input) to EEPROM; no error output means all went fine:

```
$ echo 'The quick brown fox' | avrdude -c usbasp -p attiny13 -qq -U eeprom:w::-:r
```

Execute multiple terminal mode commands separated by semicolons:

```
$ echo 'write eeprom 0 "Bonjour"; write ee 0x18 0x12345678; dump eeprom 0 0x20' | \
  avrdude -qqcdryrun -patmega328p -t

0000 42 6f 6e 6a 6f 75 72 00 ff ff ff ff ff ff ff |Bonjour.....|
0010 ff ff ff ff ff ff ff ff 78 56 34 12 ff ff ff |.....xV4.....|
```

The same using **-T**:

```
$ avrdude -qqcdryrun -patmega328p \
-T 'write eeprom 0 "Bonjour"; write ee 0x18 0x12345678; dump eeprom 0 0x20'

0000 42 6f 6e 6a 6f 75 72 00 ff ff ff ff ff ff ff ff |Bonjour.....|
0010 ff ff ff ff ff ff ff ff 78 56 34 12 ff ff ff ff |.....xV4.....|
```

Read EEPROM and write content to stdout (standard output):

```
$ avrdude -qq -cusbsp -pattiny13 -Ueeprom:r:-:i

:20000000E2809954686520717569636B2062726F776E20666F78E280990AFFFFFFFFFD3
:20002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
:00000001FF
```

Same but redirect stderr (standard error output) to /dev/null instead of using **-qq**:

```
$ avrdude -cusbsp -pattiny13 -Ueeprom:r:-:i 2>/dev/null

:20000000E2809954686520717569636B2062726F776E20666F78E280990AFFFFFFFFFD3
:20002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
:00000001FF
```

Using the Avrdude output to print strings present in flash memory:

```
$ avrdude -pattiny13 -qq -U flash:r:-:r | strings

Main menu
Distance: %d cm
Exit
```

List the serial numbers of all JTAG ICEs attached to USB; this is done by specifying an invalid serial number, and increasing the verbosity level:

```
$ avrdude -c jtag2 -p m128 -P usb:xyz -v

Using port          : usb:xyz
Using programmer     : jtag2fast
Programmer baud rate : 115200
Found JTAG ICE with serno: 00A000001C6B
Found JTAG ICE with serno: 00A000001C3A
Found JTAG ICE with serno: 00A000001C30
Error: did not find any (matching) USB device usb:xyz (03eb:2103)
Error: unable to open port usb:xyz for programmer jtag2fast

Avrdude done.  Thank you.
```

Connect to the JTAG ICE mkII with a **serial number ending in 1C37** via USB, enter **interactive terminal mode**, list all **commands for the connected part** and quit:

```
$ avrdude -c jtag2 -p m649 -P usb:1c:37 -t

avrdude> help
Valid commands:
  dump      : display a memory section as hex dump
  read      : alias for dump
  disasm    : disassemble a memory section
  write     : write data to memory; flash and EEPROM are cached
  save      : save memory segments to file
  backup    : backup memories to file
  restore   : restore memories from file
  verify    : compare memories with file
  flush     : synchronise flash and EEPROM cache with the device
  abort     : abort flash and EEPROM writes, ie, reset the r/w cache
  erase     : perform a chip or memory erase
  config    : change or show configuration properties of the part
  factory   : reset part to factory state
  regfile   : I/O register addresses and contents
  include   : include contents of named file as if it was typed
  sig       : display device signature bytes
  part      : display the current part information
  send      : send a raw command to the programmer
  verbose   : display or set -v verbosity level
  quell     : display or set -q quell level for progress bars
  help      : show help message
  quit      : synchronise flash/EEPROM cache with device and quit

  !<line> : run the shell <line> in a subshell, eg, !ls *.hex
  # ...   : ignore rest of line (eg, used as comments in scripts)

d, r, w, q and ? abbreviate dump, read, write, quit and help.
Other than that any unique abbreviation of a command works, too.
cmd -? gives more details about a command cmd.

Note that not all programmer derivatives support all commands. Flash and
EEPROM type memories are normally read and written using a cache via paged
read and write access; the cache is synchronised on quit or flush commands.
The part command displays valid memories for use with dump and write.

avrdude> quit

Avrdude done.  Thank you.
```

Factory fuse setting of a device:

```
$ avrdude -patmega328p/St | grep initval

.ptmm ATmega328P lfuse initval 0x62
.ptmm ATmega328P hfuse initval 0xd9
.ptmm ATmega328P efuse initval 0xff
.ptmm ATmega328P lock initval 0xff
```

List of all parts known to AVRDUDE:

```
$ avrdude -p */d | cut -f2 -d""

ATtiny11
ATtiny12
ATtiny13
ATtiny13A
ATtiny15
AT89S51
[...]
AVR64EA48
LGT8F88P
LGT8F168P
LGT8F328P
```

List of all modern AVR parts (with UPDI interface) known to AVRDUDE:

```
$ avrdude -p */Ud | cut -f2 -d""

ATtiny202
ATtiny204
ATtiny402
[...]
AVR64EA28
AVR64EA32
AVR64EA48
```

List of all currently plugged-in serial devices known to the libserialport library:

```
$ avrdude -P \?s
Possible candidate serial ports are:
-P /dev/ttyUSB0 or -P ft232r:A600K203
-P /dev/ttyUSB1 or -P ft232r:MCU8
-P /dev/ttyUSB3, -P ch340 or -P ch340-115k
Note that above ports might not be connected to a target board or an AVR programmer.
Also note there may be other direct serial ports not listed above.
```

List of all serial adapters known to AVRDUDE, i.e., defined in avrdude.conf:

```
$ avrdude -P \?sa

Valid serial adapters are:
ch340   = [usbvid 0x1a86, usbpid 0x7523]
ch341a  = [usbvid 0x1a86, usbpid 0x5512]
ch342   = [usbvid 0x1a86, usbpid 0x55d2]
ch343   = [usbvid 0x1a86, usbpid 0x55d3]
ch344   = [usbvid 0x1a86, usbpid 0x55d5]
ch347   = [usbvid 0x1a86, usbpid 0x55da 0x55db 0x55dd 0x55de]
ch9102  = [usbvid 0x1a86, usbpid 0x55d4]
ch9103  = [usbvid 0x1a86, usbpid 0x55d7]
cp210x  = [usbvid 0x10c4, usbpid 0xea60 0xea70 0xea71]
ft232hl = [usbvid 0x0403, usbpid 0x6010]
ft231x  = [usbvid 0x0403, usbpid 0x6015]
ft234x  = [usbvid 0x0403, usbpid 0x6015]
ft230x  = [usbvid 0x0403, usbpid 0x6015]
ft232h  = [usbvid 0x0403, usbpid 0x6014]
ft232r  = [usbvid 0x0403, usbpid 0x6001]
ft4232h = [usbvid 0x0403, usbpid 0x6011]
pl2303  = [usbvid 0x067b, usbpid 0x2303 0x2304 0x23a3 0x23b3 0x23c3 0x23d3 0x23e3]
```

Output a list of non-bootloader programmers that can be used for a part. Note that `2>&1` folds stderr into stdout in a bash shell:

```
$ avrdude -c "?" -p 32ea32 2>&1 | grep -v bootloader

Valid programmers for part AVR32EA32 are:
atmelice_updi      = Atmel-ICE (UPDI)
dryrun             = Emulates programming without a programmer (TPI, [...])
jtag2updi          = JTAGv2 to UPDI bridge (UPDI)
nanoevery          = JTAGv2 to UPDI bridge (UPDI)
jtag3updi          = Atmel AVR JTAGICE3 (UPDI)
pickit4_mplab_updi = MPLAB(R) PICKit 4 (UPDI)
pickit4_updi       = MPLAB(R) PICKit 4 (UPDI)
pickit5_updi       = MPLAB(R) PICKit 5 (UPDI)
pickit_basic_updi  = MPLAB(R) PICKit Basic (UPDI)
pickit_basic_mplab_updi = MPLAB(R) PICKit Basic (UPDI)
pkobn_updi         = Curiosity nano (nEDBG) (UPDI)
powerdebugger_updi = Atmel PowerDebugger (UPDI)
serialupdi         = SerialUPDI (UPDI)
snap_mplab_updi    = MPLAB(R) SNAP (UPDI)
snap_updi          = MPLAB(R) SNAP (UPDI)
xplainedmini_updi  = Atmel XplainedMini (UPDI)
xplainedpro_updi   = Atmel XplainedPro (UPDI)
```

Print programmer definition as understood by AVRDUDE:

```
$avrdude -c linuxspi/s

#-----
# linuxspi
#-----

# This programmer uses the built in linux SPI bus devices to program an
# attached AVR. The reset pin must be attached to a GPIO pin that
# is otherwise unused (see gpioinfo(1)); the SPI bus CE pins are not
# suitable since they would release /RESET too early.
#

programmer # linuxspi
    id                = "linuxspi";
    desc              = "Use Linux SPI device in /dev/spidev*";
    type              = "linuxspi";
    prog_modes        = PM_TPI | PM_ISP;
    extra_features    = HAS_BITCLOCK_ADJ;
    connection_type   = spi;
    reset             = 25;      # Pi GPIO number - this is J8:22
;

```

Print filename of last stored sketch with its date stamp (only with urclock programmer):

```
$avrdude -qq -curclock -P/dev/ttyUSB0 -pattiny13 -x showdate -x showfilename

2023-05-19 11.13 blink.hex

```

AVRDUDE in a bash script creating terminal scripts that reset a part to factory settings:

```
$ cat make-init-scripts

#!/bin/bash
mkdir /tmp/factory
for part in $(avrdude -p\*/d | grep = | cut -f2 -d"'"); do
    echo $part
    avrdude -p$part/St | grep initval | cut -f3,5 | grep -ve-1 \
    | sed "s/./write &/" >/tmp/factory/$part.ini
done

```

Run above script and use one of the created terminal scripts:

```
$ ./make-init-scripts

$ cat /tmp/factory/ATmega328P.ini
write lfuse 0x62
write hfuse 0xd9
write efuse 0xff
write lock 0xff

$ avrdude -qq -cusbasp -pATmega328P -t < /tmp/factory/ATmega328P.ini

```

Create a bash function `avrdude-elf` that takes an elf file as input, with support for optional Avrdude flags at the end, and **writes to all memories specified in the elf file**. In this example, the elf file did not contain any EEPROM data:

```
# Show all writable memories present for the ATtiny13
$ echo $(avrdude -pattiny13/ot | grep write | cut -f3 | uniq)

eeprom flash lfuse hfuse lock

# Function that writes to all memories present in the elf file
avrdude-elf() {
    avrdude -cusbasp -pattiny13 -U{eeprom,flash,{l,h}fuse,lock}:w:"$1":e "${@:2}"
}

# Run function where -B8 and -V is appended to the Avrdude command
$ avrdude-elf blink.elf -B8 -V

Set SCK frequency to 93750 Hz

Processing -U eeprom:w:blink.elf:e
Reading 64 bytes for eeprom from input file blink.elf
Writing 64 bytes to eeprom
Writing | ##### | 100% 0.08 s
64 bytes of eeprom written

Processing -U flash:w:blink.elf:e
Reading 1024 bytes for flash from input file blink.elf
Writing 1024 bytes to flash
Writing | ##### | 100% 0.12 s
1024 bytes of flash written

Processing -U lfuse:w:blink.elf:e
Reading 1 byte for lfuse from input file blink.elf
Writing 1 byte (0x6A) to lfuse, 1 byte written

Processing -U hfuse:w:blink.elf:e
Reading 1 byte for hfuse from input file blink.elf
Writing 1 byte (0xFF) to hfuse, 1 byte written

Processing -U lock:w:blink.elf:e
Reading 1 byte for lock from input file blink.elf
Writing 1 byte (0xFF) to lock, 1 byte written

Avrdude done. Thank you.
```

3 Terminal Mode Operation

AVRDUDE has an interactive mode called *terminal mode* that is enabled by the `-t` option. In this mode, AVRDUDE only initializes communication with the MCU, and then awaits user commands on standard input. These commands allow, for example, to display and modify the various device memories, perform a chip erase, display the device signature bytes and part parameters, and to send raw programming commands. Terminal mode provides a command history using `readline(3)`, so previously entered command lines can be recalled and edited.

3.1 Terminal Mode Commands

The *addr* and *len* parameters of the `dump`, `read`, `disasm`, `write`, `save` and `erase` commands describe a memory interval. They can be negative with the same syntax as substring computations in perl or python. The table below defines the effective memory interval [*start*, *end*], given the memory size *sz*:

<i>addr</i>	<i>len</i>	Memory interval	Comment
0/positive	positive	[<i>addr</i> , <i>addr</i> + <i>len</i> -1]	Note: <i>len</i> = <i>end</i> - <i>start</i> + 1
0/positive	negative	[<i>addr</i> , <i>sz</i> + <i>len</i>]	End is <i>len</i> bytes below mem size <i>sz</i>
negative	positive	[<i>sz</i> + <i>addr</i> , <i>sz</i> + <i>addr</i> + <i>len</i> -1]	Start is <i>addr</i> bytes below mem size
negative	negative	[<i>sz</i> + <i>addr</i> , <i>sz</i> + <i>len</i>]	Combining above two cases
any	zero	empty set	No action

Note that *addr* must be in the range [-*sz*, *sz*-1]. After computing the memory interval [*start*, *end*] as per above table, the *effective* length *end*-*start* + 1 must not be negative; if the effective length is zero then no action is carried out. *end* may be beyond the available memory for the `dump`, `read` or `disasm` commands, in which case the operation wraps around the memory end, but the effective length is always limited to the memory size.

Here some examples for a memory with size *sz* of 0x800 (2048) bytes:

<i>addr</i>	<i>len</i>	Memory interval	Comment
0x700	12	[0x700, 0x70b]	Conventional use
1024	-257	[0x400, 0x6ff]	0x6ff = 2048-257
-512	512	[0x600, 0x7ff]	Last 512 bytes
-256	-1	[0x700, 0x7ff]	Last 256 bytes
0	49	[0, 48]	First 49 bytes
0	-49	[0, 1999]	All but the last 48 = <i>len</i> +1 bytes
0	-1	[0, 0x7ff]	All memory without knowing its size
2046	4	[0x7fe, 0x801]	Wrap around for read but error for write
2046	4096	[0x7fe, 0x17fe]	Read wrap around stops at 0x7fd
-1	-1	[0x7ff, 0x7ff]	One byte at 0x7ff is addressed
-1	-2	[0x7ff, 0x7fe]	No action: effective length is zero
-1	-3	[0x7ff, 0x7fd]	Error: effective length is negative

The following commands are implemented for all programmers:

dump *memory addr len*

Read from the specified memory interval (see above), and display in the usual hexadecimal and ASCII form.

dump *memory addr*

Read from memory *addr* as many bytes as the most recent dump memory *addr len* command with this very memory had specified (default 256 bytes), and display them.

dump *memory*

Continue dumping from the memory and location where the most recent dump command left off; if no previous dump command has addressed a memory an error message will be shown.

dump

Continue dumping from the memory and location where the most recent dump command left off; if no previous dump command has addressed a memory an error message will be shown.

dump *memory addr ...*

Start reading from *addr*, all the way to the last memory address (deprecated: use **dump *memory addr -1***).

dump *memory ...*

Read all bytes from the specified memory, and display them (deprecated: use **dump *memory 0 -1***).

read

Can be used as an alias for dump.

disasm [*options*] *dump-arguments*

Like dump, the disasm command displays a part of the specified memory, albeit by interpreting the memory contents as AVR opcodes and showing it as assembler source code. Unlike dump, the disasm command has options; these control how disasm displays its result (see below). Other than that, the syntax of specifying the memory and its to be processed interval is virtually the same as that of dump: the default disasm length is 32 bytes, though, and sometimes the length can be slightly shorter or longer than requested, so that the memory section for disasm aligns with opcodes. Disasm options, once set, stay in force until switched off, typically by changing the case of the option. This way, a simple disasm without further options can be used to step through memory keeping the appearance. Disasm knows the following options:

- g** Generate avr-gcc source: this sets **-sOFQ** and outputs a .text preamble and a main symbol unless the disassembly emits one itself; **-G** (default) switches off **-g** and stops outputting a preamble
- A** Do not show addresses; **-a** (the default) shows addresses
- O** Do not show opcode bytes; **-o** (the default) show opcode bytes
- C** Do not show comments; **-c** (the default) show comments
- f** Show affected flags in SREG, eg, **---SVNZC** for the **sbiw** opcode; **-F** (the default) do not show SREG flags

- q** Show the number of machine cycles that an opcode takes; **-Q** (the default) do not show the cycles
- n** Put the opcode full name into comment (eg, subtract immediate from word); **-N** (the default) do not show the full opcode names
- e** Put a technical explanation of the opcode into the comment, eg, `Rd+1:Rd <-- Rd+1:Rd - K` for the `sbiw` opcode; **-E** (the default) do not show technical explanations
- S** Use AVR instruction set style: this means that register pairs are shown as, eg, in `r31:30` instead of `r30`; **-s** (the default) use `avr-gcc` code style
- L** Do not preprocess labels; **-l** (the default) preprocess jump/call labels
- U** Do not show unused labels; **-u** (the default) show unused tagged labels
- d** Decode all opcodes including those that are undocumented; **-D** (the default) decode only opcodes that are valid for the part
- z** Zap the list of jumps and calls before disassembly
- t=file** Delete symbols from a previously read tagfile, if any, and read the tagfile *file* for assigning addresses to symbol names.

The tagfile is an ASCII file where each line describes a symbol for code label addresses (L), variable addresses in flash (P) and variables addresses in memory or I/O space (M). Hashmarks start a tagfile comment that extends to the end of the line and is ignored by `disasm`. Here is a defining example of how a tagfile looks like

```
0x7f54 L      putch      Outputs a char # L are code labels
0x7ffe P W 1   version16 A word integer # P are PGM data
0x7f80 P A 4   headings  Column headers # Auto-aligned strings
0x0100 M B 2048 sram      2 kB SRAM      # Memory address
```

Code labels L can be, eg, function names in program space or goto labels. They use up to four columns separated by white space: the address, the letter L, the symbolic name of the label and an optional comment column for the symbol, which is copied by `disasm` into the disassembly comment column, should this label be referenced or used by the code. Variable symbols have a P or M in the second column; they can be bytes or words (16 bits) as determined by the letter B or W in the third column and either single variables or arrays as specified by the multiplicity count in the forth column. P symbols, but not M symbols, can also encode chars (8 bits), longs (32 bits), quads (64 bits) or octas (128 bits) as signified by the letters C, L, Q or O, respectively, or be the base location of nul-terminated strings as encoded by A or S in the third column. Out of necessity, the space occupied by A/S strings varies. The difference between A and S symbols is that the array of A strings might have an additional nul

character to auto-align the space occupied by them to an even address. The fifth column is the symbolic name for the P or M address that can be used by `disasm` to output relevant addresses symbolically. P areas described in the tagfile also tell `disasm` that the corresponding area is not code and should not be disassembled as such; instead the directives are used for disassembly of that area. As with L labels, P and M variables may have an optional final comment column pertaining to the symbol that may be output in the disassembly column as and when the corresponding variables are used.

Tagfiles are useful for disassembly to make the output of `disasm` more readable. They can be built manually and incrementally as one's understanding of the code grows. Alternatively, the bash shell script `elf2tag` can automatically generate a tag file from the .elf file that produced the flash contents:

```
$ elf2tag application.elf >application.tag
```

`elf2tag` uses the `avr-objdump -d` disassembly to create L labels and `avr-nm` to generate M symbols.

`write memory addr data[,] {data[,]}`

Manually program the respective memory cells, starting at address *addr*, using the data items provided. The terminal implements reading from and writing to flash, EEPROM, bootrow and usersig type memories normally through a cache and paged access functions. All other memories are directly written to without use of a cache. Some older parts without paged access, depending on the programmer, might also have flash and EEPROM directly accessed without cache. Items *data* can have the following formats:

Type	Example	Size (bytes)
String	"Hello, world\n"	varying
File	C:/My\ projects/blink.hex	varying
File with format	blink.hex:i	varying
Character	'A'	1
Binary integer	0b101010	1, 2, 4 or 8
Octal integer	012345	1, 2, 4 or 8
Decimal integer	12345	1, 2, 4 or 8
Hexadecimal integer	0x12345	1, 2, 4 or 8
Decimal float	3.1415926	4
Hexadecimal float	0xA.8p2	4
Decimal double	3.141592653589793D	8
Hexadecimal double	0xA.8p2D	8

Data can be binary, octal, decimal or hexadecimal integers, floating point numbers or C-style strings and characters. If nothing matches, *data* will be interpreted as a name of a file containing data, which will be read and inserted at this point. In order to force the interpretation of a data item as file, e.g., when the file name would be understood as a number otherwise, the file name can be given a *:f* format specifier. In absence of a format suffix, the terminal will try to auto-detect the file format.

A file name that starts with **urboot:** autogenerates a read-only urboot bootloader file. Try for example the terminal command **write flash urboot:help** for a list of features that determine the contents of the bootloader. See Chapter 5 [Autogenerated Files], page 61, for detailed documentation. It is worth noting here that writing **urboot:...** files to flash in the terminal does *not* write the necessary fuses for the bootloader to work (in contrast to **-U** operations).

A file name that starts with **dry:** autogenerates a read-only multi-memory file, normally with non-trivial random contents. See Chapter 5 [Autogenerated Files], page 61, for detailed documentation.

For integers, an optional case-insensitive suffix specifies the data size:

LL	8 bytes (64 bits)
L	4 bytes (32 bits)
H or S	2 bytes (16 bits)
HH	1 byte (8 bits)

Suffix **D** indicates a 64-bit double, **F** a 32-bit float, whilst a floating point number without suffix defaults to 32-bit float. Hexadecimal floating point notation is supported. An ambiguous trailing suffix, e.g., **0x1.8D**, is read as no-suffix float where **D** is part of the mantissa; use a zero exponent **0x1.8p0D** to clarify.

An optional **U** suffix makes integers unsigned. Ordinary **0x** hexadecimal and **0b** binary integers are always treated as unsigned. **+0x**, **-0x**, **+0b** and **-0b** numbers with an explicit sign are treated as signed unless they have a **U** suffix. Unsigned integers cannot be larger than $2^{64}-1$. If n is an unsigned integer then $-n$ is also a valid unsigned integer as in C. Signed integers must fall into the $[-2^{63}, 2^{63}-1]$ range or a correspondingly smaller range when a suffix specifies a smaller type.

Ordinary **0x** hexadecimal and **0b** binary integers with n hex digits (counting leading zeros) use the smallest size of one, two, four and eight bytes that can accommodate any n -digit hexadecimal/binary integer. If an integer suffix specifies a size explicitly the corresponding number of least significant bytes are written, and a warning shown if the number does not fit into the desired representation. Otherwise, unsigned integers occupy the smallest of one, two, four or eight bytes needed. Signed numbers are allowed to fit into the smallest signed or smallest unsigned representation: For example, 255 is stored as one byte as **255U** would fit in one byte, though as a signed number it would not fit into a one-byte interval $[-128, 127]$. The number **-1** is stored in one byte whilst **-1U** needs eight bytes as it is the same as **0xFFFFffffffFFFFFFfU**.

One trailing comma at the end of data items is ignored to facilitate copy and paste of lists.

write memory data

The start address may be omitted if the size of the memory being written to is one byte. *data* can be anything including a file.

write memory file

The start address may be omitted when a file is written to the memory.

write memory *addr len data[,]* {*data[,]*} ...

The ellipsis ... form writes the data to the entire memory interval addressed by *addr len* and, if necessary, pads the remaining space by repeating the last data item. The fill write command does not write beyond the specified memory area even if more data than needed were given.

save memory {*addr len*} *file[:format]*

Save one or more memory segments to a file in a format specified by the *:format* letter. The default is *:r* for raw binary. Each memory segment is described by an address and length pair. In absence of any memory segments the entire memory is saved to the file. Only Motorola S-Record (*:s*) and Intel Hex (*:i* or *:I*) formats store address information with the saved data. Avrdude cannot currently save ELF file formats. All the other file formats lose the address information and concatenate the chosen memory segments into the output file. If the file name is - then avrdude writes to stdout.

backup memlist *file[:format]*

Backup one or more memories to the specified file using the selected format. The default format for a single-memory backup is *:r* (raw binary); for multi-memory backups it is *:I* (Intel Hex with comments). *Memlist* can be a comma separated list of memories just as in the *-U* command line argument. **backup** flushes the cache before reading memories.

restore memlist *file[:format]*

Restore one or more memories from the specified file. It is the user's responsibility to erase memories as needed beforehand: some paged memories look like NOR-memory when using certain programmers, meaning programming cannot set bits to 1 (eg, flash under most programmers). These memories need to be erased beforehand using the erase command (see below). The format only needs to be specified if it cannot be automatically detected, eg, when the file is - for standard input. *Memlist* can be a comma separated list of memories just as in the *-U* command line argument. **restore** flushes the cache before writing memories and resets the cache after writing memories. Note that restoring read-only memories verifies file contents with the corresponding microprocessor's memories.

verify memlist *file[:format]*

Compare one or more memories with the specified file. *Memlist* can be a comma separated list of memories just as in the *-U* command line argument. **verify** flushes the cache before verifying memories.

erase Perform a chip erase and discard all pending writes to flash, EEPROM and bootrow. Note that EEPROM will be preserved if the EESAVE fuse bit is active, ie, had a corresponding value at the last reset prior to the operation.

erase memory

Erase the entire specified memory.

erase memory *addr len*

Erase a section of the specified memory.

flush Synchronise with the device all pending writes to flash, EEPROM, bootrow and usersig. With some programmer and part combinations, flash (and sometimes EEPROM, too) looks like a NOR memory, i.e., a write can only clear bits, never set them. For NOR memories a page erase or, if not available, a chip erase needs to be issued before writing arbitrary data. Usersig is unaffected by a chip erase. When a memory looks like a NOR memory, either page erase is deployed (e.g., with parts that have PDI/UPDI interfaces), or if that is not available, both EEPROM and flash caches are fully read in, a chip erase command is issued and both EEPROM and flash are written back to the device. Hence, it can take minutes to ensure that a single previously cleared bit is set and, therefore, this routine should be called sparingly.

abort Normally, caches are only ever actually written to the device when using **flush**, at the end of the terminal session after typing **quit**, or after EOF on input is encountered. The **abort** command resets the cache discarding all previous writes to the flash, EEPROM, bootrow and usersig cache.

config [-f|-a|-v]

Show all configuration properties of the part; these are usually bitfields in fuses or lock bits bytes that can take on values, which typically have a mnemonic name. Each part has their own set of configurable items. The option **-f** groups the configuration properties by the fuses and lock bits byte they are housed in, and shows the current value of these memories as well. Config **-a** outputs an initialisation script with all properties and all possible respective assignments. The currently assigned mnemonic values are the ones that are not commented out. The option **-v** increases the verbosity of the output of the config command.

config [-f|-v] property [-f|-v]

Show the current value of the named configuration property. Wildcards or initial strings are permitted (but not both), in which case the current values of all matching properties are displayed.

config [-f|-v] property= [-f|-v]

Show all possible values of the named configuration property (notice the trailing **=**). The one that is currently set is the only one not commented out. As before, wildcards or initial strings are permitted.

config [-f|-v|-c] property=value [-f|-v|-c]

Modify the named configuration property to the given value. The corresponding fuse or lock bits will be changed immediately but the change will normally only take effect the next time the part is reset, at which point the fuses and lock bits are utilised. Value can either be a valid integer or one of the symbolic mnemonics, if known. Wildcards or initial strings are permitted for either the property or the assigned mnemonic value, but an assignment only happens if both the property and the name can be uniquely resolved. Option **-v** shows the value of the assigned configuration property by reading it again from the fuse. In absence of **-v** the option **-c** confirms the new value of the configuration property only if it has changed.

It is quite possible, as is with direct writing to the underlying fuses and lock bits, to brick a part, i.e., make it unresponsive to further programming with the chosen programmer: here be dragons.

factory reset

Resets the connected part to factory state as far as possible (bootloaders, for example, cannot write fuses and may not have a means to erase EEPROM). This command may change the clock frequency `F_CPU` of the part after the next MCU reset when the changed fuse values come into effect. As such, this may require that future avrdude calls use a different bit clock rate up to `F_CPU/4` for the programmer next time. Note that the command **factory** can be abbreviated but the required argument **reset** needs to be spelled out in full.

regfile [opts]

regfile with no further argument displays the register file of a part, i.e., all register names and their contents in `io` memory, if possible: note that external programming cannot read the registers of classic parts (ISP or TPI interfaces).

Option **-a** displays the register I/O addresses in addition; **-m** displays the register memory addresses used for `lds/sts` opcodes instead of the I/O addresses. Option **-s** also shows the size of the register in bytes whilst **-v** shows a slightly expanded register explanation alongside each register.

regfile [opts] reg [opts]

regfile together with a register name *reg* shows all those registers that are matched by *reg*. Wildcards or partial strings are permitted but not both. Register names have the form *module.name* or *module.instance.name*. If the provided *reg* is a full, existing register name, e.g., `porta.out` then that is the only register that is displayed even though that might be a partial name of another register, eg, `porta.outdir`. If the provided *reg* is the same as *instance.name* or *name* then partial matching is no longer utilised and all module registers with that exact *instance.name* or *name* are shown. Partial matching can be forced through use of wildcards, e.g., using `porta.out*`

regfile [opts] reg=value [opts]

This sets a single register addressed by *reg* to the given *value*. Only external programming of modern parts (those with UPDI interface) can read from and write to register `io` memory, but as that memory is volatile, the contents will be lost after reset.

include [opts] file

Include contents of the named file *file* as if it was typed. This is useful for batch scripts, e.g., recurring initialisation code for fuses. The include option **-e** prints the lines of the file as comments before processing them; on a non-zero verbosity level the line numbers are printed, too.

signature

Display the device signature bytes.

part [*opts*]

Display the current part information, including supported programming modes, memory and variants tables. Use **-m** to only print the memory table, and **-v** to only print the variants table.

verbose [*level*]

Change (when *level* is provided), or display the verbosity level. The initial verbosity level is the number of **-v** options on the command line.

quell [*level*]

Change (when *level* is provided), or display the quell level. 1 is used to suppress progress reports. 2 or higher yields progressively quieter operations. The initial quell level is the number of **-q** options on the command line.

help

Give a short on-line summary of the available commands.

quit

Leave terminal mode and thus AVRDUDE.

!line

Run the shell *line* in a subshell, e.g., **!ls *.hex**. Subshell commands take the rest of the line as their command. For security reasons, they must be enabled explicitly by putting **allow_subshells = yes;** into your `${HOME}/.config/avrdude/avrdude.rc` or `${HOME}/.avrduderc` file.

comment Place comments onto the terminal line (useful for scripts).

d, r, w, q and **?** abbreviate **dump**, **read**, **write**, **quit** and **help**. Other than that any unique abbreviation of a command works, too. **cmd -?** gives more details about a command *cmd*.

In addition, the following commands are supported on some programmers:

pgerase *memory addr*

Erase one page of the memory specified.

send *b1 b2 b3 b4*

Send raw instruction codes to the AVR device. If you need access to a feature of an AVR part that is not directly supported by AVRDUDE, this command allows you to use it, even though AVRDUDE does not implement the command. When using direct SPI mode, up to 3 bytes can be omitted.

spi

Enter direct SPI mode. The *pgmled* pin acts as chip select. *Only supported on parallel bitbang programmers, and partially by USBtiny*. Chip Select must be externally held low for direct SPI when using USBtinyISP, and send must be a multiple of four bytes.

pgm

Return to programming mode (from direct SPI mode).

vtarg *voltage*

Set the target's supply voltage to *voltage* Volts.

varef [*channel*] *voltage*

Set the adjustable voltage source to *voltage* Volts. This voltage is normally used to drive the target's *Aref* input on the STK500 and STK600. The STK600 offers two reference voltages, which can be selected by the optional parameter *channel* (either 0 or 1).

fosc freq[M|k]

Set the programming oscillator to *freq* Hz. An optional trailing letter **M** multiplies by 1,000,000, a trailing letter **k** by 1000.

fosc off Turn the programming oscillator off.

sck period

Set the SCK clock period to *period* microseconds. Note that some official Microchip programmers store the bitclock setting and will continue to use it until a diferent value is provided. See **-B bitclock** for more information.

parms Display programmer specific parameters.

3.2 Terminal Mode Examples

Enter terminal, display part parameters, modify EEPROM, perform a chip erase and quit:

```
$ avrdude -qq -c usbasp -p atmega328p -t

avrdude> part
ATmega328P with programming modes ISP, HVPP, debugWIRE, SPM

Memory          Size  Pg size
-----
eeprom           1024      4
flash           32768     128
efuse              1        1
hfuse              1        1
lfuse              1        1
lock              1        1
signature         3        1
calibration       1        1
io                 224        1
sram              2048        1

Variants          Package  F max   T range      V range
-----
ATmega328P        N/A      20 MHz  [-40 C,  N/A] [1.8 V, 5.5 V]
ATmega328P-15MZ   MLF32    20 MHz  [-40 C,  85 C] [1.8 V, 5.5 V]
ATmega328P-AN     TQFP32   20 MHz  [-40 C, 105 C] [1.8 V, 5.5 V]
ATmega328P-ANR    TQFP32   20 MHz  [-40 C, 105 C] [1.8 V, 5.5 V]
ATmega328P-AU     TQFP32   20 MHz  [-40 C,  85 C] [1.8 V, 5.5 V]
ATmega328P-AUR    TQFP32   20 MHz  [-40 C,  85 C] [1.8 V, 5.5 V]
ATmega328P-MMH    MLF28    20 MHz  [-40 C,  85 C] [1.8 V, 5.5 V]
ATmega328P-MMHR   MLF28    20 MHz  [-40 C,  85 C] [1.8 V, 5.5 V]
ATmega328P-MN     QFN32    20 MHz  [-40 C, 105 C] [1.8 V, 5.5 V]
ATmega328P-MNR    MLF32    20 MHz  [-40 C, 105 C] [1.8 V, 5.5 V]
ATmega328P-MU     MLF32    20 MHz  [-40 C,  85 C] [1.8 V, 5.5 V]
ATmega328P-MUR    MLF32    20 MHz  [-40 C,  85 C] [1.8 V, 5.5 V]
ATmega328P-PN     PDIP28   20 MHz  [-40 C, 105 C] [1.8 V, 5.5 V]
ATmega328P-PU     PDIP28   20 MHz  [-40 C,  85 C] [1.8 V, 5.5 V]

avrdude> dump eeprom 0 16
0000  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.....|

avrdude> write eeprom 0 1 2 3 4 0xcafe "Avrdude"

avrdude> dump eeprom 0 16
0000  01 02 03 04 fe ca 41 76  72 64 75 64 65 00 ff ff  |.....Avrdude...|

avrdude> flush

avrdude> erase
erasing chip ...

avrdude> dump eeprom 0 16
0000  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.....|

avrdude> quit
```

Program the fuse bits of an ATmega328P with a fuse calculator

- Enable full-swing high speed external crystal with long startup time
- Remove default clock division by 8
- Make reset jump to bootloader
- Set the size of the bootloader to 512 bytes (256 words)
- Enable brown-out detection at 2.7 V

First display the factory defaults, then consult an external fuse calculator, select the ATmega328P part, find above settings, note the ensuing new values for the three fuses and reprogram:

```
$ avrdude -c usbasp -p atmega328p -t

avrdude> dump efuse
Reading | ##### | 100% 0.00 s
0000 ff |. |

avrdude> dump hfuse
Reading | ##### | 100% 0.00 s
0000 d9 |. |

avrdude> dump lfuse
Reading | ##### | 100% 0.00 s
0000 62 |b |

avrdude> #
avrdude> # Consult external fuse calculator
avrdude> #
avrdude> #

avrdude> write efuse 0xfd
Writing | ##### | 100% 0.01 s

avrdude> write hfuse 0xde
Writing | ##### | 100% 0.01 s

avrdude> write lfuse 0xf7
Writing | ##### | 100% 0.01 s

avrdude> quit

Avrdude done. Thank you.
```

Program the fuse bits of an ATmega328P with the config command

```
$ avrdude -qq -c usbasp -p atmega328p -t

avrdude> # Show all configurations
avrdude> config
config sut_cksel=intrcosc_8mhz_6ck_14ck_65ms # 34
config ckout=co_disabled # 1
config ckdiv8=by_8 # 0
config bootrst=application # 1
config bootasz=bs_2048w # 0
config eesave=ee_erased # 1
config wdtton=wdt_programmable # 1
config spien=isp_enabled # 0
config dwen=dw_off # 1
config rstdisbl=external_reset # 1
config bodlevel=bod_disabled # 7
config lb=no_lock # 3
config blb0=no_lock_in_app # 3
config blb1=no_lock_in_boot # 3

avrdude> # Show possible values for full-swing external crystal
avrdude> config sut_cksel=extfs
avrdude warning: (config) ambiguous; known sut_cksel extfs symbols are:
- sut_cksel=extfsxtal_258ck_14ck_4ms1 # 6
- sut_cksel=extfsxtal_1kck_14ck_65ms # 7
- sut_cksel=extfsxtal_258ck_14ck_65ms # 22
- sut_cksel=extfsxtal_16kck_14ck_0ms # 23
- sut_cksel=extfsxtal_1kck_14ck_0ms # 38
- sut_cksel=extfsxtal_16kck_14ck_4ms1 # 39
- sut_cksel=extfsxtal_1kck_14ck_4ms1 # 54
- sut_cksel=extfsxtal_16kck_14ck_65ms # 55

avrdude> # Set the one with appropriate startup times
avrdude> c su=55

avrdude> # Unprogram clock division by 8, make reset jump to bootloader
avrdude> c ckdiv8=1
avrdude> c bootrst=boot
avrdude> c bootasz=bs_256w

avrdude> # Query which bod levels exist; set brown-out at 2.7 V
avrdude> c bodlevel=
# conf bodlevel=bod_4v3 # 4
# conf bodlevel=bod_2v7 # 5
# conf bodlevel=bod_1v8 # 6
config bodlevel=bod_disabled # 7 (factory)
avrdude> c bod=*2v7

avrdude> quit
```

Show and change registers

```
$ avrdude -c xplainedmini_updi -p ATtiny817 \
-T "reg ctrlc" -T "reg usart0.baud=0x1234" -T "reg -asv usart0"

Processing -T reg ctrlc
0x00 portmux.ctrlc
0x00 adc0.ctrlc
0x03 usart0.ctrlc
0x00 tca0.ctrlc
0x00 tcd0.ctrlc

Processing -T reg usart0.baud=0x1234

Processing -T reg -asv usart0
I/O 0x800: (1) 0x00 usart0.rxdatal # Receive data low byte
I/O 0x801: (1) 0x00 usart0.rxdath # Receive data high byte
I/O 0x802: (1) 0x00 usart0.txdata1 # Transmit data low byte
I/O 0x803: (1) 0x00 usart0.txdatah # Transmit data high byte
I/O 0x804: (1) 0x20 usart0.status # Status register
I/O 0x805: (1) 0x00 usart0.ctrla # Control register A
I/O 0x806: (1) 0x00 usart0.ctrlb # Control register B
I/O 0x807: (1) 0x03 usart0.ctrlc # Control register C
I/O 0x808: (2) 0x1234 usart0.baud # Baud rate register (16 bits)
I/O 0x80b: (1) 0x00 usart0.dbgctrl # Debug control register
I/O 0x80c: (1) 0x00 usart0.evctrl # Event control register
I/O 0x80d: (1) 0x00 usart0.txplctrl # IRCOM transmitter pulse length control register
I/O 0x80e: (1) 0x00 usart0.rxplctrl # IRCOM receiver pulse length control register

Avrdude done. Thank you.
```

Show register file of a classic part

```
$ avrdude -qq -p ATtiny11 -c dryrun -T "regfile -av"

I/O 0x08: ac.acsr # Analog comparator control and status register
I/O 0x16: portb.pinb # Port B input register
I/O 0x17: portb.ddrb # Port B data direction register
I/O 0x18: portb.portb # Port B data register
I/O 0x21: wdt.wdtcr # Watchdog timer control register
I/O 0x32: tc0.tcnt0 # Timer/counter 0
I/O 0x33: tc0.tccr0 # T/C 0 control register
I/O 0x34: cpu.mcusr # MCU status register
I/O 0x35: cpu.mcucr # MCU control register
I/O 0x38: tc0.tifr # T/C interrupt flag register
I/O 0x39: tc0.timsk # T/C interrupt mask register
I/O 0x3a: exint.gifr # General interrupt flag register
I/O 0x3b: exint.gimsk # General interrupt mask register
I/O 0x3f: cpu.sreg # Status register
```

The following example demonstrates **negative address and length bytes**, and the **fill form of the write command using an ellipsis ...** where the last data item provided is used to fill up the indicated memory range.

```
$ avrdude -c usbasp -p atmega328p -t

avrdude> dump flash -64 -33
Reading | ##### | 100% 0.02 s
7fc0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|
7fd0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|

avrdude> dump flash -32 -1
Reading | ##### | 100% 0.00 s
7fe0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|
7ff0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|

avrdude> write flash -64 1234567890 'A' 'V' 'R' 2.718282 0xaa 0xbb 0xcc "Hello World!"
Caching | ##### | 100% 0.00 s

avrdude> write flash -32 -1 0x01 0b00000010 0b11 0x04 0x05 ...
Caching | ##### | 100% 0.00 s

avrdude> read flash -64 -33
Reading | ##### | 100% 0.00 s
7fc0 d2 02 96 49 41 56 52 55 f8 2d 40 aa bb cc 48 65 |...IAVRU.-@...He|
7fd0 6c 6c 6f 20 57 6f 72 6c 64 21 00 ff ff ff ff ff |llo World!.....|

avrdude> read flash -32 -1
Reading | ##### | 100% 0.00 s
7fe0 01 02 03 04 05 05 05 05 05 05 05 05 05 05 05 |.....|
7ff0 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 |.....|

avrdude> flush
Synching cache to device ...
Writing | ##### | 100% 0.05 s

avrdude> quit

Avrdude done. Thank you.
```

Disassemble the flash contents of an ATtiny13A, write the output to file `blink.S`, compile to `blink.elf` and verify that the flash contents of the ATtiny13A is the same as the one given by the compiled `blink.elf`. Then `cat` the disassembled `blink.S` to stdout.

```
$ avrdude -qq -p t13a -c avrisp2 -T "disasm -g flash 0 -1" >blink.S
$ avr-gcc -mmcu=attiny13a -nostdlib -Wl,--section-start=.text=0x0000 blink.S -o bl.elf
$ avrdude -qq -p t13a -c avrisp2 -U flash:v:bl.elf && echo OK
OK

$ cat blink.S

        .equ      io.pinb, 0x16
        .equ      io.ddrb, 0x17

        .text
main:
L000:   rjmp      Label1                ; L016
L002:   rjmp      Label0                ; L014
L004:   rjmp      Label0                ; L014
L006:   rjmp      Label0                ; L014
L008:   rjmp      Label0                ; L014
L00a:   rjmp      Label0                ; L014
L00c:   rjmp      Label0                ; L014
L00e:   rjmp      Label0                ; L014
L010:   rjmp      Label0                ; L014
L012:   rjmp      .+0                  ; L014

; Rjmp from L002, L004, L006, L008, L00a, L00c, L00e, L010
Label0:
L014:   reti

Label1:
L016:   sbi       io.ddrb, 2            ; Rjmp from L000
                                           ; Bit 2 = 0x04

Label2:
L018:   ldi       r29, 0x1e             ; Rjmp from L024
                                           ; 30

Label3:
L01a:   sbiw      r30, 0x01              ; Brne from L01c, L020
                                           ; 1
L01c:   brne      Label3                ; L01a
L01e:   dec       r29
L020:   brne      Label3                ; L01a
L022:   sbi       io.pinb, 2            ; Bit 2 = 0x04
L024:   rjmp      Label2                ; L018

L026:   .fill     493, 2, 0xffff
```

Mixing terminal commands and -U memory operations: the example below burns a boot-loader, uses a terminal line to write application data to flash, loads the application, configures the brownout detection level to 2.7 V and, finally, stores the full flash as new hex file. Note the use of different quotation marks in **bash** to pass the terminal command lines as single entity to AVRDUDE.

```
$ avrdude -qc dryrun -p m328p \  
-U urboot_m328p_1s_autobaud_uart0_pr_ee_ce.hex \  
-T 'write flash 0x7D00 0xc0cac01a 0xcafe "secret Coca Cola recipe"' \  
-U flash:w:cola-vending-machine.hex \  
-T "config -v bod=*2v7" \  
-U flash:r:app+data.hex:I  
  
Processing -U flash:w:urboot_m328p_1s_autobaud_uart0_pr_ee_ce.hex:i  
Reading 368 bytes for flash from input file urboot_m328p_1s_autobaud_uart0_pr_ee_ce.hex  
Writing 368 bytes to flash, 368 bytes written, 368 verified  
  
Processing -T write flash 0x7D00 0xc0cac01a 0xcafe "secret Coca Cola recipe"  
Synching cache to device ... done  
  
Processing -U flash:w:cola-vending-machine.hex:i  
Reading 736 bytes for flash from input file cola-vending-machine.hex  
Writing 736 bytes to flash, 736 bytes written, 736 verified  
  
Processing -T config -v bod=*2v7  
config bodlevel=bod_2v7 # 5  
  
Processing -U flash:r:app+data.hex:I  
Reading flash memory ...  
Writing 32768 bytes to output file app+data.hex  
  
Avrdude done.  Thank you.
```

4 Configuration Files

AVRDUDE reads a configuration file upon startup which describes all of the parts and programmers that it knows about. The advantage of this is that if you have a chip that is not currently supported by AVRDUDE, you can add it to the configuration file without waiting for a new release of AVRDUDE. Likewise, if you have a parallel port programmer that is not supported, chances are that you can copy an existing programmer definition and, with only a few changes, make your programmer work.

AVRDUDE first looks for a system wide configuration file in a platform dependent location. On Unix, this is usually `/usr/local/etc/avrdude.conf`. See Section A.1 [Unix], page 74, whilst on Windows it is usually in the same location as the executable file. The full name of this file can be specified using the `-C` command line option. After parsing the system wide configuration file, AVRDUDE looks for a per-user configuration file to augment or override the system wide defaults. On Unix, the per-user file is `${XDG_CONFIG_HOME}/avrdude/avrdude.rc`, whereas if `${XDG_CONFIG_HOME}` is either not set or empty, `${HOME}/.config/` is used instead. If that does not exist `.avrduderc` within the user's home directory is used. On Windows, this file is the `avrdude.rc` file located in the same directory as the executable.

4.1 AVRDUDE Defaults

```
avrdude_conf_version = "build-time-version";
    Automatically set during the build process.

default_parallel = "default-parallel-device";
    Assign the default parallel port device. Can be overridden using the -P option.

default_serial = "default-serial-device";
    Assign the default serial port device. Can be overridden using the -P option.

default_linuxgpio = "default-linuxgpio-device";
    Assign the default gpiochip for linuxgpio's libgpiod mode, e.g. "gpiochip0".
    Ignored for linuxgpio's sysfs mode. Can be overridden using the -P option.

default_programmer = "default-programmer-id";
    Assign the default programmer id. Can be overridden using the -c option.

default_baudrate = "default-baudrate";
    Assign the default baudrate value that will be used if the programmer does not
    provide its specific baudrate entry. Can be overridden using the -b option.

default_bitclock = "default-bitclock";
    Assign the default bitclock value. Can be overridden using the -B option.

allow_subshells = no;
    Whether or not AVRDUDE's interactive terminal is allowed to use subshell
    ! commands. This defaults to no for security reasons, e.g., in the rare case
    avrdude -t is set up with attached hardware to provide a web service, remote
    ssh or a login on a PC instead of a shell, say, for demo or training purposes. In
    almost all other cases this can be overridden in the per-user avrdude.rc or
    .avrduderc configuration file with yes.
```

4.2 Programmer Definitions

The format of the programmer definition is as follows:

```

programmer
  parent <id>                                # optional parent
  id      = <id1> [, <id2> ... ];            # <idN> are quoted strings
  desc    = <description>;                   # quoted string
  type    = <type>;                          # programmer type, quoted string
                                              # list known types with -c ?type

  prog_modes = PM_<i/f> {| PM_<i/f>}         # interfaces, e.g., PM_SPM|PM_PDI (1)
  is_serialadapter = <yes|no>                # programmer is also a serialadapter
  extra_features = HAS_<fea> {| HAS_<fea>}   # extra features, e.g., HAS_SUFFER (2)
  connection_type = parallel | serial | usb | spi | linuxgpio
  port     = <port>                          # default port, quoted string
  baudrate = <num>;                          # baudrate for the programmer
  vcc      = <pin1> [, <pin2> ... ];          # pin number(s) (3)
  buff     = <pin1> [, <pin2> ... ];          # pin number(s)
  reset    = <pin>;                          # pin number
  sck      = <pin>;                          # pin number
  sdolpico = <pin>;                          # pin number
  sdi|poci = <pin>;                          # pin number
  tck      = <pin>;                          # pin number
  tdi      = <pin>;                          # pin number
  tdo      = <pin>;                          # pin number
  tms      = <pin>;                          # pin number
  errled   = <pin>;                          # pin number
  rdyled   = <pin>;                          # pin number
  pgmled   = <pin>;                          # pin number
  vfyled   = <pin>;                          # pin number
  usbvid   = <hexnum>;                       # USB vendor ID
  usbpid   = <hexnum> [, <hexnum> ...];      # USB product ID (4)
  usbdev   = <interface>;                   # USB interface or other device info
  usbvendor = <vendorname>;                 # USB vendor name
  usbproduct = <productname>;               # USB product name
  usbsn    = <serialno>;                     # USB serial number
  hvupdi_support = <num> [, <num>, ... ];    # HV support for enabling UPDI
;

```

If a parent is specified, all settings of it (except its ids) are used for the new programmer. These values can be changed by setting them for the new programmer.

Programmer definitions can be modified using the following syntax

```

modify programmer <id>
  # Any property as above, for example
  port     = <port>                          # default port, quoted string
  usbsn    = <serialno>;                     # USB serial number
;

```

The `modify` keyword makes most sense in the per-user configuration file, where users may want to add specific properties of their programmer. This could be its USB serial number or default port, eg, a serialadapter such as "ch340". This technique must not be used in the system `avrdude.conf` file, as the developer option `-c <programmer>/S` that prints the programmer definition as AVRDUDE understands it will only consider the final definition; it will not print modify statements.

Notes

1. Known programming modes are
 - PM_SPM: Bootloaders, self-programming with SPM opcodes or NVM Controllers

- PM_TPI: Tiny Programming Interface (t4, t5, t9, t10, t20, t40, t102, t104)
 - PM_ISP: SPI programming for In-System Programming (almost all classic parts)
 - PM_PDI: Program and Debug Interface (xmega parts)
 - PM_UPDI: Unified Program and Debug Interface
 - PM_HVSP: High Voltage Serial Programming (some classic parts)
 - PM_HVPP: High Voltage Parallel Programming (most non-HVSP classic parts)
 - PM_debugWIRE: Simpler alternative to JTAG (a subset of HVPP/HVSP parts)
 - PM_JTAG: Joint Test Action Group standard (some classic parts)
 - PM_JTAGmkI: Subset of PM_JTAG, older parts, Atmel ICE mkI
 - PM_XMEGAJTAG: JTAG, some XMEGA parts
 - PM_AVR32JTAG: JTAG for 32-bit AVR
 - PM_aWire: AVR32 parts
2. The following extra programmer features are known
 - HAS_SUFFER: Only present on Xplained Mini/Nano programmers; the Super User Fantastic Feature Enable Register allows the user to modify the behavior of the mEDBG programmer/debugger chip, see the Xplained Mini/Nano documentation for more information
 - HAS_VTARG_SWITCH: Programmer has a programmable target power switch
 - HAS_VTARG_READ: Programmer can read the target voltage
 - HAS_VTARG_ADJ: Programmer has an adjustable target power source that can be controlled with Avrdude
 - HAS_FOSC_ADJ: Programmer has a programable frequency generator that can clock an AVR directly through its XTAL1 pin
 - HAS_VAREF_ADJ: Programmer has an adjustable analog reference voltage that can be controlled with Avrdude
 3. To invert the polarity of a pin, use a tilde ~<num>; to invert the polarity of all pins in a list use ~(<num1> [, <num2> ...])
 4. Not all programmer types can handle a list of USB PIDs

The following programmer types are currently implemented:

4.3 Serial Adapter Definitions

The format of a serial adapter definition is as follows:

```
serialadapter
  parent <id>                                # optional serialadapter or programmer parent
  id      = <id1> [, <id2> ... ];            # <idN> are quoted strings
  desc    = <description>;                   # quoted string
  baudrate = <num>;                          # optional default baudrate, eg, in .avrduderc
  usbvid   = <hexnum>;                       # USB vendor ID
  usbpid   = <hexnum> [, <hexnum> ...];      # list of USB product IDs
  usbsn    = <serialno>;                     # USB serial number in per-user .avrduderc
;
```

Technically, a `serialadapter` is implemented as `programmer` that has only USB parameters defined. It can be used for a `-P <serialadapter>[:<serial number>]` port specification instead of the created serial port. Per-user `serialadapter` definitions in `~/.avrduderc` or `avrdude.rc` files can add a serial number to assign a particular board a specific id and default communication baud rate:

```
serialadapter parent "ft232r"
    id                = "bike-shed-door";
    usbsn             = "0123456789";
    baudrate          = 250000;
;
```

This is particularly useful for programming through a bootloader as it allows specifying the port as `-P bike-shed-door` rather than having to figure out which serial port name the operating system has assigned to the plugged in bike-shed-door board at runtime. Note that each programmer that defines `usbpid` and sets `is_serialadapter = yes` can also be utilised as a `serialadapter`.

4.4 Part Definitions

```
part
    desc                = <description>;                # quoted string, the long part name, eg, "ATmega328p"
    id                  = <id>;                          # quoted string, normally an abbreviated part name
    variants            = <str1> [, <str2> ...];          # quoted strings, each starts so "<alt-name>: ..."
    family_id           = <id>;                          # quoted string, e.g., "megaAVR" or "tinyAVR"
    prog_modes          = PM_<i/f> {| PM_<i/f>};          # interfaces, e.g., PM_SPM|PM_ISP|PM_HVPP|PM_debugWIRE
    mcuid               = <num>;                          # unique id in 0..2039 for 8-bit AVR
    archnum             = <num>;                          # avr-gcc architecture number; -1 if not 8-bit AVR
    n_interrupts        = <num>;                          # number of interrupts, used for vector bootloaders
    n_page_erase        = <num>;                          # if set, number of pages erased during SPM erase
    n_boot_sections     = <num>;                          # Number of boot sections
    boot_section_size   = <num>;                          # Size of (smallest) boot section, if any
    hvupdi_variant      = <num>;                          # numeric: -1 or 0..3; how to enable UPDI with HV
    stk500_devcode      = <num>;                          # numeric
    avr910_devcode      = <num>;                          # numeric
    is_at90s1200        = <yes/no>;                      # AT90S1200 part
    signature           = <num> <num> <num>;            # signature bytes
    usbpid              = <num>;                          # DFU USB PID
    chip_erase_delay    = <num>;                          # microseconds
    reset               = dedicated | io;
    retry_pulse         = reset | sck;
    # STK500 parameters (parallel programming IO lines)
    pagel               = <num>;                          # page load pin name in hex, e.g., 0xD7
    bs2                 = <num>;                          # byte select 2 pin name in hex, e.g., 0xA0
    serial              = <yes/no>;                      # can use serial programming
    parallel            = <yes/no/pseudo>;               # can use parallel programming
    # STK500v2 parameters, to be taken from Atmel's ATDF files
    timeout             = <num>;
    stabdelay           = <num>;
    cmdexedelay         = <num>;
    synchloops          = <num>;
    bytedelay           = <num>;
    pollvalue           = <num>;
    pollindex           = <num>;
    predelay            = <num>;
    postdelay           = <num>;
    pollmethod          = <num>;
```

```

hvspcmdexedelay = <num>;
# STK500v2 HV programming parameters, from ATDFs
pp_controlstack = <num>, <num>, ...;      # PP only
hvsp_controlstack = <num>, <num>, ...;     # HVSP only
flash_instr      = <num>, <num>, <num>;
eeprom_instr     = <num>, <num>, ...;
hventerstabeldelay = <num>;
progmodedelay    = <num>;                  # PP only
latchcycles      = <num>;
togglevtg        = <num>;
poweroffdelay    = <num>;
resetdelaysms    = <num>;
resetdelayus     = <num>;
hvleavestabeldelay = <num>;
resetdelay       = <num>;
synchcycles      = <num>;                  # HVSP only
chiperasepulsewidth = <num>;              # PP only
chiperasepolltimeout = <num>;
chiperasetime     = <num>;                  # HVSP only
programfusepulsewidth = <num>;            # PP only
programfusepolltimeout = <num>;
programlockpulsewidth = <num>;            # PP only
programlockpolltimeout = <num>;
# debugWIRE and/or JTAG ICE mkII parameters, also from ATDF files
allowfullpagebitstream = <yes/no>;
enablepageprogramming = <yes/no>;
idr              = <num>;                  # IO addr of IDR (OCD) reg
rampz            = <num>;                  # IO addr of RAMPZ reg
spmcr           = <num>;                  # mem addr of SPMC[S]R reg
eecr            = <num>;                  # mem addr of EECR reg
eind            = <num>;                  # mem addr of EIND reg
mcu_base        = <num>;                  # MCU control block in ATxmega devices
nvm_base        = <num>;                  # NVM controller in ATxmega devices
ocd_base        = <num>;                  # OCD module in AVR8X/UPDI devices
syscfg_base     = <num>;                  # Chip revision ID in AVR8X/UPDI devices
ocdrev          = <num>;                  # JTAGICE3 parameter from ATDF files
pgm_enable      = <instruction format>;
chip_erase      = <instruction format>;
# parameters for bootloaders
autobaud_sync   = <num>;                  # autobaud detection byte, default 0x30
factory_fcps    = <num>;                  # F_CPU in Hz on reset and factory-set fuses

memory <memstr>
    paged        = <yes/no>;              # yes/no (flash of classic parts only)
    offset       = <num>;                  # memory offset
    size         = <num>;                  # bytes
    page_size    = <num>;                  # bytes
    num_pages    = <num>;                  # numeric
    initval      = <num>;                  # factory setting of fuses and lockbits
    bitmask      = <num>;                  # bits used (only in fuses and lockbits)
    n_word_writes = <num>;                  # TPI only: if set, number of words to write
    min_write_delay = <num>;              # micro-seconds
    max_write_delay = <num>;              # micro-seconds
    readback     = <num> <num>;           # pair of byte values
    readback_p1  = <num>;                  # byte value (first component)
    readback_p2  = <num>;                  # byte value (second component)
    pwoff_after_write = <yes/no>;         # yes/no
    mode         = <num>;                  # STK500 v2 file parameter from ATDF files

```

```

delay          = <num>;          # "
blocksize      = <num>;          # "
readsize       = <num>;          # "
read           = <instruction format>;
write          = <instruction format>;
read_lo        = <instruction format>;
read_hi        = <instruction format>;
write_lo       = <instruction format>;
write_hi       = <instruction format>;
loadpage_lo    = <instruction format>;
loadpage_hi    = <instruction format>;
writepage      = <instruction format>;
;

```

If any of the above parameters are not specified, the default value of 0 is used for numerics (except for `mcuid`, `hvpudi_variant`, `ocdrev`, `initval` and `bitmask`, all of which default to -1, and for `autobaud_sync` which defaults to 0x30) or the empty string "" for string values. If a required parameter is left empty, AVRDUDE will complain. Almost all occurrences of numbers (with the exception of pin numbers and where they are separated by space, e.g., in signature and readback) can also be given as simple expressions involving arithmetic and bitwise operators.

Part definitions can be modified using the following syntax

```

modify part <id>
    # Any part property as above, for example
    chip_erase_delay = <num>;          # microseconds
;

```

The `modify` keyword makes most sense in the per-user configuration file, where users may want to modify specific properties of a part for debugging. This technique must not be used in the system `avrdude.conf` file, as the developer option `-p <part>/S` that prints the part definition as AVRDUDE understands it will only consider the final definition; it will not print `modify` statements.

4.4.1 Parent Part

Parts can also inherit parameters from previously defined parts using the following syntax. In this case specified integer and string values override parameter values from the parent part. New memory definitions are added to the definitions inherited from the parent. If, however, a new memory definition refers to an existing one of the same name for that part then, from v7.1, the existing memory definition is extended, and components overwritten with new values. Assigning `NULL` removes an inherited SPI instruction format, memory definition, control stack, eeprom or flash instruction, e.g., as in `memory "efuse" = NULL;`. The `variants` parameter is never inherited as it almost always would be a mistake to do so: `variants` defines a string list detailing variant names of the part followed by an optional colon, the package code and some absolute maximum ratings.

Example format for part inheritance:

```

part parent <id>                                # String identifying parent
    id          = <id>;                          # Id string for new part
    <any set of other parameters from the list above>
;

```

4.4.2 Instruction Format

Instruction formats are specified as a comma separated list of string values containing information (bit specifiers) about each of the 32 bits of the instruction. Bit specifiers may be one of the following formats:

1	The bit is always set on input as well as output
0	The bit is always clear on input as well as output
x	The bit is ignored on input and output
a	The bit is an address bit, the bit-number matches this bit specifier's position within the current instruction byte
aN	The bit is the Nth address bit, bit-number = N, i.e., a12 is address bit 12 on input, a0 is address bit 0.
i	The bit is an input data bit; as with a bits an input data bit can optionally be followed by a bit number, here between 0 and 7, if the bit needs to be moved to a different position in the SPI write command byte than it appears in memory.
o	The bit is an output data bit; as with i bits an output data bit can optionally be followed by a bit number; this is useful in case the part's SPI read command places a particular bit into a different position than the write command put it, e.g., ATtiny22L or AT90S8535 lock bits.

Each instruction must be composed of 32 bit specifiers. The instruction specification closely follows the instruction data provided in Atmel's data sheets for their parts. For example, the EEPROM read and write instruction for an AT90S2313 AVR part could be encoded as:

```
read = "1 0 1 0 0 0 0 0 x x x x x x x x",
       "x a6 a5 a4 a3 a2 a1 a0 o o o o o o o o";

write = "1 1 0 0 0 0 0 0 x x x x x x x x",
        "x a6 a5 a4 a3 a2 a1 a0 i i i i i i i i";
```

As the address bit numbers in the SPI opcodes are highly systematic, they don't really need to be specified. A compact version of the format specification neither uses bit-numbers for address lines nor spaces. If such a string is longer than 7 characters, then the characters 0, 1, x, a, i and o will be recognised as the corresponding bit, whilst any of the characters ., -, _ or / can act as arbitrary visual separators, which are ignored. Examples:

```
loadpage_lo = "0100.0000--000x.xxxx--xxaa.aaaa--iiii.iiii";

loadpage_lo = "0100.0000", "000x.xxxx", "xxaa.aaaa", "iiii.iiii";
```

4.5 Other Notes

- The `stk500_devcode` parameter is the device code used by the STK500 and is obtained from the software section (`avr061.zip`) of Atmel's AVR061 application note available from http://www.atmel.com/dyn/resources/prod_documents/doc2525.pdf.

- Not all memories will implement all instructions.
- AVR Fuse bits and Lock bits are implemented as a type of memory.
- Example memories are: `flash`, `eeprom`, `fuse`, `lfuse` (low fuse), `hfuse` (high fuse), `efuse` (extended fuse), `signature`, `calibration`, `lock`.
- The memory specified on the AVRDUDE command line must match one of the memories defined for the specified chip.
- The `pwroff_after_write` flag causes AVRDUDE to attempt to power the device off and back on after an unsuccessful write to the affected memory area if VCC programmer pins are defined. If VCC pins are not defined for the programmer, a message indicating that the device needs a power-cycle is printed out. This flag was added to work around a problem with the at90s4433/2333's; see the at90s4433 errata at:
<https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/DataSheets/doc1042.pdf>
- The bootloader from application note AVR109 (and thus also the AVR Butterfly) does not support writing of fuse bits. Writing lock bits is supported, but is restricted to the boot lock bits (BLBxx). These are restrictions imposed by the underlying SPM instruction that is used to program the device from inside the bootloader. Note that programming the boot lock bits can result in a “shoot-into-your-foot” scenario as the only way to unprogram these bits is a chip erase, which will also erase the bootloader code.

The bootloader implements the “chip erase” function by erasing the flash pages of the application section.

Reading fuse and lock bits is fully supported.

5 Autogenerated files

Autogenerated files are *virtual* read-only files that do not exist externally but that are provided by AVRDUDE instead. They have the form *prefix:parameters* and can be used to write the contents of memory. The *prefix* tells AVRDUDE which sort of file to generate while the part after the colon is usually an underscore-separated list of parameters that determines the contents of the autogenerated file.

Currently, there are two types of autogenerated files. The first are non-trivial, realistic, random test files used for testing AVRDUDE and `-c` programmers. These files have names of the form `dry:features`. The other ones are urboot bootloaders from the <https://github.com/stefanrueger/urboot> project. They take the form `urboot:features`.

5.1 Test Files `dry:...`

A filename in the form `dry:[part]_[memlist]{_feature}[.hex]` signifies a virtual, read-only test file. The optional part name must be first, and if so, it must coincide with the chosen `-p part`. The autogenerated test files are multimemory files containing, by default, all writable memories and the **signature** memory. The optional *memlist* must be in second position, if *part* was given, or otherwise in first position. The memory list has the same syntax and semantics as one specified for the `-U` memory operation option, see [memory list], page 12.

The underscore-separated feature list, eg, `dry:random=1_holes` determines the test file contents. The following table lists possible features:

<code>init</code>	Initialise memories with human-readable patterns
<code>init=n</code>	Shortcut for <code>init_seed=n</code>
<code>random</code>	Initialise memories with random code/values
<code>random=n</code>	Shortcut for <code>random_seed=n</code>
<code>holes</code>	Put holes into the code of flash, eeprom, userrow and bootrow
<code>seed=n</code>	Seed random number generator with $n > 0$, default <code>time(NULL)</code>
<code>save=myfile.hex</code>	Save test file with chosen name
<code>save</code>	Save test contents to file with canonical file name
<code>help</code>	Show this help message and return

Notes. Init and random randomly configure flash wrt code, data and boot sections, which also changes the corresponding fuse settings in the multimemory test file. Generated Flash code will be benign, ie, its opcodes will not access I/O, SRAM or flash. Choose, eg, `seed=1` for reproducible flash configuration and output. The default *memlist* comprises of all writeable memories and **signature**. Test files, if saved on the file system, are per default written as Intel Hex format with comments. It is possible to write them as Motorola S-Record files using, eg, `save=myfile.srec:s`. A `save=` filename of `-` writes the autogenerated file to stdout.

Essentially, the contents of the test files are the same as if generated by the `-c dryrun` programmer with corresponding `-x` parameters. For details, see [dryrun], page 17.

The following example writes a non-trivial random test file to the flash of an ATtiny13A:

```
$ avrdude -cdryrun -pt13a -U dry:random=1_holes
Reading 317 bytes for flash from input file dry:random=1_holes
Writing 317 bytes to flash
Writing | ##### | 100% 0.00 s
Reading | ##### | 100% 0.00 s
317 bytes of flash verified

Avrdude done. Thank you.
```

Although the `dry:random=1_holes` test file contains all writable memories, only flash is written in above example as this is the default for simple `-U` programming. The contents of the test file can be written to `stdout` with the `save=` feature as the following example shows, while programming a multi-memory file with only eeprom does nothing:

```
$ avrdude -qq -cdryrun -pt13a -U sig:w:dry:random=1_holes_save= | cut -c1-85
:02000004008179
:1700080040202020402040204020402040204020402040204020402040A1 // 810008
:02000004008179
:04002200402020401A // 810022
:02000004008179
:0B002E004040202040204020402020C7 // 81002e
:020000040000FA
:2000BF00404C717E51511B486B84633A4194E4313A9016323E576FEEEA9BF4BC4625360849 // 000bf>
:2000DF0042E95B5E2C606FD06C857EFAF5D80935376671824A65F270285A728A2829134417 // 000df>
:0100FF007090 // 000ff>
:020000040000FA
:01017F00403F // 0017f>
:020000040000FA
:01024000407D // 00240>
:020000040000FA
:2002FD00CFFFCFF241FB1E46E1C2625408F8271B23E8E98D43760F5A2563729A0766FB3246 // 002fd>
:20031D005D0D5058E4A3302561E95AC6E09B765424821D74F821F587F0701AE1F867230179 // 0031d>
:20033D005F972ADC4A96775C582A4EEC42B07CFB06181BDB7F02F91A48FE78FB40FA26D13A // 0033d>
:20035D00EDE63F05F07565D80FBE15F9E9BB26993C9540B157EB35F1E4B350057EEF5DF7B2 // 0035d>
:20037D000BA1EAE50A0B70CB4AD07F48316447BD961F12235D1E3F1C477BEEC5FD142B7337 // 0037d>
:20039D007B5A3E4B23632F64EE24069E700911DC59AAE0D45BF2111B5E10213B70337ECDC5 // 0039d>
:2003BD00E7E30E5C01BB4F6F011901BA249B577D300B37E11E1C01F85A2970F818A4341396 // 003bd>
:1703DD0077B555CA2A4EE7981232F6E0E84D5C3DEA341ABC5D86CF39 // 003dd>
:020000040000FA
:0303FD00CFFFCF60 // 003fd>
:02000004008278
:010000006A95 // 820000
:02000004008278
:01000100FFFF // 820001
:02000004008377
:01000000FF00 // 830000
:02000004008476
:030000001E900748 // 840000
:00000001FF

$ avrdude -cdryrun -pt13a -U dry:eeprom_random=1

Avrdude done. Thank you.
```

As expected, the `-U all:w:file` writes the writable memories from the multi-memory file:

```
$ avrdude -q -cdryrun -pt13a -U all:w:dry:random=1_holes
Reading 361 bytes for multiple memories from input file dry:random=1_holes
Writing 38 bytes to eeprom, 38 bytes written, 38 verified
Writing 317 bytes to flash, 317 bytes written, 317 verified
Writing 1 byte (0x6A) to lfuse, 1 byte written, 1 verified
Writing 1 byte (0xFF) to hfuse, 1 byte written, 1 verified
Writing 1 byte (0xFF) to lock, 1 byte written, 1 verified

Avrdude done. Thank you.
```

5.2 Bootloader Files `urboot:...`

Bootloaders are specified with a filename that starts with `urboot:` followed by an underscore-separated feature list in arbitrary order, eg, `urboot:autobaud_2s`. For a complete list of possible features see below. The following example writes a suitable autobaud bootloader to the flash of an ATmega328P target:

```
$ avrdude -c dryrun -p m328p -U urboot:autobaud
Reading 260 bytes for flash from input file urboot:autobaud
Writing 260 bytes to flash
Writing | ##### | 100% 0.00 s
Reading | ##### | 100% 0.00 s
260 bytes of flash verified
Setting fuses for bootloader urboot:autobaud

Avrdude done. Thank you.
```

Autogenerated bootloaders can be used in the terminal as well:

```
$ avrdude -c dryrun -p m328p -t
avrdude> write flash urboot:autobaud
Caching | ##### | 100% 0.00 s

avrdude> quit
Synching cache to device ...
Writing | ##### | 100% 0.00 s

Avrdude done. Thank you.
```

Note that using `urboot:...` in `-U` options has the desired side effect of also setting all necessary fuse configurations for the bootloader to work correctly. This does *not* happen when using the terminal `write` command, where the necessary fuse configurations need to be requested manually.

The terminal mode is great for exploring possible bootloaders interactively:

```
$ avrdude -c dryrun -p m328p -t
avrdude> write flash urboot:autobaud_list
  Selection Size Use Vers Features  Type                Canonical file name
    _pr *256 256 u8.0 w---jPra- vector/spmready    urboot_m328p_1s_autobaud_u...
   _ee_ce_pr *378 384 u8.0 weU-jPrac vector/spmready    urboot_m328p_1s_autobaud_u...
   _ee_u4_pr  372 384 u8.0 weU-jPra- vector/spmready    urboot_m328p_1s_autobaud_u...
   _ce_u4_pr  362 384 u8.0 w-U-jPrac vector/spmready    urboot_m328p_1s_autobaud_u...
    _u4_pr  316 384 u8.0 w-U-jPra- vector/spmready    urboot_m328p_1s_autobaud_u...
   _ee_ce_u4_hw *400 512 u8.0 weU-hprac hardware-supported urboot_m328p_1s_autobaud_u...
   _ee_ce_u4_pr  414 512 u8.0 weU-jPrac vector/spmready    urboot_m328p_1s_autobaud_u...
   _ee_u4_hw  354 512 u8.0 weU-hpra- hardware-supported urboot_m328p_1s_autobaud_u...
```

The `list` feature does not generate the bootloader code but instead interactively lists possible selections. In above example, the last column is cut off for space reasons. Using successively `ee` (provide a bootloader with EEPROM r/w capability), `ce` (add chip erase emulation code to the generated bootloader) and `u4` (generate code skipping all redundant flash page writes and page erases to wear out the flash memory less and speed up the bootloading process) will then iteratively reduce the available selection:

```
avrdude> write flash urboot:autobaud_list_ee
Selection Size Use Vers Features  Type                Canonical file name
    _ce_pr *378 384 u8.0 weU-jPrac vector/spmready    urboot_m328p_1s_autobaud_uart...
    _u4_pr  372 384 u8.0 weU-jPra- vector/spmready    urboot_m328p_1s_autobaud_uart...
   _ce_u4_hw *400 512 u8.0 weU-hprac hardware-supported urboot_m328p_1s_autobaud_uart...
   _ce_u4_pr  414 512 u8.0 weU-jPrac vector/spmready    urboot_m328p_1s_autobaud_uart...
    _u4_hw  354 512 u8.0 weU-hpra- hardware-supported urboot_m328p_1s_autobaud_uart...

avrdude> write flash urboot:autobaud_list_ee_ce
Select Size Use Vers Features  Type                Canonical file name
    _pr *378 384 u8.0 weU-jPrac vector/spmready    urboot_m328p_1s_autobaud_uart0_r...
   _u4_hw *400 512 u8.0 weU-hprac hardware-supported urboot_m328p_1s_autobaud_uart0_r...
   _u4_pr  414 512 u8.0 weU-jPrac vector/spmready    urboot_m328p_1s_autobaud_uart0_r...

avrdude> write flash urboot:autobaud_list_ee_ce_u4
Sel Size Use Vers Features  Type                Canonical file name
   _hw *400 512 u8.0 weU-hprac hardware-supported urboot_m328p_1s_autobaud_uart0_rxd0...
   _pr  414 512 u8.0 weU-jPrac vector/spmready    urboot_m328p_1s_autobaud_uart0_rxd0...
```

In all lists the starred bootloaders are those that are the most feature-rich given the available used flash space, which — out of necessity — must be a multiple of flash pages or minimal boot section. These can be selected with the `best` feature. The `show` feature displays the properties of the bootloader that would be written without actually writing it; removing `show` will then write the bootloader to flash and, using `-U`, also set the necessary fuses:

```
$ avrdude -qq -c dryrun -p m328p -U urboot:autobaud_show_best_ee
378 384 u8.0 weU-jPrac vector/spmready urboot_m328p_1s_autobaud_uart0_rxd0_txd1_no...

$ avrdude -qq -c dryrun -p m328p -U urboot:autobaud_best_ee && echo OK
OK
```

Like the urboot project AVRDUDE currently only provides bootloaders for classic parts. The following table lists possible features.

<code>2s</code>	WDT timeout: 250ms, 500ms, 1s (default), 2s, 4s or 8s (*0)
<code>autobaud</code>	Bootloader adapts to host baud rate within MCU capability (*1)
<code>uart<n></code>	Hardware UART number, eg, <code>uart0</code> (default), <code>uart1</code> , ...
<code>alt<n></code>	Alternative UART I/O lines (*2)
<code>9.6kbaud</code>	Or other reasonable baud rates; also accepting baud unit
<code>16MHz</code>	Or other CPU frequencies; also accepting kHz and Hz units
<code>[xa-q]</code>	Optional F_CPU prefix designator, eg, <code>i9.2MHz</code> (*3)
	- <code>x</code> : external oscillator (default)
	- <code>i</code> : internal oscillator
	- <code>[a-h]</code> : internal oscillator that is 10% (a) to 1.25% (h) slow
	- <code>[j-q]</code> : internal oscillator that is 1.25% (j) to 10% (q) fast
<code>swio</code>	Software I/O, must specify <code>rx</code> and <code>tx</code> pins
<code>rx[a-h] [0-7]</code>	MCU receive pin for <code>swio</code> , eg, <code>rxb0</code>
<code>tx[a-h] [0-7]</code>	MCU transfer pin for <code>swio</code> , eg, <code>txb1</code>
<code>lednop</code>	If no LED is specified generate template bootloader
<code>no-led/nored</code>	Drop blinking code unless a LED is specified
<code>led[+-] [a-h] [0-7]</code>	Generate code for activity LED with polarity +/-, eg, <code>led+b5</code>
<code>dual</code>	Dual boot; must specify CS pin for external SPI flash (*4)
<code>cs[a-h] [0-7]</code>	Chip select pin for dual boot, eg, <code>csd5</code>
<code>hw</code>	Generate bootloader with hardware boot section (*5)
<code>v<n></code>	Optional vector for vector bootloader, eg, <code>v25</code> or <code>vspready</code>
<code>ee</code>	Generate bootloader with EEPROM r/w support
<code>ce</code>	Generate bootloader that can emulate a chip erase
<code>pr</code>	Generate vector bootloader with reset vector protection (*6)
<code>u1</code>	Generate bootloader that skips redundant flash page writes (*7,8)
<code>u2</code>	... and skips redundant flash page erases during emulated CE (*7,8)
<code>u3</code>	... and skips not needed flash page erases during page write (*7,8)
<code>u4</code>	... and skips empty flash page writes after page erase (*8)
<code>serialno=abc123</code>	Put serial number, eg, <code>abc123</code> , in top of unused bootloader flash
<code>fill=urboot\x20</code>	Fill otherwise unused bootloader flash repeatedly with argument
<code>save=myfile.hex</code>	Save bootloader to file with chosen name (*9)
<code>save</code>	Save bootloader to file with canonical file name (*9)
<code>tags=myfile.tag</code>	Save symbols to tag file with chosen name (*9)
<code>tags</code>	Save symbols to tag file with canonical file name (*9)
<code>configs</code>	Show needed fuse configuration but do not write to memories
<code>show</code>	Show bootloader features but do not write to flash
<code>list</code>	List possible bootloader configurations but do not write to flash
<code>best</code>	Select smallest feature-rich bootloader (first in list) and, if the baud rate error is too high for UART, switch to <code>swio</code>
<code>help</code>	Show this help message and return

Notes. Features can also be specified like in elements of a canonical file name. For details on urboot bootloaders and their features see <https://github.com/stefanrueger/urboot>.

*0 Some parts do not provide 4 s or 8 s watchdog timeout

*1 `autobaud` only works with UART I/O where the RX port pin is bit-addressable

- *2 From classic parts, only ATtiny441/841 have alternate UART signals
- *3 There is a subtle difference between external oscillators (**x**), which are reasonably accurate, and internal oscillators (**a...h**, **i**, **j...q**), which tend to be inaccurate for classic parts. The former can afford higher baud rate errors up to 2.2% while for the latter AVRDUDE warns at the lower threshold of 0.7% baud rate error. Classic UARTs have integer baud rate divisors, which can lead to high baud rate quantisation errors. Used with the feature **best** AVRDUDE can automatically replace UART I/O code with **swio** I/O code when the former exhibits too high baud rate errors. For that, AVRDUDE better knows the type of oscillator (**x** or **i**) that drives a board. For lowest baud rate errors on an individual board that runs on internal oscillators it is best to measure **F_CPU** and use the measured value with the **i** prefix. Alternatively, one can use the nominal internal **F_CPU** (say 8 MHz) and use prefix letters that make the bootloader work: depending on the letter AVRDUDE subtracts from or adds to the nominal **F_CPU** multiples of 1.25%.
- *4 Dual boot is not supported for parts that lack standard SPI communication
- *5 Not all parts provide hardware bootloader support; **hw** renders **pr** meaningless
- *6 Reset vector protection is only available if flash size is a power of 2 (not ATmega406)
- *7 **u1...u3** is only advisory, ie, can result in any of **u1...u4**
- *8 **u1...u4** is not available for parts with the 4-page-erase quirk
- *9 The file is still written to disk in connection with **configs**, **show** or **list**. Bootloader files are per default written as Intel Hex format with comments. A filename of **-** writes the file to **stdout**. If a bootloader is a vector bootloader then the bootloader file contains a segment for initialising the reset vector to point to the bootloader. Tag files can be used in connection with the terminal **disasm** command.

Baud rate quantification errors are displayed with **-v**. A 0.79% baud rate error is considered OK for external oscillators but too high for internal oscillators. Hence, AVRDUDE selects software I/O when neither **uart** nor **swio** are explicitly requested:

```
$ avrdude -qv -c dryrun -p m328p -U urboot:x8mhz_56kbaud |& grep -i baud.rate
Baud rate error -0.79% for external oscillator OK

$ avrdude -qv -c dryrun -p m328p -U urboot:i8mhz_56kbaud |& grep -i baud.rate
Switching to SWIO as baud rate error -0.79% too high for internal oscillator
Baud rate error -0.10% for internal oscillator OK
```

There is a warning when the user requests hardware UART I/O:

```
$ avrdude -qvc dryrun -p m328p -U urboot:i8mhz_56kbaud_uart0 |& grep -i baud.rate
Warning: high baud rate error -0.79% for int oscillator: consider switching to swio
```

Requesting **best** makes AVRDUDE switch to **swio** when UART quantisation errors are considered too high:

```
$ avrdude -qvc dryrun -p m328p -U urboot:i8mhz_56kbaud_uart0_best |& grep -i baud.rate
Switching to SWIO as baud rate error -0.79% too high for internal oscillator
Baud rate error -0.10% for internal oscillator OK
```

Alternatively, the user can request `swio` on those `rx/tx` lines that the UART uses:

```
$ avrdude -qvc dryrun -p m328p -U urboot:i8mhz_56kbaud_uart0_swio |& grep -i baud.rate
Baud rate error -0.10% for internal oscillator OK
```

Tag files can be used in connection with `disasm`:

```
$ avrdude -qqc dryrun -p m328p -U urboot:m328p_autobaud_ee_tags=/tmp/x \
-T "disasm -gt=/tmp/x flash 0 -1"

[...]
.equ    io.mcusr,    0x34
.equ    io.spmcsr,   0x37
.equ    mem.wdtcsr,  0x60
[...]
.text
main:
__vector_reset:
L0000: rjmp    .-386                ; L7e80 (urboot)

urmarker:
L0002: .byte   0x75, 0x72          ; ur
; ur

L0004: .fill   16190, 2, 0xffff

; Rjmp from L0000
urboot:
L7e80: clr     r1                  ; Entry point for bootloader
L7e82: in      r2, io.mcusr
L7e84: out     io.mcusr, r1
L7e86: ldi     r24, 0x00           ; 0
L7e88: rcall   set_watchdog       ; L7f5c
L7e8a: sbrs    r2, 1              ; Bit 1 = 0x02
application:
L7e8c: rjmp    .+470              ; L0064 (__vector_spm_ready)
serial_boot:
L7e8e: ldi     r24, 0x0e          ; 14
L7e90: rcall   set_watchdog       ; L7f5c

[...]

Label127:
L7fea: clr     r1
L7fec: ret

unused:
L7fee: .fill   6, 2, 0xffff       ; 12 bytes for _serialno= and/or _fill=

npages_vecnum:
L7ffa: .byte   0x03, 0x19        ; Usage of 3 pages; vector/spmready bootloader
; --

L7ffc: rjmp    pgm_write_page     ; L7f78

features_version:
L7ffe: .byte   0xe6, 0x40        ; Encodes u8.0 weU-jPra-
; _
```

It is noteworthy that `-c dryboot` allows programming one urboot bootloader and, from that point onwards, emulates `-c urclock -x no metadata`. This means, eg, `-c dryboot` will not allow the bootloader be overwritten:

```
$ avrdude -q -c dryboot -p m328p -U urboot:autobaud_ee -U blink.hex \
-T "write flash 0x7f80 0x00"

Processing -U flash:w:urboot:autobaud_ee:i
Reading 376 bytes for flash from input file urboot:autobaud_ee
Writing 376 bytes to flash, 376 bytes written, 376 verified
Setting fuses for bootloader urboot:autobaud_ee
Detected urboot bootloader u8.0 weU-jPra- in [0x7e80, 0x7fff] with vector=25

Processing -U flash:w:blink.hex:i
Reading 354 bytes for flash from input file blink.hex
Writing 354 bytes to flash, 354 bytes written, 354 verified

Processing -T write flash 0x7f80 0x00
Warning: (write) programmer write protects flash address 0x7f80

Avrdude done. Thank you.
```

More importantly, `-c dryboot` patches the vector table of any to-be-programmed sketch if the previously installed bootloader is an urboot vector bootloader just like `avrdude -c urclock -x no metadata` would; this is needed for the vector bootloader to work (for details, see the <https://github.com/stefanrueger/urboot> project):

```
$ avrdude -qq -c dryboot -p m328p -U urboot:autobaud_ee -U blink.hex \
-T "disasm flash 0 2" -T "disasm flash 100 4"

L0000: 3f cf      rjmp      .-386                ; L7e80
L0064: 0c 94 34 00 jmp      0x0068
```

In contrast, using `-c dryrun` does not patch the vector table of a sketch as it emulates and behaves like an external programmer:

```
$ avrdude -qq -c dryrun -p m328p -U urboot:autobaud_ee -U blink.hex \
-T "disasm flash 0 2" -T "disasm flash 100 2"

L0000: 33 c0      rjmp      .+102                ; L0068
L0064: 11 c0      rjmp      .+34                 ; L0088
```

The following example prepares a patched sketch for programming through a self-written downloader/uploader or through urboot's dual boot. Note that 385 is the size of the bootloader plus 1 and that the generated file `blink-patched.hex` has the size of flash minus 384. In absence of the option `-A` AVRDUDE drops all trailing `0xff` from the patched sketch.

```
$ avrdude -qq -c dryboot -p m328p -U urboot:autobaud_ee -U blink.hex \
-T "save flash 0 -385 blink-patched.hex:I"
```

6 Programmer-Specific Information

6.1 Atmel STK600

The following devices are supported by the respective STK600 routing and socket card:

Routing card	Socket card	Devices
	STK600-ATTINY10	t4 t5 t9 t10
STK600-RC008T-2	STK600-DIP	t11 t12 t13 t13a t25 t45 t85
STK600-RC008T-7	STK600-DIP	t15
STK600-RC014T-42	STK600-SOIC	t20
STK600-RC020T-1	STK600-DIP	t2313 t2313a t4313
	STK600-TinyX3U	t43u
STK600-RC014T-12	STK600-DIP	t24 t44 t84 t24a t44a
STK600-RC020T-8	STK600-DIP	t26 t261 t261a t461 t861 t861a
STK600-RC020T-43	STK600-SOIC	t261 t261a t461 t461a t861 t861a
STK600-RC020T-23	STK600-SOIC	t87 t167
STK600-RC028T-3	STK600-DIP	t28
STK600-RC028M-6	STK600-DIP	t48 t88 m8 m8a m48 m88 m168 m48p m48pa m88p m88pa m168p m168pa m328p
	QT600-ATTINY88-QT8	t88
STK600-RC040M-4	STK600-DIP	m8515 m162
STK600-RC044M-30	STK600-TQFP44	m8515 m162
STK600-RC040M-5	STK600-DIP	m8535 m16 m16a m32 m32a m164p m164pa m324p m324pa m644 m644p m644pa m1284p
STK600-RC044M-31	STK600-TQFP44	m8535 m16 m16a m32 m32a m164p m164pa m324p m324pa m644 m644p m644pa m1284p
	QT600-ATMEGA324-QM64	m324pa
STK600-RC032M-29	STK600-TQFP32	m8 m8a m48 m88 m168 m48p m48pa m88p m88pa m168p m168pa m328p
STK600-RC064M-9	STK600-TQFP64	m64 m64a m128 m128a m1281 m2561 c32 c64 c128
STK600-RC064M-10	STK600-TQFP64	m165 m165p m169 m169p m169pa m325 m325p m329 m329p m645 m649 m649p
STK600-RC100M-11	STK600-TQFP100	m640 m1280 m2560
	STK600-ATMEGA2560	m2560
STK600-RC100M-18	STK600-TQFP100	m3250 m3250p m3290 m3290p m6450 m6490
STK600-RC032U-20	STK600-TQFP32	usb82 usb162 m8u2 m16u2 m32u2
STK600-RC044U-25	STK600-TQFP44	m16u4 m32u4
STK600-RC064U-17	STK600-TQFP64	m32u6 usb646 usb1286 usb647 usb1287
STK600-RCPWM-22	STK600-TQFP32	m32c1 m64c1 m16m1 m32m1 m64m1

STK600-RCPWM-19	STK600-S0IC	pwm2 pwm3 pwm2b pwm3b pwm216 pwm316
STK600-RCPWM-26	STK600-S0IC	pwm81
STK600-RC044M-24	STK600-TSSOP44	m16hvb m32hvb
	STK600-HVE2	m64hve
	STK600-ATMEGA128RFA1	m128rfa1
STK600-RC100X-13	STK600-TQFP100	x64a1 x128a1 x128A1revd x128a1u
	STK600-ATXMEGA1281A1	x128a1
	QT600-ATXMEGA128A1- QT16	x128a1
STK600-RC064X-14	STK600-TQFP64	x64a3 x128a3 x256a3 x64d3 x128d3 x192d3 x256d3
STK600-RC064X-14	STK600-MLF64	x256a3b
STK600-RC044X-15	STK600-TQFP44	x32a4 x16a4 x16d4 x32d4
	STK600-ATXMEGATO	x32t0
	STK600-uC3-144	uc3a0512 uc3a0256b uc3a0128b
STK600-RCUC3A144-33	STK600-TQFP144	uc3a0512 uc3a0256b uc3a0128b
STK600-RCuC3A100-28	STK600-TQFP100	uc3a1512b uc3a1256b uc3a1128b
STK600-RCuC3B0-21	STK600-TQFP64-2	uc3b0256b uc3b0512revcb uc3b0512b uc3b0128b uc3b064b uc3d1128b
STK600-RCuC3B48-27	STK600-TQFP48	uc3b1256b uc3b164b
STK600-RCUC3A144-32	STK600-TQFP144	uc3a3512b uc3a3256b uc3a3128b uc3a364b uc3a3256sb uc3a3128sb uc3a364sb
STK600-RCUC3C0-36	STK600-TQFP144	uc3c0512b uc3c0256b uc3c0128b uc3c064b
STK600-RCUC3C1-38	STK600-TQFP100	uc3c1512b uc3c1256b uc3c1128b uc3c164b
STK600-RCUC3C2-40	STK600-TQFP64-2	uc3c2512b uc3c2256b uc3c2128b uc3c264b
STK600-RCUC3C0-37	STK600-TQFP144	uc3c0512b uc3c0256b uc3c0128b uc3c064b
STK600-RCUC3C1-39	STK600-TQFP100	uc3c1512b uc3c1256b uc3c1128b uc3c164b
STK600-RCUC3C2-41	STK600-TQFP64-2	uc3c2512b uc3c2256b uc3c2128b uc3c264b
STK600-RCUC3L0-34	STK600-TQFP48	uc3l064b uc3l032b uc3l016b
	QT600-AT32UC3L-QM64	uc3l064b

Ensure the correct socket and routing card are mounted *before* powering on the STK600. While the STK600 firmware ensures the socket and routing card mounted match each other (using a table stored internally in nonvolatile memory), it cannot handle the case where a wrong routing card is used, e. g. the routing card **STK600-RC040M-5** (which is meant for 40-pin DIP AVR that have an ADC, with the power supply pins in the center of the package) was used but an ATmega8515 inserted (which uses the “industry standard” pinout with Vcc and GND at opposite corners).

Note that for devices that use the routing card STK600-RC008T-2, in order to use ISP mode, the jumper for AREFO must be removed as it would otherwise block one of the ISP signals. High-voltage serial programming can be used even with that jumper installed.

The ISP system of the STK600 contains a detection against shortcuts and other wiring errors. AVRDUDE initiates a connection check before trying to enter ISP programming mode, and display the result if the target is not found ready to be ISP programmed.

High-voltage programming requires the target voltage to be set to at least 4.5 V in order to work. This can be done using *Terminal Mode*, see Chapter 3 [Terminal Mode Operation], page 36.

6.2 DFU Bootloader Using FLIP Version 1

Bootloaders using the FLIP protocol version 1 experience some very specific behaviour.

These bootloaders have no option to access memory areas other than Flash and EEPROM.

When the bootloader is started, it enters a *security mode* where the only acceptable access is to query the device configuration parameters (which are used for the signature on AVR devices). The only way to leave this mode is a *chip erase*. As a chip erase is normally implied by the -U option when reprogramming the flash, this peculiarity might not be very obvious immediately.

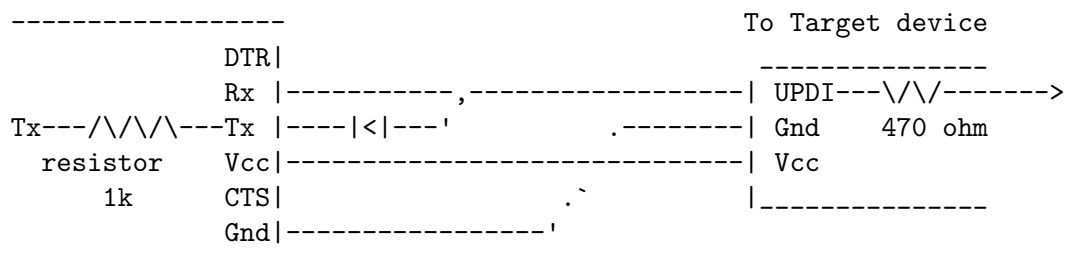
Sometimes, a bootloader with security mode already disabled seems to no longer respond with sensible configuration data, but only 0xFF for all queries. As these queries are used to obtain the equivalent of a signature, AVRDUDE can only continue in that situation by forcing the signature check to be overridden with the -F option.

A *chip erase* might leave the EEPROM unerased, at least on some versions of the bootloader.

6.3 SerialUPDI Programmer

SerialUPDI programmer can be used for programming UPDI-only devices using very simple serial connection. You can read more about the details here <https://github.com/SpenceKonde/AVR-Guidance/blob/master/UPDI/jtag2updi.md>

SerialUPDI programmer has been tested using FT232RL USB->UART interface with the following connection layout (copied from Spence Kohde's page linked above):



There are several limitations in current SerialUPDI/AVRDUDE integration, listed below.

Currently available devices support only UPDI NVM programming model 0, 2 3 and 5, but there is also experimental implementation of model 4 - it has been tested only on a

single device, so issues with other devices are expected. Full NVM v4 mode support will be provided once the hardware is widely available.

One of the core AVRDUDE features is verification of the connection by reading device signature prior to any operation, but this operation is not possible on UPDI locked devices. Therefore, to be able to connect to such a device, you have to provide `-F` to override this check.

Please note: using `-F` during write operation to locked device will force chip erase. Use carefully.

Another issue you might notice is slow performance of EEPROM writing using SerialUPDI for AVR Dx devices. This can be addressed by changing *avrdude.conf* section for this device - changing EEPROM page size to 0x20 (instead of default 1), like so:

```
#-----
# AVR128DB28
#-----

part parent      ".avr dx"
  id              = "avr128db28";
  desc            = "AVR128DB28";
  signature       = 0x1E 0x97 0x0E;

  memory "flash"
    size         = 0x20000;
    offset        = 0x800000;
    page_size     = 0x200;
    readsize      = 0x100;
  ;

  memory "eeprom"
    size          = 0x200;
    offset         = 0x1400;
    page_size      = 0x20;
    readsize       = 0x100;
  ;
;
```

The point of USERROW is to provide ability to write configuration details to already locked device and currently SerialUPDI interface supports this feature. Please note: on locked devices it's not possible to read back USERROW contents when written, so the automatic verification will most likely fail and to prevent error messages, use `-V`.

In case you run into issues with the SerialUPDI interface, please make sure to run the intended command with debug output enabled (`-v -v -v`) and provide this verbose output with your bug report. You can also try to perform the same action using *pymcuprog* (<https://github.com/microchip-pic-avr-tools/pymcuprog>) utility with `-v debug` and provide its output too. You will notice that both outputs are pretty similar, and this was implemented like that on purpose - it was supposed to make analysis of UPDI protocol quirks easier.

6.4 Programmer LED Management

Some hardware programmers have LEDs, and the firmware controls them fully without AVRDUDE having a way to influence their LED states. Other programmers have LEDs and expect the host downloader/uploader to handle them, for example bit-banging programmers, ftdi-based programmers or linuxgpio programmers. For those programmers AVRDUDE provides support of four LEDs (RDY, ERR, PGM and VFY) which can be set via corresponding subroutines in the code for the respective `-c` programmer.

The RDY LED is set once the programmer is initialised and switched off when AVRDUDE exits. During reading, writing or erasing the target the PGM LED flashes with around 2.5 Hz, whilst the VFY LED comes on during -U verification of the written contents. Errors are indicated with the ERR LED.

Assuming AVRDUDE got to the point where LEDs are accessible and the RDY LED was switched on then, on exit, AVRDUDE will leave the LEDs in the following states:

PGM	VFY	ERR	Semantics
off	off	off	OK: all tasks done without errors
off	off	on	Some error not related to read, write or erase
on	off	on	Read, write or erase error
off	on	on	Verification error but no read, write or erase error
on	on	on	Verification error and read, write or erase error

Other combinations should not show after exit.

Appendix A Platform Dependent Information

A.1 Unix

A.1.1 Unix Installation

Refer to <https://github.com/avrdudes/avrdude/wiki> for the latest installation tips.

A.1.2 Unix Configuration Files

When AVRDUDE is built using the default `--prefix` configure option, the default configuration file for a Unix system is located at `/usr/local/etc/avrdude.conf`. This can be overridden by using the `-C` command line option. Additionally, the user's home directory is searched for a file named `.avrduderc`, and if found, is used to augment the system default configuration file.

A.1.2.1 FreeBSD Configuration Files

When AVRDUDE is installed using the FreeBSD ports system, the system configuration file is always `/usr/local/etc/avrdude.conf`.

A.1.2.2 Linux Configuration Files

When AVRDUDE is installed using from an rpm package, the system configuration file will be always be `/etc/avrdude.conf`.

A.1.3 Unix Port Names

The parallel and serial port device file names are system specific. macOS has no default serial port names, but available ports can be found under `/dev/cu.*`. Please take note AVRDUDE does not support parallel port programming under macOS.

The following table lists the default names for a given system.

System	Default Parallel Port	Default Serial Port
FreeBSD	<code>/dev/ppi0</code>	<code>/dev/cuad0</code>
Linux	<code>/dev/parport0</code>	<code>/dev/ttyS0</code>
Solaris	<code>/dev/printers/0</code>	<code>/dev/term/a</code>

On FreeBSD systems, AVRDUDE uses the `ppi(4)` interface for accessing the parallel port and the `sio(4)` driver for serial port access.

On Linux systems, AVRDUDE uses the `ppdev` interface for accessing the parallel port and the `tty` driver for serial port access.

On Solaris systems, AVRDUDE uses the `ecpp(7D)` driver for accessing the parallel port and the `asy(7D)` driver for serial port access.

A.1.4 Unix USB Permissions

In most cases the kernel driver initializes a plug-and-play device to be owned by user `root` and group `root` with only `r/w` permission for the user `root` rendering the device inaccessible to regular users. Whilst users can run AVRDUDE sessions as `root` this is definitely *not good practice*. Giving USB plug-and-play devices the correct permissions is much better. USB

AVR programmers are normally identified by a two-byte hexadecimal vendor ID and a two-byte hexadecimal product id. Both are typically used to identify the device that needs new permissions.

A.1.4.1 FreeBSD USB Permissions

In FreeBSD a so-called `devd` config files in `/usr/local/etc/devd` serve to modify permissions of plugged-in USB devices. Here is an example how Atmel's JTAGICE3 programmer (product ID 0x2110 or 0x2140) by Atmel (vendor ID 0x03eb) can be given appropriate permissions using a file `jtagice3.conf`:

```
notify 100 {
    match "system" "USB";
    match "subsystem" "DEVICE";
    match "type" "ATTACH";
    match "vendor" "0x03eb";
    match "product" "(0x2110|0x2140)";
    action "chmod 660 /dev/$cdev";
    action "chgrp yourgroup /dev/$cdev";
};
```

`yourgroup` would be a group that the user(s) should be member of who wish to have access to the programmer.

A.1.4.2 Linux USB Permissions

Linux has a special userspace `/dev` device manager called `udev` that deals with, amongst other things, plug-and-play USB devices. It is recommended to specify so-called `udev` rules to define access permissions for these devices instead. These rules typically reside in a file with the name `nn-descriptive-name.rules` in the directory `/etc/udev/rules.d`. Here, `nn` is a two-digit number that determines the lexical order in which the `udev` rule files are processed. Rules processed later can overwrite earlier rules, but it not recommended to put user-generated rules higher than 60, as some of the actions they require are processed by higher-level system rules.

Here a typical `udev` rule for allowing an ordinary user access to the plugged-in AVRISP mkII programmer (product ID 0x2104) by Atmel (vendor ID 0x03eb):

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2104", \
MODE="0660", TAG+="uaccess"
```

This furnishes the corresponding device node with `0660` access permissions: this means `r/w` for the user `root` and any user belonging to the group of the device, which the device driver might assign to a different group than the default `root`. The key of the rule is the attached `TAG` named `uaccess`, which has the effect that the login daemon applies a dynamic user access control list to the device node making the device usable for the currently logged-in user. When used in anger, `udev` rules must appear on one line; above example was broken into two lines so it fits into the example box.

AVRDUDE's developer option `-c programmer/u` will show above suggested `udev` rule for the named programmer. Wildcards are allowed:

```

$ avrdude -c jtag\*/u

1. Examine the suggested udev rules below; to install run:

avrdude -c "jtag*/u" | tail -n +11 | sudo tee /etc/udev/rules.d/55-avrdude-jtagX.rules
sudo chmod 0644 /etc/udev/rules.d/55-avrdude-jtagX.rules

2. Unplug any AVRDUDE USB programmers and plug them in again
3. Enjoy user access to the USB programmer(s)

Note: To install all udev rules known to AVRDUDE follow: avrdude -c "*/u" | more

# Generated from avrdude -c "jtag*/u"

ACTION!="add|change", GOTO="avrdude_end"

# jtag2dw, jtag2fast, jtag2, jtag2isp, jtag2pdi, jtag2slow, jtagmkII, jtag2avr32
SUBSYSTEM=="usb", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2103", \
    MODE="0660", TAG+="uaccess"

# jtag3, jtag3dw, jtag3isp, jtag3pdi, jtag3updi
SUBSYSTEM=="usb", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2110", \
    MODE="0660", TAG+="uaccess"
KERNEL=="hidraw*", SUBSYSTEM=="hidraw", ATTRS{idVendor}=="03eb", \
    ATTRS{idProduct}=="2110", MODE="0660", TAG+="uaccess"
SUBSYSTEM=="usb", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2140", \
    MODE="0660", TAG+="uaccess"
KERNEL=="hidraw*", SUBSYSTEM=="hidraw", ATTRS{idVendor}=="03eb", \
    ATTRS{idProduct}=="2140", MODE="0660", TAG+="uaccess"

# jtagkey
SUBSYSTEM=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="cff8", \
    MODE="0660", TAG+="uaccess"

```

Again, each rule must be written as one line: breaking up rules into two lines was only done to fit AVRDUDE's output to the boxed display. USB devices in HID mode require a second rule dealing with the `hidraw` subsystem as seen above.

A.1.5 Unix Documentation

AVRDUDE installs a manual page as well as info, HTML and PDF documentation. The manual page is installed in `/usr/local/man/man1` area, while the HTML and PDF documentation is installed in `/usr/local/share/doc/avrdude` directory. The info manual is installed in `/usr/local/info/avrdude.info`.

Note that these locations can be altered by various configure options such as `--prefix`.

A.2 Windows

A.2.1 Installation

Refer to <https://github.com/avrdudes/avrdude/wiki> for the latest installation tips.

A.2.2 Windows Configuration Files

A.2.2.1 Windows Configuration File Names

AVRDUDE on Windows looks for a system configuration file name of `avrdude.conf` and looks for a user override configuration file of `avrdude.rc` in the same directory where `avrdude.exe` is located.

A.2.2.2 Windows Configuration File Location

AVRDUDE on Windows has a different way of searching for the system and user configuration files. Below is the search method for locating the configuration files:

1. Only for the system configuration file: `<directory from which application loaded>/../etc/avrdude.conf`
2. The directory from which the application loaded.
3. The current directory.
4. The Windows system directory. On Windows NT, the name of this directory is `SYSTEM32`.
5. The Windows directory.
6. The directories that are listed in the `PATH` environment variable.

A.2.3 Windows Port Names

A.2.3.1 Windows Serial Ports

When you select a serial port (i.e. when using an STK500) use the Windows serial port device names such as: `COM1`, `COM2`, etc. If the board in question uses an USB-serial adapter, or is connected via one, there are several ways to identify the correct port name:

- In the Windows Device Manager under Ports, then COM & LPT
- Enquiring `avrdude -P "?s"` if compiled with `libserialport`
- Use the serial adapter name instead of `COMx`, e.g., `-P ft232r`

A.2.3.2 Windows Parallel Ports

AVRDUDE does not support parallel port programming in Windows. If you need to run AVRDUDE using a programmer on a parallel port, you might want to try one of the BSDs or Linux.

Appendix B Troubleshooting

Please report any bugs encountered via <https://github.com/avrdudes/avrdude/issues>.

AVRDUDE's wiki <https://github.com/avrdudes/avrdude/wiki> is a great place to learn about installing AVRDUDE on various platforms and, generally, to learn a few tricks of the trade. In particular, the FAQ (<https://github.com/avrdudes/avrdude/wiki/FAQ>) and the known limitations (<https://github.com/avrdudes/avrdude/wiki/Known-limitations-of-avrdude>) of avrdude are worth reading.

Here are a few examples for things that can go wrong and what to do:

- Problem: I'm using a serial programmer under Windows and get the following error:
`avrdude: serial_open(): can't set attributes for device "com1",`
 Solution: This problem seems to appear with certain versions of Cygwin. Specifying `"/dev/com1"` instead of `"com1"` should help.
- Problem: I'm using Linux and my AVR910 programmer is really slow.
 Solution: There are two problems here. First, the system may wait some time before it passes data from the serial port to the program. Under Linux the following command works around this (you may need root privileges for this).
`setserial port low_latency`
 Secondly, the serial interface chip may delay the interrupt for some time. This behaviour can be changed by setting the FIFO-threshold to one. Under Linux this can only be done by changing the kernel source in `drivers/char/serial.c`. Search the file for `UART_FCR_TRIGGER_8` and replace it with `UART_FCR_TRIGGER_1`. Note that overall performance might suffer if there is high throughput on serial lines. Also note that you are modifying the kernel at your own risk.
- Problem: I'm not using Linux and my AVR910 programmer is really slow.
 Solution: The reasons for this are the same as above. If you know how to work around this on your OS, please let us know.
- Problem: Page-mode programming the EEPROM using the `-U` option does not erase EEPROM cells before writing, and thus cannot necessarily overwrite non-0xff values.
 Solution: This is an inherent feature of how JTAG EEPROM programming works, and is documented as such in the datasheets. In order to successfully program the EEPROM, a prior chip erase with the EESAVE fuse unprogrammed is required. This also applies to the STK500 and STK600 in high-voltage programming mode.
 The terminal, however, recognises that the programmer struggles to write to EEPROM. It then reads flash, EEPROM and, if present, bootrow contents, performs a chip erase and then writes these memories back. This happens when flushing the cache or leaving the terminal and takes some time. EESAVE needs to be unprogrammed for this.
- Problem: How do I turn off the DWEN fuse?
 Solution: If the DWEN (debugWIRE enable) fuse is activated, the `/RESET` pin is not functional anymore, so normal ISP communication cannot be established. There are two options to deactivate that fuse again: high-voltage programming, or getting the JTAG ICE mkII talk debugWIRE, and prepare the target AVR to accept normal ISP communication again.

The first option requires a programmer that is capable of high-voltage programming (either serial or parallel, depending on the AVR device), for example the STK500. In high-voltage programming mode, the `/RESET` pin is activated initially using a 12 V pulse (thus the name *high voltage*), so the target AVR can subsequently be reprogrammed, and the `DWEN` fuse can be cleared. Typically, this operation cannot be performed while the AVR is located in the target circuit though.

The second option requires a JTAG ICE mkII that can talk the debugWIRE protocol. The ICE needs to be connected to the target using the JTAG-to-ISP adapter, so the JTAG ICE mkII can be used as a debugWIRE initiator as well as an ISP programmer. AVRDUDE will then be activated using the `jtag2isp` programmer type. The initial ISP communication attempt will fail, but AVRDUDE then tries to initiate a debugWIRE reset. When successful, this will leave the target AVR in a state where it can accept standard ISP communication. The ICE is then signed off (which will make it signing off from the USB as well), so AVRDUDE has to be called again afterwards. This time, standard ISP communication can work, so the `DWEN` fuse can be cleared. The pin mapping for the JTAG-to-ISP adapter is:

JTAG	ISP
1	3
2	6
3	1
4	2
6	5
9	4

- Problem: Differentiate multiple USBtinyISP (or USBasp) programmers

Solution: The `-c usbtiny` programmer distinguishes multiple physical USBtinyISP devices based on their `busdir:devicefile` pairs that describe their place in the USB hierarchy on a specific host. This pair can be specified in the `-P usb:busdir:devicefile` option. The naming convention for the bus and device depends on the operating system. Examples for Linux, FreeBSD and Windows, respectively:

```
$ avrdude -c usbtiny -p atmega8 -P usb:003:025
$ avrdude -c usbtiny -p atmega8 -P usb:/dev/usb:/dev/ugen1.3
$ avrdude -c usbtiny -p atmega8 \
  -P 'usb:bus-0:\\?\\usb#vid_16c0&pid_05dc#0001#{a5...ed}--WinUSB'
```

The Windows device name contains the backslash file separator, so the `-P` option will need appropriate quoting on the command line, eg, in bash with single quotes.

For USBasp the same `-P usb:busdir:devicefile` as with USBtiny selects the right device. Alternatively, USBasp can select the device via its serial number using `-P usb:serialno`.

Note that `avrdude -v -P usb:xyz` will print out suitable programmers on the bus assuming `xyz` does not match any device.

```
$ avrdude -v -Pusb:xyz -c usbasp -p m328p 2>&1 | grep ^Found
Found USBasp with busdir:devicefile = 001:008, serial_number = 0001
Found USBasp with busdir:devicefile = 001:009, serial_number = 1234
```

```
$ avrdude -qq -c USBasp -p atmega8 -P usb:34
```

- Problem: I cannot do . . . when the target is in debugWIRE mode.

Solution: debugWIRE mode imposes several limitations.

The debugWIRE protocol is Atmel's proprietary one-wire (plus ground) protocol to allow an in-circuit emulation of the smaller AVR devices, using the /RESET line. DebugWIRE mode is initiated by activating the DWEN fuse, and then power-cycling the target. While this mode is mainly intended for debugging/emulation, it also offers limited programming capabilities. Effectively, the only memory areas that can be read or programmed in this mode are flash and EEPROM. It is also possible to read out the signature. All other memory areas cannot be accessed. There is no *chip erase* functionality in debugWIRE mode; instead, while reprogramming the flash, each flash page is erased right before updating it. This is done transparently by the JTAG ICE mkII (or AVR Dragon). The only way back from debugWIRE mode is to initiate a special sequence of commands to the JTAG ICE mkII (or AVR Dragon), so the debugWIRE mode will be temporarily disabled, and the target can be accessed using normal ISP programming. This sequence is automatically initiated by using the JTAG ICE mkII or AVR Dragon in ISP mode, when they detect that ISP mode cannot be entered.

- Problem: I want to use my JTAG ICE mkII to program an Xmega device through PDI. The documentation tells me to use the *XMEGA PDI adapter for JTAGICE mkII* that is supposed to ship with the kit, yet I don't have it.

Solution: Use the following pin mapping:

JTAG ICE mkII probe	Target pins	Squid cable colors	PDI header
1 (TCK)		Black	
2 (GND)	GND	White	6
3 (TDO)		Grey	
4 (VTref)	VTref	Purple	2
5 (TMS)		Blue	
6 (nSRST)	PDI_CLK	Green	5
7 (N.C.)		Yellow	
8 (nTRST)		Orange	
9 (TDI)	PDI_DATA	Red	1
10 (GND)		Brown	

- Problem: I want to use my AVR Dragon to program an Xmega device through PDI.

Solution: Use the 6 pin ISP header on the Dragon and the following pin mapping:

Dragon ISP Header	Target pins
1 (SDI)	PDI_DATA
2 (VCC)	VCC
3 (SCK)	
4 (SDO)	
5 (RESET)	PDI_CLK / RST
6 (GND)	GND

- Problem: I want to use my AVRISP mkII to program an ATtiny4/5/9/10 device through TPI. How to connect the pins?

Solution: Use the following pin mapping:

AVRISP connector	Target pins	ATtiny pin #
1 (SDI)	TPIDATA	1
2 (VTref)	Vcc	5
3 (SCK)	TPICLK	3
4 (SDO)		
5 (RESET)	/RESET	6
6 (GND)	GND	2

- Problem: I want to program an ATtiny4/5/9/10 device using a serial/parallel bitbang programmer. How to connect the pins?

Solution: Since TPI has only 1 pin for bi-directional data transfer, both SDI and SDO pins should be connected to the TPIDATA pin on the ATtiny device. However, a 1K resistor should be placed between the SDO and TPIDATA. The SDI pin connects to TPIDATA directly. The SCK pin is connected to TPICLK.

In addition, the Vcc, /RESET and GND pins should be connected to their respective ports on the ATtiny device.

- Problem: How can I use a FTDI FT232R USB-to-Serial device for bitbang programming?

Solution: When connecting the FT232 directly to the pins of the target Atmel device, the polarity of the pins defined in the `programmer` definition should be inverted by prefixing a tilde. For example, the `dasa` programmer would look like this when connected via a FT232R device (notice the tildes in front of pins 7, 4, 3 and 8):

```
programmer
    id      = "dasa_ftdi";
    desc    = "serial port banging, reset=rts sck=dtr sdo=txd sdi=cts";
    type    = serbb;
    reset   = ~7;
    sck     = ~4;
    sdo     = ~3;
    sdi     = ~8;
;
```

Note that this uses the FT232 device as a normal serial port, not using the FTDI drivers in the special bitbang mode.

- Problem: My ATtiny4/5/9/10 reads out fine, but any attempt to program it (through TPI) fails. Instead, the memory retains the old contents.

Solution: Mind the limited programming supply voltage range of these devices.

In-circuit programming through TPI is only guaranteed by the datasheet at Vcc = 5 V.

- Problem: My ATxmega...A1/A2/A3 cannot be programmed through PDI with my AVR Dragon. Programming through a JTAG ICE mkII works though, as does programming through JTAG.

Solution: None (may be a firmware issue of the AVR Dragon).

It is said that the AVR Dragon can only program devices from the A4 Xmega sub-family.

- Problem: after flashing a firmware that reduces the target's clock speed (e.g. through the CLKPR register), further ISP connection attempts fail. Or a programmer cannot initialize communication with a brand new chip.

Solution: Even though ISP starts with pulling /RESET low, the target continues to run at the internal clock speed either as defined by the firmware running before or as set by the factory. Therefore, the ISP clock speed must be reduced appropriately (to less than 1/4 of the internal clock speed) using the -B option before the ISP initialization sequence will succeed.

As that slows down the entire subsequent ISP session, it might make sense to just issue a *chip erase* using the slow ISP clock (option -e), and then start a new session at higher speed. Option -D might be used there, to prevent another unneeded erase cycle.

Appendix C List of Programmers

AVRDUDE supports the programmers below: the left column lists the programmer's id as used for `-c`, whilst the right column contains a short description and the list of available programming interfaces in brackets; see Section 4.2 [Programmer Definitions], page 54. There is more detail about each programmer in the AVRDUDE configuration file.

2232hio	2232hio based on FT2232H with buffer and LEDs (TPI, ISP)
4232h	FT4232H programmer (TPI, ISP)
89isp	Atmel at89isp cable (TPI, ISP)
abcmini	ABCmini Board, aka Dick Smith HOTCHIP (TPI, ISP)
adafruit_gemma	Trinket Gemma bootloader disguised as USBtiny (SPM)
alf	Nightshade ALF-PgmAVR via PC parallel port (TPI, ISP)
arduino	Arduino bootloader using STK500 v1 protocol (SPM)
arduino-ft232r	Arduino: FT232R connected to ISP (TPI, ISP)
diecimila	Arduino: FT232R connected to ISP (TPI, ISP)
arduino_as_isp	AVR as programmer with Arduino-as-ISP FW (ISP)
arduino_gemma	Arduino Gemma bootloader disguised as USBtiny (SPM)
arduinoisp	Arduino-branded USBtiny ISP Programmer (TPI, ISP)
arduinoisporg	Arduino-branded USBtiny ISP Programmer (TPI, ISP)
atisp	AT-ISP v1.1 programming cable for AVR-SDK1 (TPI, ISP)
atmelice	Atmel-ICE (JTAG, XMEGAJTAG, AVR32JTAG)
atmelice_jtag	Atmel-ICE (JTAG, XMEGAJTAG, AVR32JTAG)
atmelice_dw	Atmel-ICE (debugWIRE)
atmelice_isp	Atmel-ICE (ISP)
atmelice_pdi	Atmel-ICE (PDI)
atmelice_tpi	Atmel-ICE (TPI)
atmelice_updi	Atmel-ICE (UPDI)
avr109	Atmel bootloader (AVR109, AVR911) (SPM)
avr911	Atmel bootloader (AVR109, AVR911) (SPM)
avr910	Atmel Low Cost Serial Programmer (ISP)
avrftdi	FT2232H/D programmer (TPI, ISP)
2232h	FT2232H/D programmer (TPI, ISP)
avrisp	Serial Atmel AVR ISP using STK500 (ISP)
avrisp-u	Kanda AVRISP-U (TPI, ISP)
avrispmkII	USB Atmel AVR ISP mkII (TPI, ISP, PDI)
avrisp2	USB Atmel AVR ISP mkII (TPI, ISP, PDI)
avrispv2	Serial Atmel AVR ISP (TPI, ISP)
bascom	Bascom SAMPLE programming cable (TPI, ISP)
blaster	Altera ByteBlaster (TPI, ISP)
bsd	Brian S. Dean's parallel programmer (TPI, ISP)
buspirate	The Bus Pirate in AVR programming mode (ISP)
buspirate_bb	The Bus Pirate in bitbang mode (TPI, ISP)
butterfly	Atmel bootloader (Butterfly Development Board) (SPM)
butterfly_mk	Mikrokoetter.de Butterfly bootloader (SPM)
mkbutterfly	Mikrokoetter.de Butterfly bootloader (SPM)
bwmega	BitWizard ftdi_atmega builtin programmer (TPI, ISP)

c232hm	C232HM cable from FTDI (TPI, ISP)
c2n232i	serial port: reset=dtr sck=!rts sdo=!txd sdi=!cts (TPI, ISP)
ch341a	CH341A programmer: note AVR F_CPU > 6.8 MHz (ISP)
dapa	Direct AVR Parallel Access cable (TPI, ISP)
dasa	serial port: reset=rts sck=dtr sdo=txd sdi=cts (TPI, ISP)
dasa3	serial port: reset=!dtr sck=rts sdo=txd sdi=cts (TPI, ISP)
digilent-hs2	Digilent JTAG HS2 (MPSSE) (TPI, ISP)
dragon_dw	Atmel AVR Dragon (debugWIRE)
dragon_hvsp	Atmel AVR Dragon (HVSP)
dragon_isp	Atmel AVR Dragon (ISP)
dragon_jtag	Atmel AVR Dragon (JTAG, XMEGAJTAG, AVR32JTAG)
dragon_pdi	Atmel AVR Dragon (PDI)
dragon_pp	Atmel AVR Dragon (HVPP)
dryboot	Emulates bootloader programming without the part (SPM)
dryrun	Emulates programming without a programmer (TPI, ISP, PDI, UPDI, HVSP, HVPP, aWire)
dt006	Dontronics DT006 (TPI, ISP)
ehajo-isp	AVR ISP programmer from eHaJo.de (TPI, ISP)
ere-isp-avr	ERE ISP-AVR (TPI, ISP)
flip1	FLIP bootloader using USB DFU v1 (doc7618) (SPM)
flip2	FLIP bootloader using USB DFU v2 (AVR4023) (SPM)
flyswatter2	TinCan Tools Flyswatter 2 (TPI, ISP)
frank-stk200	Frank STK200 (TPI, ISP)
ft2232h	FT2232H/D programmer (TPI, ISP)
ft2232h_jtag	FT2232H based generic JTAG programmer (JTAG)
ft232h	FT232H based generic programmer (TPI, ISP)
ft232h_jtag	FT232H based generic JTAG programmer (JTAG)
ft232r	FT232R based generic programmer (TPI, ISP)
ft245r	FT245R based generic programmer (TPI, ISP)
ft4232h	FT4232H programmer (TPI, ISP)
futurlec	Futurlec.com programming cable (TPI, ISP)
iseavrprog	AVR ISP programmer from iascaled.com (TPI, ISP)
jtag1slow	Atmel JTAG ICE mkI (JTAGmkI)
jtag2dw	Atmel JTAG ICE mkII (debugWIRE)
jtag2fast	Atmel JTAG ICE mkII (JTAG, XMEGAJTAG, AVR32JTAG)
jtag2	Atmel JTAG ICE mkII (JTAG, XMEGAJTAG, AVR32JTAG)
jtag2isp	Atmel JTAG ICE mkII (TPI, ISP)
jtag2pdi	Atmel JTAG ICE mkII (PDI)
jtag2slow	Atmel JTAG ICE mkII (JTAG, XMEGAJTAG, AVR32JTAG)
jtag2updi	JTAGv2 to UPDI bridge (UPDI)
nanoevery	JTAGv2 to UPDI bridge (UPDI)
jtag3	Atmel AVR JTAGICE3 (JTAG, XMEGAJTAG, AVR32JTAG)
jtag3dw	Atmel AVR JTAGICE3 (debugWIRE)
jtag3isp	Atmel AVR JTAGICE3 (ISP)
jtag3pdi	Atmel AVR JTAGICE3 (PDI)
jtag3updi	Atmel AVR JTAGICE3 (UPDI)

jtagkey	Amontec JTAGKey/JTAGKey-Tiny/JTAGKey2 (TPI, ISP)
jtagmkI	Atmel JTAG ICE mkI (JTAGmkI)
jtag1	Atmel JTAG ICE mkI (JTAGmkI)
jtagmkII	Atmel JTAG ICE mkII (JTAG, XMEGAJTAG, AVR32JTAG)
jtagmkII_avr32	Atmel JTAG ICE mkII (aWire)
jtag2avr32	Atmel JTAG ICE mkII (aWire)
ktlink	KT-LINK FT2232H: IO switching, voltage buffers (TPI, ISP)
linuxspi	Use Linux SPI device in /dev/spidev* (TPI, ISP)
lm3s811	Luminary Micro LM3S811 Eval Board (Rev. A) (TPI, ISP)
mib510	Crossbow MIB510 programming board (TPI, ISP)
micronucleus	Micronucleus bootloader (SPM)
nibobee	NIBObee (TPI, ISP)
o-link	O-Link, OpenJTAG ARM JTAG USB (TPI, ISP)
openmoko	Openmoko debug board (v3) (TPI, ISP)
pavr	Jason Kyle's pAVR Serial Programmer (ISP)
pickit2	Microchip PICKit 2 programmer (ISP)
pickit4	MPLAB(R) PICKit 4 (JTAG, XMEGAJTAG)
pickit4_jtag	MPLAB(R) PICKit 4 (JTAG, XMEGAJTAG)
pickit4_dw	MPLAB(R) PICKit 4 (debugWIRE)
pickit4_isp	MPLAB(R) PICKit 4 (ISP)
pickit4_mplab	MPLAB(R) PICKit 4 (JTAG, XMEGAJTAG)
pickit4_mplab_jtag	MPLAB(R) PICKit 4 (JTAG, XMEGAJTAG)
pickit4_mplab_dw	MPLAB(R) PICKit 4 (debugWIRE)
pickit4_mplab_isp	MPLAB(R) PICKit 4 (ISP)
pickit4_mplab_pdi	MPLAB(R) PICKit 4 (PDI)
pickit4_mplab_tpi	MPLAB(R) PICKit 4 (TPI)
pickit4_mplab_updi	MPLAB(R) PICKit 4 (UPDI)
pickit4_pdi	MPLAB(R) PICKit 4 (PDI)
pickit4_tpi	MPLAB(R) PICKit 4 (TPI)
pickit4_updi	MPLAB(R) PICKit 4 (UPDI)
pickit5	MPLAB(R) PICKit 5 (JTAG, XMEGAJTAG)
pickit5_jtag	MPLAB(R) PICKit 5 (JTAG, XMEGAJTAG)
pickit5_dw	MPLAB(R) PICKit 5 (debugWIRE)
pickit5_isp	MPLAB(R) PICKit 5 (ISP)
pickit5_pdi	MPLAB(R) PICKit 5 (PDI)
pickit5_tpi	MPLAB(R) PICKit 5 (TPI)
pickit5_updi	MPLAB(R) PICKit 5 (UPDI)
pickit_basic	MPLAB(R) PICKit Basic (JTAG, XMEGAJTAG)
pickit_basic_mplab	MPLAB(R) PICKit Basic (JTAG, XMEGAJTAG)
pickit_basic_jtag	MPLAB(R) PICKit Basic (JTAG, XMEGAJTAG)
pickit_basic_	MPLAB(R) PICKit Basic (JTAG, XMEGAJTAG)
mplab_jtag	
pickit_basic_dw	MPLAB(R) PICKit Basic (debugWIRE)
pickit_basic_	MPLAB(R) PICKit Basic (debugWIRE)
mplab_dw	
pickit_basic_isp	MPLAB(R) PICKit Basic (ISP)

<code>pickit_basic_</code>	MPLAB(R) PICkit Basic (ISP)
<code>mplab_isp</code>	
<code>pickit_basic_pdi</code>	MPLAB(R) PICkit Basic (PDI)
<code>pickit_basic_</code>	MPLAB(R) PICkit Basic (PDI)
<code>mplab_pdi</code>	
<code>pickit_basic_tpi</code>	MPLAB(R) PICkit Basic (TPI)
<code>pickit_basic_</code>	MPLAB(R) PICkit Basic (TPI)
<code>mplab_tpi</code>	
<code>pickit_basic_updi</code>	MPLAB(R) PICkit Basic (UPDI)
<code>pickit_basic_</code>	MPLAB(R) PICkit Basic (UPDI)
<code>mplab_updi</code>	
<code>picoweb</code>	Picoweb Programming Cable (TPI, ISP)
<code>pkobn_updi</code>	Curiosity nano (nEDBG) (UPDI)
<code>pony-stk200</code>	Pony Prog STK200 (TPI, ISP)
<code>ponyser</code>	ponyprog serial: reset=ltxd sck=rts sdo=dtr sdi=cts (TPI, ISP)
<code>powerdebugger</code>	Atmel PowerDebugger (JTAG, XMEGAJTAG, AVR32JTAG)
<code>powerdebugger_jtag</code>	Atmel PowerDebugger (JTAG, XMEGAJTAG, AVR32JTAG)
<code>powerdebugger_dw</code>	Atmel PowerDebugger (debugWIRE)
<code>powerdebugger_isp</code>	Atmel PowerDebugger (ISP)
<code>powerdebugger_pdi</code>	Atmel PowerDebugger (PDI)
<code>powerdebugger_tpi</code>	Atmel PowerDebugger (TPI)
<code>powerdebugger_updi</code>	Atmel PowerDebugger (UPDI)
<code>raspberrypi_gpio</code>	Raspberry Pi GPIO via sysfs/libgpiod (ISP)
<code>serialupdi</code>	SerialUPDI (UPDI)
<code>serprog</code>	Program via the Serprog protocol from Flashrom (ISP)
<code>siprogram</code>	Lancos SI-Prog (same as ponyser) (TPI, ISP)
<code>snap</code>	MPLAB(R) SNAP (JTAG, XMEGAJTAG)
<code>snap_jtag</code>	MPLAB(R) SNAP (JTAG, XMEGAJTAG)
<code>snap_dw</code>	MPLAB(R) SNAP (debugWIRE)
<code>snap_isp</code>	MPLAB(R) SNAP (ISP)
<code>snap_mplab</code>	MPLAB(R) SNAP (JTAG, XMEGAJTAG)
<code>snap_mplab_jtag</code>	MPLAB(R) SNAP (JTAG, XMEGAJTAG)
<code>snap_mplab_dw</code>	MPLAB(R) SNAP (debugWIRE)
<code>snap_mplab_isp</code>	MPLAB(R) SNAP (ISP)
<code>snap_mplab_pdi</code>	MPLAB(R) SNAP (PDI)
<code>snap_mplab_tpi</code>	MPLAB(R) SNAP (TPI)
<code>snap_mplab_updi</code>	MPLAB(R) SNAP (UPDI)
<code>snap_pdi</code>	MPLAB(R) SNAP (PDI)
<code>snap_tpi</code>	MPLAB(R) SNAP (TPI)
<code>snap_updi</code>	MPLAB(R) SNAP (UPDI)
<code>sp12</code>	Steve Bolt's Programmer (TPI, ISP)
<code>stk200</code>	STK200 starter kit (TPI, ISP)
<code>stk500</code>	Atmel STK500 (probes v2 first then v1) (ISP)
<code>stk500hvsp</code>	Atmel STK500 v2 (HVSP)
<code>scratchmonkey_hvsp</code>	Atmel STK500 v2 (HVSP)
<code>stk500pp</code>	Atmel STK500 v2 (HVPP)

scratchmonkey_pp	Atmel STK500 v2 (HVPP)
stk500v1	Atmel STK500 v1 (ISP)
stk500v2	Atmel STK500 v2 (TPI, ISP)
scratchmonkey	Atmel STK500 v2 (TPI, ISP)
stk600	Atmel STK600 (ISP, PDI)
stk600hvsp	Atmel STK600 (HVSP)
stk600pp	Atmel STK600 (HVPP)
tc2030	Tag-Connect TC2030 (TPI, ISP)
teensy	Teensy bootloader (SPM)
tigard	Tigard interface board (TPI, ISP)
ttd1232r	FTDI TTL232R-5V with ICSP adapter (TPI, ISP)
tumpa	TIAO USB Multi-Protocol Adapter (TPI, ISP)
tumpa-b	TIAO USB Multi-Protocol Adapter (TPI, ISP)
tumpa_jtag	TIAO USB Multi-Protocol Adapter (JTAG)
um232h	UM232H module from FTDI (TPI, ISP)
uncompatino	uncompatino with all pairs of pins shorted (TPI, ISP)
urclock	Urboot bootloaders using urprotocol (SPM)
usbasp	USBasp ISP and TPI programmer (TPI, ISP)
usbasp-clone	Any usbasp clone with correct VID/PID (TPI, ISP)
usbtiny	USBtiny simple USB programmer (TPI, ISP)
wiring	Wiring bootloader using STK500 v2 protocol (SPM)
xbee	XBeeBoot Over-The-Air bootloader (STK500 v1) (SPM)
xil	Xilinx JTAG cable (TPI, ISP)
xplainedmini	Atmel XplainedMini (ISP)
xplainedmini_isp	Atmel XplainedMini (ISP)
xplainedmini_dw	Atmel XplainedMini (debugWIRE)
xplainedmini_tpi	Atmel XplainedMini (TPI)
xplainedmini_updi	Atmel XplainedMini (UPDI)
xplainedpro	Atmel XplainedPro (JTAG, XMEGAJTAG, AVR32JTAG)
xplainedpro_jtag	Atmel XplainedPro (JTAG, XMEGAJTAG, AVR32JTAG)
xplainedpro_pdi	Atmel XplainedPro (PDI)
xplainedpro_updi	Atmel XplainedPro (UPDI)

Appendix D List of Parts

AVRDUDE supports the parts below: the left column lists the part's id, whilst the right column contains its official part name; alternative names, if any; and the list of available programming interfaces in brackets; see Section 4.2 [Programmer Definitions], page 54). There is more detail about each part in the AVRDUDE configuration file.

uc3a0512	AT32UC3A0512, AT32UC3A0512AU (AVR32JTAG, aWire)
89S51	AT89S51 (ISP, HVPP)
89S52	AT89S52 (ISP, HVPP)
c128	AT90CAN128 (SPM, ISP, HVPP, JTAG, JTAGmkI)
c32	AT90CAN32 (SPM, ISP, HVPP, JTAG)
c64	AT90CAN64 (SPM, ISP, HVPP, JTAG)
pwm1	AT90PWM1 (SPM, ISP, HVPP, debugWIRE)
pwm161	AT90PWM161 (SPM, ISP, HVPP, debugWIRE)
pwm2	AT90PWM2 (SPM, ISP, HVPP, debugWIRE)
pwm216	AT90PWM216 (SPM, ISP, HVPP, debugWIRE)
pwm2b	AT90PWM2B (SPM, ISP, HVPP, debugWIRE)
pwm3	AT90PWM3 (SPM, ISP, HVPP, debugWIRE)
pwm316	AT90PWM316 (SPM, ISP, HVPP, debugWIRE)
pwm3b	AT90PWM3B (SPM, ISP, HVPP, debugWIRE)
pwm81	AT90PWM81, AT90PWM81EP (SPM, ISP, HVPP, debugWIRE)
1200	AT90S1200, AT90S1200A (SPM, ISP, HVPP)
2313	AT90S2313 (SPM, ISP, HVPP)
2323	AT90S2323 (SPM, ISP, HVSP)
2333	AT90S2333 (SPM, ISP, HVPP)
2343	AT90S2343 (SPM, ISP, HVSP)
4414	AT90S4414 (SPM, ISP, HVPP)
4433	AT90S4433 (SPM, ISP, HVPP)
4434	AT90S4434 (SPM, ISP, HVPP)
8515	AT90S8515 (SPM, ISP, HVPP)
8535	AT90S8535 (SPM, ISP, HVPP)
usb1286	AT90USB1286 (SPM, ISP, HVPP, JTAG)
usb1287	AT90USB1287 (SPM, ISP, HVPP, JTAG)
usb162	AT90USB162 (SPM, ISP, HVPP, debugWIRE)
usb646	AT90USB646 (SPM, ISP, HVPP, JTAG)
usb647	AT90USB647 (SPM, ISP, HVPP, JTAG)
usb82	AT90USB82 (SPM, ISP, HVPP, debugWIRE)
a5505	ATA5505 (SPM, ISP, HVPP, debugWIRE)
a6612c	ATA6612C (SPM, ISP, HVPP, debugWIRE)
a6613c	ATA6613C (SPM, ISP, HVPP, debugWIRE)
a6614q	ATA6614Q (SPM, ISP, HVPP, debugWIRE)
a6616c	ATA6616C (SPM, ISP, HVPP, debugWIRE)
a6617c	ATA6617C (SPM, ISP, HVPP, debugWIRE)
a664251	ATA664251 (SPM, ISP, HVPP, debugWIRE)
m103	ATmega103, ATmega103L (SPM, ISP, HVPP)
m128	ATmega128, ATmega128L (SPM, ISP, HVPP, JTAG, JTAGmkI)

m1280	ATmega1280, ATmega1280V (SPM, ISP, HVPP, JTAG)
m1281	ATmega1281, ATmega1281V (SPM, ISP, HVPP, JTAG)
m1284	ATmega1284 (SPM, ISP, HVPP, JTAG)
m1284p	ATmega1284P (SPM, ISP, HVPP, JTAG)
m1284rfr2	ATmega1284RFR2 (SPM, ISP, HVPP, JTAG)
m128a	ATmega128A (SPM, ISP, HVPP, JTAG, JTAGmkI)
m128rfa1	ATmega128RFA1 (SPM, ISP, HVPP, JTAG)
m128rfr2	ATmega128RFR2 (SPM, ISP, HVPP, JTAG)
m16	ATmega16, ATmega16L (SPM, ISP, HVPP, JTAG, JTAGmkI)
m1608	ATmega1608 (SPM, UPDI)
m1609	ATmega1609 (SPM, UPDI)
m161	ATmega161, ATmega161L (SPM, ISP, HVPP)
m162	ATmega162, ATmega162L, ATmega162V (SPM, ISP, HVPP, JTAG, JTAGmkI)
m163	ATmega163, ATmega163L (SPM, ISP, HVPP)
m164a	ATmega164A (SPM, ISP, HVPP, JTAG)
m164p	ATmega164P, ATmega164PV (SPM, ISP, HVPP, JTAG)
m164pa	ATmega164PA (SPM, ISP, HVPP, JTAG)
m165	ATmega165, ATmega165V (SPM, ISP, HVPP, JTAG)
m165a	ATmega165A (SPM, ISP, HVPP, JTAG)
m165p	ATmega165P, ATmega165PV (SPM, ISP, HVPP, JTAG)
m165pa	ATmega165PA (SPM, ISP, HVPP, JTAG)
m168	ATmega168, ATmega168V (SPM, ISP, HVPP, debugWIRE)
m168a	ATmega168A (SPM, ISP, HVPP, debugWIRE)
m168p	ATmega168P, ATmega168PV (SPM, ISP, HVPP, debugWIRE)
m168pa	ATmega168PA (SPM, ISP, HVPP, debugWIRE)
m168pb	ATmega168PB (SPM, ISP, HVPP, debugWIRE)
m169	ATmega169, ATmega169L, ATmega169V (SPM, ISP, HVPP, JTAG, JTAGmkI)
m169a	ATmega169A (SPM, ISP, HVPP, JTAG, JTAGmkI)
m169p	ATmega169P, ATmega169PV (SPM, ISP, HVPP, JTAG)
m169pa	ATmega169PA (SPM, ISP, HVPP, JTAG)
m16a	ATmega16A (SPM, ISP, HVPP, JTAG, JTAGmkI)
m16hva	ATmega16HVA (SPM, ISP, HVSP, debugWIRE)
m16hvb	ATmega16HVB (SPM, ISP, HVPP, debugWIRE)
m16hvbrevb	ATmega16HVBrevB, ATMEGA16HVB (SPM, ISP, HVPP, debugWIRE)
m16m1	ATmega16M1 (SPM, ISP, HVPP, debugWIRE)
m16u2	ATmega16U2 (SPM, ISP, HVPP, debugWIRE)
m16u4	ATmega16U4, ATmega16U4RC (SPM, ISP, HVPP, JTAG)
m2560	ATmega2560, ATmega2560V (SPM, ISP, HVPP, JTAG)
m2561	ATmega2561, ATmega2561V (SPM, ISP, HVPP, JTAG)
m2564rfr2	ATmega2564RFR2 (SPM, ISP, HVPP, JTAG)
m256rfr2	ATmega256RFR2 (SPM, ISP, HVPP, JTAG)
m32	ATmega32, ATmega32L (SPM, ISP, HVPP, JTAG, JTAGmkI)
m3208	ATmega3208 (SPM, UPDI)

m3209	ATmega3209 (SPM, UPDI)
m324a	ATmega324A (SPM, ISP, HVPP, JTAG)
m324p	ATmega324P, ATmega324PV (SPM, ISP, HVPP, JTAG)
m324pa	ATmega324PA (SPM, ISP, HVPP, JTAG)
m324pb	ATmega324PB (SPM, ISP, HVPP, JTAG)
m325	ATmega325, ATmega325V (SPM, ISP, HVPP, JTAG)
m3250	ATmega3250, ATmega3250V (SPM, ISP, HVPP, JTAG)
m3250a	ATmega3250A (SPM, ISP, HVPP, JTAG)
m3250p	ATmega3250P, ATmega3250PV (SPM, ISP, HVPP, JTAG)
m3250pa	ATmega3250PA (SPM, ISP, HVPP, JTAG)
m325a	ATmega325A (SPM, ISP, HVPP, JTAG)
m325p	ATmega325P, ATmega325PV (SPM, ISP, HVPP, JTAG)
m325pa	ATmega325PA (SPM, ISP, HVPP, JTAG)
m328	ATmega328 (SPM, ISP, HVPP, debugWIRE)
m328p	ATmega328P (SPM, ISP, HVPP, debugWIRE)
m328pb	ATmega328PB (SPM, ISP, HVPP, debugWIRE)
m329	ATmega329, ATmega329V (SPM, ISP, HVPP, JTAG)
m3290	ATmega3290, ATmega3290V (SPM, ISP, HVPP, JTAG)
m3290a	ATmega3290A (SPM, ISP, HVPP, JTAG)
m3290p	ATmega3290P, ATmega3290PV (SPM, ISP, HVPP, JTAG)
m3290pa	ATmega3290PA (SPM, ISP, HVPP, JTAG)
m329a	ATmega329A (SPM, ISP, HVPP, JTAG)
m329p	ATmega329P, ATmega329PV (SPM, ISP, HVPP, JTAG)
m329pa	ATmega329PA (SPM, ISP, HVPP, JTAG)
m32a	ATmega32A (SPM, ISP, HVPP, JTAG, JTAGmkI)
m32c1	ATmega32C1 (SPM, ISP, HVPP, debugWIRE)
m32hvb	ATmega32HVB (SPM, ISP, HVPP, debugWIRE)
m32hvbrevb	ATmega32HVBrevB, ATMEGA32HVB (SPM, ISP, HVPP, debugWIRE)
m32hve2	ATmega32HVE2 (SPM, ISP, HVSP, debugWIRE)
m32m1	ATmega32M1 (SPM, ISP, HVPP, debugWIRE)
m32u2	ATmega32U2 (SPM, ISP, HVPP, debugWIRE)
m32u4	ATmega32U4, ATmega32U4RC (SPM, ISP, HVPP, JTAG)
m406	ATmega406 (SPM, HVPP, JTAG)
m48	ATmega48, ATmega48V (SPM, ISP, HVPP, debugWIRE)
m4808	ATmega4808 (SPM, UPDI)
m4809	ATmega4809 (SPM, UPDI)
m48a	ATmega48A (SPM, ISP, HVPP, debugWIRE)
m48p	ATmega48P, ATmega48PV (SPM, ISP, HVPP, debugWIRE)
m48pa	ATmega48PA (SPM, ISP, HVPP, debugWIRE)
m48pb	ATmega48PB (SPM, ISP, HVPP, debugWIRE)
m64	ATmega64, ATmega64L (SPM, ISP, HVPP, JTAG, JTAGmkI)
m640	ATmega640, ATmega640V (SPM, ISP, HVPP, JTAG)
m644	ATmega644, ATmega644V (SPM, ISP, HVPP, JTAG)
m644a	ATmega644A (SPM, ISP, HVPP, JTAG)
m644p	ATmega644P, ATmega644PV (SPM, ISP, HVPP, JTAG)

m644pa	ATmega644PA (SPM, ISP, HVPP, JTAG)
m644rfr2	ATmega644RFR2 (SPM, ISP, HVPP, JTAG)
m645	ATmega645, ATmega645V (SPM, ISP, HVPP, JTAG)
m6450	ATmega6450, ATmega6450V (SPM, ISP, HVPP, JTAG)
m6450a	ATmega6450A (SPM, ISP, HVPP, JTAG)
m6450p	ATmega6450P (SPM, ISP, HVPP, JTAG)
m645a	ATmega645A (SPM, ISP, HVPP, JTAG)
m645p	ATmega645P (SPM, ISP, HVPP, JTAG)
m649	ATmega649, ATmega649V (SPM, ISP, HVPP, JTAG)
m6490	ATmega6490, ATmega6490V (SPM, ISP, HVPP, JTAG)
m6490a	ATmega6490A (SPM, ISP, HVPP, JTAG)
m6490p	ATmega6490P (SPM, ISP, HVPP, JTAG)
m649a	ATmega649A (SPM, ISP, HVPP, JTAG)
m649p	ATmega649P (SPM, ISP, HVPP, JTAG)
m64a	ATmega64A (SPM, ISP, HVPP, JTAG, JTAGmkI)
m64c1	ATmega64C1 (SPM, ISP, HVPP, debugWIRE)
m64hve2	ATmega64HVE2 (SPM, ISP, HVSP, debugWIRE)
m64m1	ATmega64M1 (SPM, ISP, HVPP, debugWIRE)
m64rfr2	ATmega64RFR2 (SPM, ISP, HVPP, JTAG)
m8	ATmega8, ATmega8L (SPM, ISP, HVPP)
m808	ATmega808 (SPM, UPDI)
m809	ATmega809 (SPM, UPDI)
m8515	ATmega8515, ATmega8515L (SPM, ISP, HVPP)
m8535	ATmega8535, ATmega8535L (SPM, ISP, HVPP)
m88	ATmega88, ATmega88V (SPM, ISP, HVPP, debugWIRE)
m88a	ATmega88A (SPM, ISP, HVPP, debugWIRE)
m88p	ATmega88P, ATmega88PV (SPM, ISP, HVPP, debugWIRE)
m88pa	ATmega88PA (SPM, ISP, HVPP, debugWIRE)
m88pb	ATmega88PB (SPM, ISP, HVPP, debugWIRE)
m8a	ATmega8A (SPM, ISP, HVPP)
m8hva	ATmega8HVA (SPM, ISP, HVSP, debugWIRE)
m8u2	ATmega8U2 (SPM, ISP, HVPP, debugWIRE)
ms128	ATmegaS128 (SPM, ISP, HVPP, JTAG, JTAGmkI)
ms64m1	ATmegaS64M1 (SPM, ISP, HVPP, debugWIRE)
t10	ATtiny10 (TPI)
t102	ATtiny102, ATtiny102F (TPI)
t104	ATtiny104, ATtiny104F (TPI)
t11	ATtiny11, ATtiny11L (HVSP)
t12	ATtiny12, ATtiny12L, ATtiny12V (ISP, HVSP)
t13	ATtiny13, ATtiny13V (SPM, ISP, HVSP, debugWIRE)
t13a	ATtiny13A (SPM, ISP, HVSP, debugWIRE)
t15	ATtiny15, ATtiny15L (ISP, HVSP)
t1604	ATtiny1604 (SPM, UPDI)
t1606	ATtiny1606 (SPM, UPDI)
t1607	ATtiny1607 (SPM, UPDI)
t1614	ATtiny1614 (SPM, UPDI)
t1614auto	ATtiny1614auto, ATtiny1614 (SPM, UPDI)

t1616	ATtiny1616 (SPM, UPDI)
t1616auto	ATtiny1616auto, ATtiny1616 (SPM, UPDI)
t1617	ATtiny1617 (SPM, UPDI)
t1617auto	ATtiny1617auto, ATtiny1617 (SPM, UPDI)
t1624	ATtiny1624 (SPM, UPDI)
t1626	ATtiny1626 (SPM, UPDI)
t1627	ATtiny1627 (SPM, UPDI)
t1634	ATtiny1634, ATtiny1634R (SPM, ISP, HVPP, debugWIRE)
t1634r	ATtiny1634R (SPM, ISP, HVPP, debugWIRE)
t167	ATtiny167 (SPM, ISP, HVPP, debugWIRE)
t20	ATtiny20 (TPI)
t202	ATtiny202 (SPM, UPDI)
t204	ATtiny204 (SPM, UPDI)
t212	ATtiny212 (SPM, UPDI)
t214	ATtiny214 (SPM, UPDI)
t22	ATtiny22, ATtiny22L (SPM, ISP, HVSP)
t2313	ATtiny2313, ATtiny2313V (SPM, ISP, HVPP, debugWIRE)
t2313a	ATtiny2313A (SPM, ISP, HVPP, debugWIRE)
t24	ATtiny24, ATtiny24V (SPM, ISP, HVSP, debugWIRE)
t24a	ATtiny24A (SPM, ISP, HVSP, debugWIRE)
t25	ATtiny25, ATtiny25V (SPM, ISP, HVSP, debugWIRE)
t26	ATtiny26, ATtiny26L (ISP, HVPP)
t261	ATtiny261, ATtiny261V (SPM, ISP, HVPP, debugWIRE)
t261a	ATtiny261A (SPM, ISP, HVPP, debugWIRE)
t28	ATtiny28, ATtiny28L, ATtiny28V (HVPP)
t3216	ATtiny3216 (SPM, UPDI)
t3216auto	ATtiny3216auto, ATtiny3216 (SPM, UPDI)
t3217	ATtiny3217 (SPM, UPDI)
t3217auto	ATtiny3217auto, ATtiny3217 (SPM, UPDI)
t3224	ATtiny3224 (SPM, UPDI)
t3226	ATtiny3226 (SPM, UPDI)
t3227	ATtiny3227 (SPM, UPDI)
t4	ATtiny4 (TPI)
t40	ATtiny40 (TPI)
t402	ATtiny402 (SPM, UPDI)
t404	ATtiny404 (SPM, UPDI)
t406	ATtiny406 (SPM, UPDI)
t412	ATtiny412 (SPM, UPDI)
t414	ATtiny414 (SPM, UPDI)
t416	ATtiny416 (SPM, UPDI)
t416auto	ATtiny416auto, ATtiny416 (SPM, UPDI)
t417	ATtiny417 (SPM, UPDI)
t417auto	ATtiny417auto, ATtiny417 (SPM, UPDI)
t424	ATtiny424 (SPM, UPDI)
t426	ATtiny426 (SPM, UPDI)
t427	ATtiny427 (SPM, UPDI)
t4313	ATtiny4313 (SPM, ISP, HVPP, debugWIRE)

t43u	ATtiny43U (SPM, ISP, HVPP, debugWIRE)
t44	ATtiny44, ATtiny44V (SPM, ISP, HVSP, debugWIRE)
t441	ATtiny441 (SPM, ISP, HVSP, debugWIRE)
t44a	ATtiny44A (SPM, ISP, HVSP, debugWIRE)
t45	ATtiny45, ATtiny45V (SPM, ISP, HVSP, debugWIRE)
t461	ATtiny461, ATtiny461V (SPM, ISP, HVPP, debugWIRE)
t461a	ATtiny461A (SPM, ISP, HVPP, debugWIRE)
t48	ATtiny48 (SPM, ISP, HVPP, debugWIRE)
t5	ATtiny5 (TPI)
t804	ATtiny804 (SPM, UPDI)
t806	ATtiny806 (SPM, UPDI)
t807	ATtiny807 (SPM, UPDI)
t814	ATtiny814 (SPM, UPDI)
t814auto	ATtiny814auto, ATtiny814 (SPM, UPDI)
t816	ATtiny816 (SPM, UPDI)
t816auto	ATtiny816auto, ATtiny816 (SPM, UPDI)
t817	ATtiny817 (SPM, UPDI)
t817auto	ATtiny817auto, ATtiny817 (SPM, UPDI)
t824	ATtiny824 (SPM, UPDI)
t826	ATtiny826 (SPM, UPDI)
t827	ATtiny827 (SPM, UPDI)
t828	ATtiny828, ATtiny828R (SPM, ISP, HVPP, debugWIRE)
t828r	ATtiny828R (SPM, ISP, HVPP, debugWIRE)
t84	ATtiny84, ATtiny84V (SPM, ISP, HVSP, debugWIRE)
t841	ATtiny841 (SPM, ISP, HVSP, debugWIRE)
t84a	ATtiny84A (SPM, ISP, HVSP, debugWIRE)
t85	ATtiny85, ATtiny85V (SPM, ISP, HVSP, debugWIRE)
t861	ATtiny861, ATtiny861V (SPM, ISP, HVPP, debugWIRE)
t861a	ATtiny861A (SPM, ISP, HVPP, debugWIRE)
t87	ATtiny87 (SPM, ISP, HVPP, debugWIRE)
t88	ATtiny88 (SPM, ISP, HVPP, debugWIRE)
t9	ATtiny9 (TPI)
x128a1	ATxmega128A1 (SPM, PDI, XMEGAJTAG)
x128a1d	ATxmega128A1revD (SPM, PDI, XMEGAJTAG)
x128a1u	ATxmega128A1U (SPM, PDI, XMEGAJTAG)
x128a3	ATxmega128A3 (SPM, PDI, XMEGAJTAG)
x128a3u	ATxmega128A3U (SPM, PDI, XMEGAJTAG)
x128a4	ATxmega128A4 (SPM, PDI, XMEGAJTAG)
x128a4u	ATxmega128A4U (SPM, PDI)
x128b1	ATxmega128B1 (SPM, PDI, XMEGAJTAG)
x128b3	ATxmega128B3 (SPM, PDI, XMEGAJTAG)
x128c3	ATxmega128C3 (SPM, PDI)
x128d3	ATxmega128D3 (SPM, PDI)
x128d4	ATxmega128D4 (SPM, PDI)
x16a4	ATxmega16A4 (SPM, PDI)
x16a4u	ATxmega16A4U (SPM, PDI)
x16c4	ATxmega16C4 (SPM, PDI)

x16d4	ATxmega16D4 (SPM, PDI)
x16e5	ATxmega16E5 (SPM, PDI)
x192a1	ATxmega192A1 (SPM, PDI, XMEGAJTAG)
x192a3	ATxmega192A3 (SPM, PDI, XMEGAJTAG)
x192a3u	ATxmega192A3U (SPM, PDI, XMEGAJTAG)
x192c3	ATxmega192C3 (SPM, PDI)
x192d3	ATxmega192D3 (SPM, PDI)
x256a1	ATxmega256A1 (SPM, PDI, XMEGAJTAG)
x256a3	ATxmega256A3 (SPM, PDI, XMEGAJTAG)
x256a3b	ATxmega256A3B (SPM, PDI, XMEGAJTAG)
x256a3bu	ATxmega256A3BU (SPM, PDI, XMEGAJTAG)
x256a3u	ATxmega256A3U (SPM, PDI, XMEGAJTAG)
x256c3	ATxmega256C3 (SPM, PDI)
x256d3	ATxmega256D3 (SPM, PDI)
x32a4	ATxmega32A4 (SPM, PDI)
x32a4u	ATxmega32A4U (SPM, PDI)
x32c3	ATxmega32C3 (SPM, PDI)
x32c4	ATxmega32C4 (SPM, PDI)
x32d3	ATxmega32D3 (SPM, PDI)
x32d4	ATxmega32D4 (SPM, PDI)
x32e5	ATxmega32E5 (SPM, PDI)
x384c3	ATxmega384C3 (SPM, PDI)
x384d3	ATxmega384D3 (SPM, PDI)
x64a1	ATxmega64A1 (SPM, PDI, XMEGAJTAG)
x64a1u	ATxmega64A1U (SPM, PDI, XMEGAJTAG)
x64a3	ATxmega64A3 (SPM, PDI, XMEGAJTAG)
x64a3u	ATxmega64A3U (SPM, PDI, XMEGAJTAG)
x64a4	ATxmega64A4 (SPM, PDI, XMEGAJTAG)
x64a4u	ATxmega64A4U (SPM, PDI)
x64b1	ATxmega64B1 (SPM, PDI, XMEGAJTAG)
x64b3	ATxmega64B3 (SPM, PDI, XMEGAJTAG)
x64c3	ATxmega64C3 (SPM, PDI)
x64d3	ATxmega64D3 (SPM, PDI)
x64d4	ATxmega64D4 (SPM, PDI)
x8e5	ATxmega8E5 (SPM, PDI)
128da28	AVR128DA28, AVR128DA28T (SPM, UPDI)
128da28s	AVR128DA28S (SPM, UPDI)
128da32	AVR128DA32, AVR128DA32T (SPM, UPDI)
128da32s	AVR128DA32S (SPM, UPDI)
128da48	AVR128DA48, AVR128DA48T (SPM, UPDI)
128da48s	AVR128DA48S (SPM, UPDI)
128da64	AVR128DA64, AVR128DA64T (SPM, UPDI)
128da64s	AVR128DA64S (SPM, UPDI)
128db28	AVR128DB28, AVR128DB28T (SPM, UPDI)
128db32	AVR128DB32, AVR128DB32T (SPM, UPDI)
128db48	AVR128DB48, AVR128DB48T (SPM, UPDI)
128db64	AVR128DB64, AVR128DB64T (SPM, UPDI)

16dd14	AVR16DD14 (SPM, UPDI)
16dd20	AVR16DD20 (SPM, UPDI)
16dd28	AVR16DD28 (SPM, UPDI)
16dd32	AVR16DD32 (SPM, UPDI)
16du14	AVR16DU14 (SPM, UPDI)
16du20	AVR16DU20 (SPM, UPDI)
16du28	AVR16DU28 (SPM, UPDI)
16du32	AVR16DU32 (SPM, UPDI)
16ea28	AVR16EA28 (SPM, UPDI)
16ea32	AVR16EA32 (SPM, UPDI)
16ea48	AVR16EA48 (SPM, UPDI)
16eb14	AVR16EB14 (SPM, UPDI)
16eb20	AVR16EB20 (SPM, UPDI)
16eb28	AVR16EB28 (SPM, UPDI)
16eb32	AVR16EB32 (SPM, UPDI)
16la14	AVR16LA14, AVR16LA14T (SPM, UPDI)
16la20	AVR16LA20, AVR16LA20T (SPM, UPDI)
16la28	AVR16LA28, AVR16LA28T (SPM, UPDI)
16la32	AVR16LA32, AVR16LA32T (SPM, UPDI)
32da28	AVR32DA28, AVR32DA28T (SPM, UPDI)
32da28s	AVR32DA28S (SPM, UPDI)
32da32	AVR32DA32, AVR32DA32T (SPM, UPDI)
32da32s	AVR32DA32S (SPM, UPDI)
32da48	AVR32DA48, AVR32DA48T (SPM, UPDI)
32da48s	AVR32DA48S (SPM, UPDI)
32db28	AVR32DB28, AVR32DB28T (SPM, UPDI)
32db32	AVR32DB32, AVR32DB32T (SPM, UPDI)
32db48	AVR32DB48, AVR32DB48T (SPM, UPDI)
32dd14	AVR32DD14 (SPM, UPDI)
32dd20	AVR32DD20 (SPM, UPDI)
32dd28	AVR32DD28 (SPM, UPDI)
32dd32	AVR32DD32 (SPM, UPDI)
32du14	AVR32DU14 (SPM, UPDI)
32du20	AVR32DU20 (SPM, UPDI)
32du28	AVR32DU28 (SPM, UPDI)
32du32	AVR32DU32 (SPM, UPDI)
32ea28	AVR32EA28 (SPM, UPDI)
32ea32	AVR32EA32 (SPM, UPDI)
32ea48	AVR32EA48 (SPM, UPDI)
32eb14	AVR32EB14 (SPM, UPDI)
32eb20	AVR32EB20 (SPM, UPDI)
32eb28	AVR32EB28 (SPM, UPDI)
32eb32	AVR32EB32 (SPM, UPDI)
32la14	AVR32LA14, AVR32LA14T (SPM, UPDI)
32la20	AVR32LA20, AVR32LA20T (SPM, UPDI)
32la28	AVR32LA28, AVR32LA28T (SPM, UPDI)
32la32	AVR32LA32, AVR32LA32T (SPM, UPDI)

32sd20	AVR32SD20 (SPM, UPDI)
32sd28	AVR32SD28 (SPM, UPDI)
32sd32	AVR32SD32 (SPM, UPDI)
64da28	AVR64DA28, AVR64DA28T (SPM, UPDI)
64da28s	AVR64DA28S (SPM, UPDI)
64da32	AVR64DA32, AVR64DA32T (SPM, UPDI)
64da32s	AVR64DA32S (SPM, UPDI)
64da48	AVR64DA48, AVR64DA48T (SPM, UPDI)
64da48s	AVR64DA48S (SPM, UPDI)
64da64	AVR64DA64, AVR64DA64T (SPM, UPDI)
64da64s	AVR64DA64S (SPM, UPDI)
64db28	AVR64DB28, AVR64DB28T (SPM, UPDI)
64db32	AVR64DB32, AVR64DB32T (SPM, UPDI)
64db48	AVR64DB48, AVR64DB48T (SPM, UPDI)
64db64	AVR64DB64, AVR64DB64T (SPM, UPDI)
64dd14	AVR64DD14 (SPM, UPDI)
64dd20	AVR64DD20 (SPM, UPDI)
64dd28	AVR64DD28 (SPM, UPDI)
64dd32	AVR64DD32 (SPM, UPDI)
64du28	AVR64DU28 (SPM, UPDI)
64du32	AVR64DU32 (SPM, UPDI)
64ea28	AVR64EA28 (SPM, UPDI)
64ea32	AVR64EA32 (SPM, UPDI)
64ea48	AVR64EA48 (SPM, UPDI)
8ea28	AVR8EA28 (SPM, UPDI)
8ea32	AVR8EA32 (SPM, UPDI)
lgt168p	LGT8F168P (SPM, ISP, HVPP, debugWIRE)
lgt328p	LGT8F328P (SPM, ISP, HVPP, debugWIRE)
lgt88p	LGT8F88P (SPM, ISP, HVPP, debugWIRE)

Notes

1. Support of 32-bit AVR (via aWire or AVT32JTAG) is experimental at best.
2. Flash addressing above 128 KB is not supported by all programming hardware, though most will support it.
3. The ISP programming protocol of the AT90S1200 differs in subtle ways from that of other AVRs. Thus, not all ISP programmers support this device. Known to work are all direct bitbang programmers, and all programmers talking the STK500v2 protocol.
4. Not all programmers can serve all memories that a part has. Bootloader can never write to fuses, for example.

Appendix E List of Memories

E.1 Classic parts

Classic devices may have the following memories in addition to `eeeprom`, `flash`, `signature` and `lock`:

<code>calibration</code>	One or more bytes of RC oscillator calibration data
<code>efuse</code>	Extended fuse byte
<code>fuse</code>	Fuse byte in devices that have only a single fuse byte
<code>hfuse</code>	High fuse byte
<code>lfuse</code>	Low fuse byte
<code>prodsig</code>	Signature, calibration byte and serial number in a small read-only memory, which is only documented to be available for ATmega324PB, ATmega328PB, ATtiny102 and ATtiny104; AVRDUDE generally tries to make this memory available, also for parts where it is not documented, but not all programmers may be able to read this memory
<code>sigrow</code>	Memory alias for <code>prodsig</code>
<code>sernum</code>	The serial number part of <code>prodsig</code> ; owing to scarce documentation this may not actually turn out to be a serial number or be readable by some programmers
<code>usersig</code>	Three extra flash pages for firmware settings; this memory is not erased during a chip erase. Only some classic parts, ATmega(64 128 256 644 1284 2564)RFR2, have a <code>usersig</code> memory. <code>Usersig</code> is different to <code>flash</code> in the sense that it can neither be accessed with ISP serial programming nor written to by bootloaders. AVRDUDE offers JTAG programming of classic-part <code>usersig</code> memories. As with all flash-type memories the <code>-U</code> option can only write 0-bits but not 1-bits. Hence, <code>usersig</code> needs to be erased before a file can be written to this memory region, e.g., using <code>-T "erase usersig" -U usersig:w:parameters.hex:i</code>
<code>io</code>	Volatile register memory; it cannot be accessed by external programming methods only by bootloaders, which has limited use unless the bootloader jumps to the application directly, i.e., without a WDT reset
<code>sram</code>	Volatile RAM memory; like <code>io</code> it cannot be accessed by external programming

E.2 ATxmegas

ATxmega devices have the following memories in addition to `eeeprom`, `flash`, `signature` and `lock`:

<code>application</code>	Application flash area
<code>apptable</code>	Application table flash area

boot	Boot flash area
calibration	An area of 4 (ATxmega-A series) or 5 bytes (ATxmega-B/C/D/E) with oscillator calibration values; this is a sub-memory of prodsig
fuses	A logical memory of 7 bytes containing all fuseX of a part, which can be used to program all fuses at the same time; note that some of the fuse bytes will be reserved, though
fuse0	A.k.a. jtaguid : JTAG user ID for some devices
fuse1	Watchdog configuration
fuse6	Fault detection action configuration TC4/5 for ATxmega E series parts
fuseN	Other fuse bytes of ATxmega devices, where <i>N</i> is 2, 4 or 5, for system configuration
prodsig	The production signature row is a read-only memory section for factory programmed data such as calibration values for oscillators or analogue modules; it also contains a serial number that consists of the production lot number, wafer number and wafer coordinates for the part
sernum	Serial number with a unique ID for the part consisting of 10 bytes; these are part of the prodsig memory above
sigrow	Memory alias for prodsig
tempsense	A two-byte memory, which is located within prodsig ; it contains a 12-bit temperature sensor calibration value
usersig	Additional flash memory page that can be used for firmware settings; this memory is not erased during a chip erase
io	Volatile register memory; AVRDUDE can read this memory but not write to it using external programming
sram	Volatile RAM memory; cannot be usefully accessed by external programming

E.3 Modern AVR Parts

Modern 8-bit AVR devices have the following memories in addition to **eeeprom**, **flash**, **signature** and **lock**:

fuse0	A.k.a. wdtcfg : watchdog configuration
fuse1	A.k.a. bodcfg : brownout detection configuration
fuse2	A.k.a. osccfg : oscillator configuration
fuse3	A.k.a. pincfg (not all devices): UPDI and reset pin configuration
fuse4	A.k.a. tcd0cfg (not all devices): timer counter type D configuration A.k.a. hwmoncfg (not all devices): CPU safety function checks for illegal op-codes

<code>fuse5</code>	A.k.a. <code>syscfg0</code> : system configuration 0
<code>fuse6</code>	A.k.a. <code>syscfg1</code> : system configuration 1
<code>fuse7</code>	A.k.a. <code>append</code> or <code>codesize</code> : either the end of the application code section or the code size in blocks of 256/512 bytes
<code>fuse8</code>	A.k.a. <code>bootend</code> or <code>bootsize</code> : end of the boot section or the boot size in blocks of 256/512 bytes
<code>fusea</code>	A.k.a. <code>pdicfg</code> : configures/locks updi access; it is the only fuse that consists of two bytes
<code>fuses</code>	A logical memory of up to 16 bytes containing all <code>fuseX</code> of a part, which can be used to program all fuses at the same time
<code>osc16err</code>	Two bytes typically describing the 16 MHz oscillator frequency error at 3 V and 5 V, respectively (not all devices)
<code>osc20err</code>	Two bytes typically describing the 20 MHz oscillator frequency error at 3 V and 5 V, respectively (not all devices)
<code>osccal16</code>	One or two oscillator calibration bytes for 16 MHz (not all devices)
<code>osccal20</code>	One or two oscillator calibration bytes for 20 MHz (not all devices)
<code>prodsig</code>	Read-only memory section for factory programmed data such as the signature, calibration values and serial number
<code>sigrow</code>	Memory alias for <code>prodsig</code>
<code>sernum</code>	Serial number with a unique ID for the part (10 or 16 bytes)
<code>tempsense</code>	Temperature sensor calibration values
<code>bootrow</code>	Extra page of memory that is only accessible by the MCU in bootloader code; UDPI can read and write this memory only when the device is unlocked
<code>userrow</code>	Extra page of EEPROM memory that can be used for firmware settings; this memory is not erased during a chip erase
<code>sib</code>	Special system information block memory with information about AVR family, chip revision etc.
<code>io</code>	Volatile register memory; AVRDUDE can program this memory but this is of limited utility because anything written to the io memory will be undefined or lost after reset; writing to individual registers in the terminal can still be used, e.g., to test I/O ports
<code>sram</code>	Volatile RAM memory; can be read and written but contents will be lost after reset

Concept Index

!

! (subshell) 44

—

--baud *baudrate* 6
 --bitclock *bitclock* 6
 --command *cmd* 12
 --config *config-file* 7
 --erase 8
 --exitspecs *exitspec*[. . .] 9
 --extended *parameter* 15
 --force 9
 --help 15
 --isp-clock-delay *delay* 9
 --keep-trailing-0xff 8
 --logfile *logfile* 10
 --memory *mem:op:file[:fmt]* 12
 --noconfig 8
 --noerase 8
 --noverify-memory 15
 --osccal 10
 --part *partname* 6
 --part *wildcard/flags* 6
 --port *port* 10
 --programmer *programmer-id* 7
 --programmer *wildcard/flags* 7
 --quell 12
 --reconnect 12
 --terminal 12
 --test-memory 10
 --verbose 15
 -A 8
 -b *baudrate* 6
 -B *bitclock* 6
 -c *programmer-id* 7
 -c *wildcard/flags* 7
 -C *config-file* 7
 -D 8
 -e 8
 -E *d_high* 16
 -E *d_low* 16
 -E *exitspec*[. . .] 9
 -E *noreset* 16
 -E *novcc* 16
 -E *reset* 16
 -E *vcc* 16
 -F 9
 -h 15
 -i *delay* 9
 -l *logfile* 10
 -n 10
 -N 8
 -O 10

-p *partname* 6
 -p *wildcard/flags* 6
 -P *port* 10
 -q 12
 -r 12
 -t 12
 -T *cmd* 12
 -U *mem:op:file[:fmt]* 12
 -v 15
 -V 15
 -x Arduino 21
 -x Atmel-ICE 18
 -x AVR Dragon 18
 -x AVR109 20
 -x AVR910 21
 -x BusPirate 24
 -x Curiosity Nano 20
 -x dryboot 17
 -x dryrun 17
 -x flip2 16
 -x jtag2updi 27
 -x JTAG ICE mkII/3 18
 -x linuxgpio 16
 -x linuxspi 16, 27
 -x Micronucleus bootloader 26
 -x MPLAB(R) SNAP 18
 -x parallel port programmers 16
 -x *parameter* 15
 -x PICkit 4 18
 -x PICkit 5 18
 -x PICkit2 26
 -x pickit4_mplab, pickit5 16
 -x Power Debugger 18
 -x raspberry_pi_gpio 16
 -x serialupdi 27
 -x serprog 27
 -x STK500 20
 -x STK600 20
 -x Teensy bootloader 26
 -x Urclock 21
 -x USBasp 26
 -x Wiring 26
 -x xbee 27
 -x Xplained Mini 19

?

? (help) 44

1

1200.....	88
128da28.....	94
128da28s.....	94
128da32.....	94
128da32s.....	94
128da48.....	94
128da48s.....	94
128da64.....	94
128da64s.....	94
128db28.....	94
128db32.....	94
128db48.....	94
128db64.....	94
16dd14.....	94
16dd20.....	95
16dd28.....	95
16dd32.....	95
16du14.....	95
16du20.....	95
16du28.....	95
16du32.....	95
16ea28.....	95
16ea32.....	95
16ea48.....	95
16eb14.....	95
16eb20.....	95
16eb28.....	95
16eb32.....	95
16la14.....	95
16la20.....	95
16la28.....	95
16la32.....	95

2

2232h.....	83
2232hio.....	83
2313.....	88
2323.....	88
2333.....	88
2343.....	88

3

32da28.....	95
32da28s.....	95
32da32.....	95
32da32s.....	95
32da48.....	95
32da48s.....	95
32db28.....	95
32db32.....	95
32db48.....	95
32dd14.....	95
32dd20.....	95
32dd28.....	95
32dd32.....	95

32du14.....	95
32du20.....	95
32du28.....	95
32du32.....	95
32ea28.....	95
32ea32.....	95
32ea48.....	95
32eb14.....	95
32eb20.....	95
32eb32.....	95
32la14.....	95
32la20.....	95
32la28.....	95
32la32.....	95
32sd20.....	95
32sd28.....	96
32sd32.....	96

4

4232h.....	83
4414.....	88
4433.....	88
4434.....	88

6

64da28.....	96
64da28s.....	96
64da32.....	96
64da32s.....	96
64da48.....	96
64da48s.....	96
64da64.....	96
64da64s.....	96
64db28.....	96
64db32.....	96
64db48.....	96
64db64.....	96
64dd14.....	96
64dd20.....	96
64dd28.....	96
64dd32.....	96
64du28.....	96
64du32.....	96
64ea28.....	96
64ea32.....	96
64ea48.....	96

8

8515	88
8535	88
89isp	83
89S51	88
89S52	88
8ea28	96
8ea32	96

A

a5505	88
a6612c	88
a6613c	88
a6614q	88
a6616c	88
a6617c	88
a664251	88
abcmmini	83
ABCmini Board	83
abort	42
adafruit_gemma	83
alf	83
allow_subshells	53
Altera ByteBlaster	83
Amontec JTAGKey	84
append	99
application	97
apptable	97
arduino	83
Arduino bootloader	3, 21, 83
Arduino Gemma bootloader	83
Arduino-branded USBtiny	83
arduino-ft232r	83
Arduino: FT232R to ISP	83
arduino_as_isp	83
arduino_gemma	83
arduinosp	83
arduinosporg	83
At89isp cable	83
AT-ISP v1.1 cable	83
AT32UC3A0512, AT32UC3A0512AU	88
AT89S51	88
AT89S52	88
AT90CAN128	88
AT90CAN32	88
AT90CAN64	88
AT90PWM1	88
AT90PWM161	88
AT90PWM2	88
AT90PWM216	88
AT90PWM2B	88
AT90PWM3	88
AT90PWM316	88
AT90PWM3B	88
AT90PWM81, AT90PWM81EP	88
AT90S1200, AT90S1200A	88
AT90S2313	88

AT90S2323	88
AT90S2333	88
AT90S2343	88
AT90S4414	88
AT90S4433	88
AT90S4434	88
AT90S8515	88
AT90S8535	88
AT90USB1286	88
AT90USB1287	88
AT90USB162	88
AT90USB646	88
AT90USB647	88
AT90USB82	88
ATA5505	88
ATA6612C	88
ATA6613C	88
ATA6614Q	88
ATA6616C	88
ATA6617C	88
ATA664251	88
atisp	83
ATmega103, ATmega103L	88
ATmega128, ATmega128L	88
ATmega1280, ATmega1280V	88
ATmega1281, ATmega1281V	89
ATmega1284	89
ATmega1284P	89
ATmega1284RFR2	89
ATmega128A	89
ATmega128RFA1	89
ATmega128RFR2	89
ATmega16, ATmega16L	89
ATmega1608	89
ATmega1609	89
ATmega161, ATmega161L	89
ATmega162, ATmega162L, ATmega162V	89
ATmega163, ATmega163L	89
ATmega164A	89
ATmega164P, ATmega164PV	89
ATmega164PA	89
ATmega165, ATmega165V	89
ATmega165A	89
ATmega165P, ATmega165PV	89
ATmega165PA	89
ATmega168, ATmega168V	89
ATmega168A	89
ATmega168P, ATmega168PV	89
ATmega168PA	89
ATmega168PB	89
ATmega169, ATmega169L, ATmega169V	89
ATmega169A	89
ATmega169P, ATmega169PV	89
ATmega169PA	89
ATmega16A	89
ATmega16HVA	89
ATmega16HVB	89
ATmega16HVBrevB, ATMEGA16HVB	89

ATmega16M1	89	ATmega6450A	91
ATmega16U2	89	ATmega6450P	91
ATmega16U4, ATmega16U4RC	89	ATmega645A	91
ATmega2560, ATmega2560V	89	ATmega645P	91
ATmega2561, ATmega2561V	89	ATmega649, ATmega649V	91
ATmega2564RFR2	89	ATmega6490, ATmega6490V	91
ATmega256RFR2	89	ATmega6490A	91
ATmega32, ATmega32L	89	ATmega6490P	91
ATmega3208	89	ATmega649A	91
ATmega3209	89	ATmega649P	91
ATmega324A	90	ATmega64A	91
ATmega324P, ATmega324PV	90	ATmega64C1	91
ATmega324PA	90	ATmega64HVE2	91
ATmega324PB	90	ATmega64M1	91
ATmega325, ATmega325V	90	ATmega64RFR2	91
ATmega3250, ATmega3250V	90	ATmega8, ATmega8L	91
ATmega3250A	90	ATmega808	91
ATmega3250P, ATmega3250PV	90	ATmega809	91
ATmega3250PA	90	ATmega8515, ATmega8515L	91
ATmega325A	90	ATmega8535, ATmega8535L	91
ATmega325P, ATmega325PV	90	ATmega88, ATmega88V	91
ATmega325PA	90	ATmega88A	91
ATmega328	90	ATmega88P, ATmega88PV	91
ATmega328P	90	ATmega88PA	91
ATmega328PB	90	ATmega88PB	91
ATmega329, ATmega329V	90	ATmega8A	91
ATmega3290, ATmega3290V	90	ATmega8HVA	91
ATmega3290A	90	ATmega8U2	91
ATmega3290P, ATmega3290PV	90	ATmegaS128	91
ATmega3290PA	90	ATmegaS64M1	91
ATmega329A	90	Atmel at89isp cable	83
ATmega329P, ATmega329PV	90	Atmel AVR Dragon	3, 18, 84
ATmega329PA	90	Atmel AVR JTAGICE3	2, 18, 84
ATmega32A	90	Atmel bootloader (AVR109, AVR911)	2, 20, 83
ATmega32C1	90	Atmel bootloader (Butterfly)	83
ATmega32HVB	90	Atmel JTAG ICE mkI	2, 84, 85
ATmega32HVBrevB, ATMEGA32HVB	90	Atmel JTAG ICE mkII	2, 18, 84, 85
ATmega32HVE2	90	Atmel low-cost programmer AVR910	21, 83
ATmega32M1	90	Atmel PowerDebugger	18, 86
ATmega32U2	90	Atmel STK500	2, 20, 86
ATmega32U4, ATmega32U4RC	90	Atmel STK500 v1	87
ATmega406	90	Atmel STK500 v2	86, 87
ATmega48, ATmega48V	90	Atmel STK600	2, 20, 69, 87
ATmega4808	90	Atmel XplainedMini	3, 19, 87
ATmega4809	90	Atmel XplainedPro	3, 87
ATmega48A	90	Atmel-ICE	3, 83
ATmega48P, ATmega48PV	90	atmelice	83
ATmega48PA	90	atmelice_dw	83
ATmega48PB	90	atmelice_isp	83
ATmega64, ATmega64L	90	atmelice_jtag	83
ATmega640, ATmega640V	90	atmelice_pdi	83
ATmega644, ATmega644V	90	atmelice_tpi	83
ATmega644A	90	atmelice_updi	83
ATmega644P, ATmega644PV	90	ATtiny10	91
ATmega644PA	90	ATtiny102, ATtiny102F	91
ATmega644RFR2	91	ATtiny104, ATtiny104F	91
ATmega645, ATmega645V	91	ATtiny11, ATtiny11L	91
ATmega6450, ATmega6450V	91	ATtiny12, ATtiny12L, ATtiny12V	91

ATtiny13, ATtiny13V	91	ATtiny44A	93
ATtiny13A	91	ATtiny45, ATtiny45V	93
ATtiny15, ATtiny15L	91	ATtiny461, ATtiny461V	93
ATtiny1604	91	ATtiny461A	93
ATtiny1606	91	ATtiny48	93
ATtiny1607	91	ATtiny5	93
ATtiny1614	91	ATtiny804	93
ATtiny1614auto, ATtiny1614	91	ATtiny806	93
ATtiny1616	91	ATtiny807	93
ATtiny1616auto, ATtiny1616	92	ATtiny814	93
ATtiny1617	92	ATtiny814auto, ATtiny814	93
ATtiny1617auto, ATtiny1617	92	ATtiny816	93
ATtiny1624	92	ATtiny816auto, ATtiny816	93
ATtiny1626	92	ATtiny817	93
ATtiny1627	92	ATtiny817auto, ATtiny817	93
ATtiny1634, ATtiny1634R	92	ATtiny824	93
ATtiny1634R	92	ATtiny826	93
ATtiny167	92	ATtiny827	93
ATtiny20	92	ATtiny828, ATtiny828R	93
ATtiny202	92	ATtiny828R	93
ATtiny204	92	ATtiny84, ATtiny84V	93
ATtiny212	92	ATtiny841	93
ATtiny214	92	ATtiny84A	93
ATtiny22, ATtiny22L	92	ATtiny85, ATtiny85V	93
ATtiny2313, ATtiny2313V	92	ATtiny861, ATtiny861V	93
ATtiny2313A	92	ATtiny861A	93
ATtiny24, ATtiny24V	92	ATtiny87	93
ATtiny24A	92	ATtiny88	93
ATtiny25, ATtiny25V	92	ATtiny9	93
ATtiny26, ATtiny26L	92	ATxmega128A1	93
ATtiny261, ATtiny261V	92	ATxmega128A1revD	93
ATtiny261A	92	ATxmega128A1U	93
ATtiny28, ATtiny28L, ATtiny28V	92	ATxmega128A3	93
ATtiny3216	92	ATxmega128A3U	93
ATtiny3216auto, ATtiny3216	92	ATxmega128A4	93
ATtiny3217	92	ATxmega128A4U	93
ATtiny3217auto, ATtiny3217	92	ATxmega128B1	93
ATtiny3224	92	ATxmega128B3	93
ATtiny3226	92	ATxmega128C3	93
ATtiny3227	92	ATxmega128D3	93
ATtiny4	92	ATxmega128D4	93
ATtiny40	92	ATxmega16A4	93
ATtiny402	92	ATxmega16A4U	93
ATtiny404	92	ATxmega16C4	93
ATtiny406	92	ATxmega16D4	93
ATtiny412	92	ATxmega16E5	94
ATtiny414	92	ATxmega192A1	94
ATtiny416	92	ATxmega192A3	94
ATtiny416auto, ATtiny416	92	ATxmega192A3U	94
ATtiny417	92	ATxmega192C3	94
ATtiny417auto, ATtiny417	92	ATxmega192D3	94
ATtiny424	92	ATxmega256A1	94
ATtiny426	92	ATxmega256A3	94
ATtiny427	92	ATxmega256A3B	94
ATtiny4313	92	ATxmega256A3BU	94
ATtiny43U	92	ATxmega256A3U	94
ATtiny44, ATtiny44V	93	ATxmega256C3	94
ATtiny441	93	ATxmega256D3	94

ATxmega32A4	94	AVR16LA14, AVR16LA14T	95
ATxmega32A4U	94	AVR16LA20, AVR16LA20T	95
ATxmega32C3	94	AVR16LA28, AVR16LA28T	95
ATxmega32C4	94	AVR16LA32, AVR16LA32T	95
ATxmega32D3	94	AVR32DA28, AVR32DA28T	95
ATxmega32D4	94	AVR32DA28S	95
ATxmega32E5	94	AVR32DA32, AVR32DA32T	95
ATxmega384C3	94	AVR32DA32S	95
ATxmega384D3	94	AVR32DA48, AVR32DA48T	95
ATxmega64A1	94	AVR32DA48S	95
ATxmega64A1U	94	AVR32DB28, AVR32DB28T	95
ATxmega64A3	94	AVR32DB32, AVR32DB32T	95
ATxmega64A3U	94	AVR32DB48, AVR32DB48T	95
ATxmega64A4	94	AVR32DD14	95
ATxmega64A4U	94	AVR32DD20	95
ATxmega64B1	94	AVR32DD28	95
ATxmega64B3	94	AVR32DD32	95
ATxmega64C3	94	AVR32DU14	95
ATxmega64D3	94	AVR32DU20	95
ATxmega64D4	94	AVR32DU28	95
ATxmega8E5	94	AVR32DU32	95
Auto-detect mode	14	AVR32EA28	95
Auto-erase	9	AVR32EA32	95
Autogenerated files	61	AVR32EA48	95
avr109	83	AVR32EB14	95
avr910	83	AVR32EB20	95
avr911	83	AVR32EB28	95
AVR as programmer	83	AVR32EB32	95
AVR Dragon	3, 18, 84	AVR32JTAG	55
AVR ISP programmer	84	AVR32LA14, AVR32LA14T	95
AVR JTAGICE3	2, 18, 84	AVR32LA20, AVR32LA20T	95
AVR128DA28, AVR128DA28T	94	AVR32LA28, AVR32LA28T	95
AVR128DA28S	94	AVR32LA32, AVR32LA32T	95
AVR128DA32, AVR128DA32T	94	AVR32SD20	95
AVR128DA32S	94	AVR32SD28	96
AVR128DA48, AVR128DA48T	94	AVR32SD32	96
AVR128DA48S	94	AVR64DA28, AVR64DA28T	96
AVR128DA64, AVR128DA64T	94	AVR64DA28S	96
AVR128DA64S	94	AVR64DA32, AVR64DA32T	96
AVR128DB28, AVR128DB28T	94	AVR64DA32S	96
AVR128DB32, AVR128DB32T	94	AVR64DA48, AVR64DA48T	96
AVR128DB48, AVR128DB48T	94	AVR64DA48S	96
AVR128DB64, AVR128DB64T	94	AVR64DA64, AVR64DA64T	96
AVR16DD14	94	AVR64DA64S	96
AVR16DD20	95	AVR64DB28, AVR64DB28T	96
AVR16DD28	95	AVR64DB32, AVR64DB32T	96
AVR16DD32	95	AVR64DB48, AVR64DB48T	96
AVR16DU14	95	AVR64DB64, AVR64DB64T	96
AVR16DU20	95	AVR64DD14	96
AVR16DU28	95	AVR64DD20	96
AVR16DU32	95	AVR64DD28	96
AVR16EA28	95	AVR64DD32	96
AVR16EA32	95	AVR64DU28	96
AVR16EA48	95	AVR64DU32	96
AVR16EB14	95	AVR64EA28	96
AVR16EB20	95	AVR64EA32	96
AVR16EB28	95	AVR64EA48	96
AVR16EB32	95	AVR8EA28	96

AVR8EA32	96
avrdude.conf	53
AVRDUDE defaults	53
avrftdi	83
avrisp	83
avrisp-u	83
avrisp2	83
avrispmkII	83
avrispv2	83
aWire	55

B

backup <i>memlist file[:format]</i>	41
bascom	83
Bascom SAMPLE cable	83
benign code	17
Binary file mode	14
BitWizard ftdi-atmega	83
blaster	83
bodcfg	98
boot	97
bootend	99
Bootloader (AVR109, AVR911)	20, 83
Bootloader (Butterfly)	83
bootrow	41, 42, 99
bootsize	99
Brian S. Dean's programmer	83
bsd	83
BusPirate	24, 83
buspirate	83
buspirate_bb	83
butterfly	2, 83
butterfly_mk	83
bwmega	83

C

c128	88
c232hm	83
c2n232i	84
c32	88
c64	88
C232HM cable from FTDI	83
calibration	1, 10, 13, 97, 98, 99
ch341a	84
CH341A programmer	3, 84
codesize	99
Command line options	6
config <i>[-fl-a -v]</i>	42
config <i>[-fl-v] property [-fl-v]</i>	42
config <i>[-fl-v] property= [-fl-v]</i>	42
config <i>[-fl-v -c] property=value [-fl-v -c]</i>	42
Configuration files	7, 53, 74
Crossbow MIB510	85
Curiosity nano	4, 20, 86

D

d (dump)	44
dapa	84
dasa	84
dasa3	84
debugWIRE	2, 55
Decimal file mode	14
default_bitclock	53
default_linuxgpio	53
default_parallel	53
default_programmer	53
default_serial	53
DFU Bootloader using FLIP version 1	71
Dick Smith HOTCHIP	83
diecimila	83
Digilent JTAG HS2	84
digilent-hs2	84
Direct AVR Parallel cable	84
disasm <i>[memory [addr [len]]]</i>	37
disasm example	51
Dontronics DT006	84
Dragon	3, 18
dragon_dw	84
dragon_hvsp	84
dragon_isp	84
dragon_jtag	84
dragon_pdi	84
dragon_pp	84
dry: test files	61
dryboot	1, 68, 84
dryrun	1, 68, 84
dt006	84
Dual boot	68
dump <i>[memory [addr [len]]]</i>	37
dump <i>memory [addr]</i>	37

E

eeeprom ... 1, 8, 9, 10, 13, 21, 22, 29, 41, 42, 43, 71, 72, 78, 97, 98, 99	
ehajo-isp	84
ELF (Executable and Linkable Format)	14
Emulating a bootloader (dryboot)	1, 17, 68, 84
Emulating a HW programmer (dryrun) .	1, 17, 68, 84
erase	41
erase <i>memory [addr len]</i>	41
ere-isp-avr	84
ERE ISP-AVR	84
Example Command line invocations	28

F

factory reset	43
flash ... 1, 2, 4, 8, 9, 13, 14, 15, 17, 21, 22, 23, 25, 28, 29, 30, 39, 41, 51, 60, 71, 97, 98	
Flashcom serprog protocol	27, 86
flip1	84
flip2	84
FLIP bootloader	4, 16
FLIP bootloader DFU v1 (doc7618)	84
FLIP bootloader DFU v2 (AVR4023)	84
flush	42
flyswatter2	84
fosc freq[M k]	45
fosc off	45
FOSC_ADJ	55
Frank STK200	84
frank-stk200	84
FreeBSD configuration files	74
FreeBSD USB permissions	75
ft2232h	84
ft2232h_jtag	84
ft232h	84
ft232h_jtag	84
ft232r	84
ft245r	84
ft4232h	84
FT2232H JTAG programmer	2, 84
FT2232H with buffer and LEDs	2, 83
FT2232H/D programmer	2, 83, 84
FT232H JTAG programmer	2, 84
FT232H programmer	2, 84
FT232R programmer	2, 84
FT232R Synchronous BitBang	2
FT245R programmer	2, 84
FT4232H programmer	2, 83, 84
FTDI TTL232R-5V	2, 87
fuse0	98
fuse1	98
fuse6	98
fuses	98, 99
futurlec	84
Futurlec.com cable	84

H

HAS_FOSC_ADJ	55
HAS_SUFFER	55
HAS_VAREF_ADJ	55
HAS_VTARG_ADJ	55
HAS_VTARG_READ	55
HAS_VTARG_SWITCH	55
Hexadecimal file mode	14
History and credits	4
HVPP	55
HVSP	55
hwmoncfg	98

I

Immediate file mode	14
include [opts] file	43
Installation	74, 76
Instruction format	59
Intel Hex	13
Introduction	1
io	98, 99
iseavrprog	84
ISP	2, 55

J

Jason Kyle's pAVR	85
jtag1	85
jtag1slow	84
jtag2	84
jtag2avr32	85
jtag2dw	84
jtag2fast	84
jtag2isp	84
jtag2pdi	84
jtag2slow	84
jtag2updi	84
jtag3	84
jtag3dw	84
jtag3isp	84
jtag3pdi	84
jtag3updi	84
JTAG ICE mkI	2, 84, 85
JTAG ICE mkII	2, 18, 84, 85
JTAGICE3	2, 18
jtagkey	84
jtagmkI	85
jtagmkII	85
jtagmkII_avr32	85
JTAG	2, 55
JTAGmkI	55
JTAGv2 to UPDI bridge	27, 84

K

Kanda AVRISP-U	83
KT-LINK FT2232H	85
ktlink	85

L

Lancos SI-Prog	86
LED management	73
lgt168p	96
lgt328p	96
lgt88p	96
LGT8F168P	96
LGT8F328P	96
LGT8F88P	96
Linux /dev/spidev* programmer	27, 85
Linux configuration files	74
Linux USB permissions	75
linuxspi	85
lm3s811	85
Low-cost programmer AVR910	21, 83
Luminary Micro LM3S811	85

M

m103	88
m128	88
m1280	88
m1281	89
m1284	89
m1284p	89
m1284rfr2	89
m128a	89
m128rfal	89
m128rfr2	89
m16	89
m1608	89
m1609	89
m161	89
m162	89
m163	89
m164a	89
m164p	89
m164pa	89
m165	89
m165a	89
m165p	89
m165pa	89
m168	89
m168a	89
m168p	89
m168pa	89
m168pb	89
m169	89
m169a	89
m169p	89
m169pa	89
m16a	89
m16hva	89
m16hvb	89
m16hvbrevb	89
m16m1	89
m16u2	89
m16u4	89

m2560	89
m2561	89
m2564rfr2	89
m256rfr2	89
m32	89
m3208	89
m3209	89
m324a	90
m324p	90
m324pa	90
m324pb	90
m325	90
m3250	90
m3250a	90
m3250p	90
m3250pa	90
m325a	90
m325p	90
m325pa	90
m328	90
m328p	90
m328pb	90
m329	90
m3290	90
m3290a	90
m3290p	90
m3290pa	90
m329a	90
m329p	90
m329pa	90
m32a	90
m32c1	90
m32hvb	90
m32hvbrevb	90
m32hve2	90
m32m1	90
m32u2	90
m32u4	90
m406	90
m48	90
m4808	90
m4809	90
m48a	90
m48p	90
m48pa	90
m48pb	90
m64	90
m640	90
m644	90
m644a	90
m644p	90
m644pa	90
m644rfr2	91
m645	91
m6450	91
m6450a	91
m6450p	91
m645a	91

m645p.....	91
m649.....	91
m6490.....	91
m6490a.....	91
m6490p.....	91
m649a.....	91
m649p.....	91
m64a.....	91
m64c1.....	91
m64hve2.....	91
m64m1.....	91
m64rfr2.....	91
m8.....	91
m808.....	91
m809.....	91
m8515.....	91
m8535.....	91
m88.....	91
m88a.....	91
m88p.....	91
m88pa.....	91
m88pb.....	91
m8a.....	91
m8hva.....	91
m8u2.....	91
Memories of ATxmegas.....	97
Memories of classic parts.....	97
Memories of modern AVR parts.....	98
Metadata.....	23
mib510.....	85
Microchip PICkit 2 programmer.....	26, 85
micronucleus.....	85
Micronucleus bootloader.....	4, 26, 85
Mikrokoetter.de Butterfly.....	83
mkbutterfly.....	83
Motorola S-Record.....	14
MPLAB(R) PICkit 4.....	4, 18, 85
MPLAB(R) PICkit 5.....	4, 18, 85
MPLAB(R) PICkit Basic.....	4, 18, 85, 86
MPLAB(R) SNAP.....	4, 18, 86
ms128.....	91
ms64m1.....	91

N

nanoevery.....	84
nibobee.....	85
NIBObree.....	85
Nightshade ALF-PgmAVR.....	83

O

o-link.....	85
O-Link, OpenJTAG ARM JTAG USB.....	85
Octal file mode.....	14
openmoko.....	85
Openmoko debug board.....	85
Option --baud <i>baudrate</i>	6
Option --bitclock <i>bitclock</i>	6
Option --command <i>cmd</i>	12
Option --config <i>config-file</i>	7
Option --erase.....	8
Option --exitspecs <i>exitspec</i> [,...].....	9
Option --extended <i>parameter</i>	15
Option --force.....	9
Option --help.....	15
Option --isp-clock-delay <i>delay</i>	9
Option --keep-trailing-0xff.....	8
Option --logfile <i>logfile</i>	10
Option --memory <i>mem:op:file[:fmt]</i>	12
Option --noconfig.....	8
Option --noerase.....	8
Option --noverify-memory.....	15
Option --osccal.....	10
Option --part <i>partname</i>	6
Option --part <i>wildcard/flags</i>	6
Option --port <i>port</i>	10
Option --programmer <i>programmer-id</i>	7
Option --programmer <i>wildcard/flags</i>	7
Option --quell.....	12
Option --reconnect.....	12
Option --terminal.....	12
Option --test-memory.....	10
Option --verbose.....	15
Option --version.....	15
Option -A.....	8
Option -b <i>baudrate</i>	6
Option -B <i>bitclock</i>	6
Option -c <i>config-file</i>	7
Option -c <i>programmer-id</i>	7
Option -c <i>wildcard/flags</i>	7
Option -D.....	8
Option -e.....	8
Option -E d_high.....	16
Option -E d_low.....	16
Option -E <i>exitspec</i> [,...].....	9
Option -E noreset.....	16
Option -E novcc.....	16
Option -E reset.....	16
Option -E vcc.....	16
Option -F.....	9
Option -h.....	15
Option -i <i>delay</i>	9
Option -l <i>logfile</i>	10
Option -n.....	10
Option -N.....	8
Option -O.....	10
Option -p <i>partname</i>	6
Option -p <i>wildcard/flags</i>	6

Option -P <i>port</i>	10
Option -q	12
Option -r	12
Option -t	12
Option -T <i>cmd</i>	12
Option -U <i>mem:op:file[:fmt]</i>	12
Option -v	15
Option -V	15
Option -x Arduino	21
Option -x Atmel-ICE	18
Option -x AVR Dragon	18
Option -x AVR109	20
Option -x AVR109	21
Option -x BusPirate	24
Option -x Curiosity Nano	20
Option -x dryboot	17
Option -x dryrun	17
Option -x flip2	16
Option -x jtag2updi	27
Option -x JTAG ICE mkII/3	18
Option -x linuxgpio	16
Option -x linuxspi	16, 27
Option -x Micronucleus bootloader	26
Option -x MPLAB(R) SNAP	18
Option -x parallel port programmers	16
Option -x <i>parameter</i>	15
Option -x PICkit 4	18
Option -x PICkit 5	18
Option -x PICkit2	26
Option -x pickit4_mplab, pickit5	16
Option -x Power Debugger	18
Option -x raspberry_pi_gpio	16
Option -x serialupdi	27
Option -x serprog	27
Option -x STK500	20
Option -x STK600	20
Option -x Teensy bootloader	26
Option -x Urclock	21
Option -x USBasp	26
Option -x Wiring	26
Option -x xbee	27
Option -x Xplained Mini	19
Option descriptions	6
Options (command-line)	6
osc16err	99
osc20err	99
osccal16	99
osccal20	99
osccfg	98
Other notes	59

P

Parent part	58
parms	45
part	46, 56
part [<i>opts</i>]	44
Part definitions	56
Part specification	6
Patching the vector table	68
pavr	85
pdicfg	99
PDI	2, 55
pgerase <i>memory addr</i>	44
pgm	44
PICkit 2 programmer	26, 85
PICkit 4	4, 18, 85
PICkit 5	4, 18, 85
PICkit Basic	4, 18, 85, 86
pickit_basic	85
pickit_basic_dw	85
pickit_basic_isp	85
pickit_basic_jtag	85
pickit_basic_mplab	85
pickit_basic_mplab_dw	85
pickit_basic_mplab_isp	85
pickit_basic_mplab_jtag	85
pickit_basic_mplab_pdi	86
pickit_basic_mplab_tpi	86
pickit_basic_mplab_updi	86
pickit_basic_pdi	86
pickit_basic_tpi	86
pickit_basic_updi	86
pickit2	85
pickit4	85
pickit4_dw	85
pickit4_isp	85
pickit4_jtag	85
pickit4_mplab	85
pickit4_mplab_dw	85
pickit4_mplab_isp	85
pickit4_mplab_jtag	85
pickit4_mplab_pdi	85
pickit4_mplab_tpi	85
pickit4_mplab_updi	85
pickit4_pdi	85
pickit4_tpi	85
pickit4_updi	85
pickit5	85
pickit5_dw	85
pickit5_isp	85
pickit5_jtag	85
pickit5_pdi	85
pickit5_tpi	85
pickit5_updi	85
picoweb	86
Picoweb Programming Cable	86
pincfg	98
pkobn_updi	86
PM_AVR32JTAG	55

PM_aWire	55
PM_debugWIRE	55
PM_HVPP	55
PM_HVSP	55
PM_ISP	55
PM_JTAG	55
PM_JTAGmkI	55
PM_PDI	55
PM_SPM	54
PM_TPI	54
PM_UPDI	55
PM_XMEGAJTAG	55
Pony Prog STK200	86
pony-stk200	86
ponyser	86
Port specification	10
PowerDebugger	18, 86
powerdebugger	86
powerdebugger_dw	86
powerdebugger_isp	86
powerdebugger_jtag	86
powerdebugger_pdi	86
powerdebugger_tpi	86
powerdebugger_updi	86
prodsig	98, 99
programmer	53
Programmer definitions	54
Programmer LED management	73
Programmer specification	7
Programmer-specific information	69
Programmers accepting exitspec parameters	16
Programmers accepting extended parameters	17
Programmers supported	1, 83
Programming mode	44
Programming modes	54
pwm1	88
pwm161	88
pwm2	88
pwm216	88
pwm2b	88
pwm3	88
pwm316	88
pwm3b	88
pwm81	88

Q

q (quit)	44
quell [level]	44
quit	44

R

r (read)	44
raspberrypi_gpio	86
Raw binary	14
read [memory [addr [len]]]	37
read memory [addr]	37
regfile [opts]	43
regfile [opts] reg [opts]	43
regfile [opts] reg=value [opts]	43
restore memlist file[:format]	41
RPi GPIO programmer	86

S

save memory {addr len} file[:format]	41
sck period	45
scratchmonkey	87
scratchmonkey_hvsp	86
scratchmonkey_pp	86
send b1 b2 b3 b4	44
Serial adapter definitions	55
Serial Atmel AVR ISP	83
Serial Atmel AVR ISPv2	83
Serial port programmer	84, 86
serialadapter	55
serialupdi	86
SerialUPDI	4, 27, 71, 86
SerialUPDI programmer	71
sernum	98, 99
serprog	86
sib	99
signature	1, 9, 13, 43, 72, 97, 98, 99
sigrow	98, 99
siprog	86
snap	86
snap_dw	86
snap_isp	86
snap_jtag	86
snap_mplab	86
snap_mplab_dw	86
snap_mplab_isp	86
snap_mplab_jtag	86
snap_mplab_pdi	86
snap_mplab_tpi	86
snap_mplab_updi	86
snap_pdi	86
snap_tpi	86
snap_updi	86
SNAP	4, 18, 86
sp12	86
spi	44
SPM	54
sram	98, 99
Steve Bolt's Programmer	86
stk200	86
stk500	86
stk500hvsp	86
stk500pp	86

stk500v1	87
stk500v2	87
stk600	87
stk600hvsp	87
stk600pp	87
STK200 starter kit	86
STK500	2, 20, 86
STK500 v1	87
STK500 v2	86, 87
STK600	2, 20, 69, 87
SUFFER	55
syscfg0	98
syscfg1	99

T

t10	91
t102	91
t104	91
t11	91
t12	91
t13	91
t13a	91
t15	91
t1604	91
t1606	91
t1607	91
t1614	91
t1614auto	91
t1616	91
t1616auto	92
t1617	92
t1617auto	92
t1624	92
t1626	92
t1627	92
t1634	92
t1634r	92
t167	92
t20	92
t202	92
t204	92
t212	92
t214	92
t22	92
t2313	92
t2313a	92
t24	92
t24a	92
t25	92
t26	92
t261	92
t261a	92
t28	92
t3216	92
t3216auto	92
t3217	92
t3217auto	92

t3224	92
t3226	92
t3227	92
t4	92
t40	92
t402	92
t404	92
t406	92
t412	92
t414	92
t416	92
t416auto	92
t417	92
t417auto	92
t424	92
t426	92
t427	92
t4313	92
t43u	92
t44	93
t441	93
t44a	93
t45	93
t461	93
t461a	93
t48	93
t5	93
t804	93
t806	93
t807	93
t814	93
t814auto	93
t816	93
t816auto	93
t817	93
t817auto	93
t824	93
t826	93
t827	93
t828	93
t828r	93
t84	93
t841	93
t84a	93
t85	93
t861	93
t861a	93
t87	93
t88	93
t9	93
Tag-Connect TC2030	87
Tagfile	38
tc2030	87
tcd0cfg	98
teensy	87
Teensy bootloader	4, 26, 87
tempsense	98, 99
Terminal mode commands	36

Terminal mode examples 46
 Terminal mode operation 36
 The Bus Pirate 3, 24, 83
 TIAO USB programmer 87
tigard 87
 Tigard interface board 87
 TinCan Tools Flyswatter 2 84
 TPI 54
 Trinket Gemma bootloader 83
ttl232r 87
tumpa 87
tumpa-b 87
tumpa_jtag 87

U

uc3a0512 88
um232h 87
 UM232H module from FTDI 87
uncompatino 87
 Uncompatino programmer 87
 Unix 74
 Unix configuration files 74
 Unix documentation 76
 Unix installation 74
 Unix port names 74
 Unix USB permissions 74
 UPDI 55
 Urboot bootloader 3, 21, 87
urboot: bootloader files 63
urboot: 61
 urclock 68, 87
 Urclock programmer 3, 21, 87
 Urprotocol 3, 21, 87
usb1286 88
usb1287 88
usb162 88
usb646 88
usb647 88
usb82 88
 USB Atmel AVR ISP mkII 83
 USB permissions 74, 75, 76
usbasp 87
 Usbasp clones 87
 USBasp ISP and TPI programmer 2, 26, 87
usbasp-clone 87
usbtiny 87
 USBtiny simple USB programmer 2, 87
userrow 99
usersig 42, 98

V

varef [*channel*] *voltage* 44
 VAREF_ADJ 55
 Variants of parts 44
 Vector table 68
verbose [*level*] 44
verify memlist file[:format] 41
vtarg voltage 44
 VTARG_ADJ 55
 VTARG_READ 55
 VTARG_SWITCH 55

W

w (write) 44
wdtcfg 98
 Windows 76
 Windows configuration file location 77
 Windows configuration file names 77
 Windows configuration files 76
 Windows parallel ports 77
 Windows port names 77
 Windows serial ports 77
wiring 87
 Wiring bootloader 3, 26, 87
write memory addr data[,] {*data[,]*} 39
write memory addr len data[,] {*data[,]*} 41
write memory data 40
write memory file 40

X

x128a1 93
x128a1d 93
x128a1u 93
x128a3 93
x128a3u 93
x128a4 93
x128a4u 93
x128b1 93
x128b3 93
x128c3 93
x128d3 93
x128d4 93
x16a4 93
x16a4u 93
x16c4 93
x16d4 93
x16e5 94
x192a1 94
x192a3 94
x192a3u 94
x192c3 94
x192d3 94
x256a1 94
x256a3 94
x256a3b 94
x256a3bu 94

x256a3u	94	x64c3	94
x256c3	94	x64d3	94
x256d3	94	x64d4	94
x32a4	94	x8e5	94
x32a4u	94	xbee	87
x32c3	94	XBeeBoot OTA bootloader	27, 87
x32c4	94	xil	87
x32d3	94	Xilinx JTAG cable	87
x32d4	94	XMEGAJTAG	55
x32e5	94	xplainedmini	87
x384c3	94	XplainedMini	3, 19, 87
x384d3	94	xplainedmini_dw	87
x64a1	94	xplainedmini_isp	87
x64a1u	94	xplainedmini_tpi	87
x64a3	94	xplainedmini_updi	87
x64a3u	94	xplainedpro	87
x64a4	94	XplainedPro	3, 87
x64a4u	94	xplainedpro_jtag	87
x64b1	94	xplainedpro_pdi	87
x64b3	94	xplainedpro_updi	87