AVRDUDE

A program for downloading/uploading AVR microcontroller flash, EEPROM and more
for AVRDUDE, Version 8.2, 22 July 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 56
4.4 Part Definitions 57
4.4.1 Parent Part 59
4.4.2 Instruction Format 59
4.5 Other Notes 60

5 Autogenerated files 62

6 Programmer-Specific Information 68
6.1 Atmel STK600 68
6.2 DFU Bootloader Using FLIP Version 1 70
6.3 SerialUPDI Programmer 70
6.4 Programmer LED Management 72

Appendix A Platform Dependent Information . 73
A.1 Unix 73
A.1.1 Unix Installation 73
A.1.2 Unix Configuration Files 73
A.1.2.1 FreeBSD Configuration Files 73
A.1.2.2 Linux Configuration Files 73
A.1.3 Unix Port Names 73
A.1.4 Unix USB Permissions 73
A.1.4.1 FreeBSD USB Permissions 74
A.1.4.2 Linux USB Permissions 74

A.1.5 Unix Documentation ... 75
A.2 Windows ... 75
A.2.1 Installation ... 75
A.2.2 Windows Configuration Files ... 75
A.2.2.1 Windows Configuration File Names ... 76
A.2.2.2 Windows Configuration File Location ... 76
A.2.3 Windows Port Names ... 76
A.2.3.1 Windows Serial Ports ... 76
A.2.3.2 Windows Parallel Ports ... 76

Appendix B Troubleshooting ... 77

Appendix C List of Programmers ... 82

Appendix D List of Parts ... 87

Appendix E List of Memories ... 96
E.1 Classic parts ... 96
E.2 ATxmegas ... 96
E.3 Modern AVR Parts ... 97

Concept Index ... 99

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 392 parts and 203 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 defaults to `/dev/spidev0.0:/dev/gpiochip0` for this programmer. 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 AVRDUDE) *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 AVRDUDE 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 70, 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 82, 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 87, 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 bootloader. See Appendix C [List of Programmers], page 82, 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 normally specifies the connection type; in absence of a `-P` specification, system-dependent default values `default_serial`, `default_spi`, `default_linuxgpio` or `default_parallel` from the configuration file are used. If you need to use a different port, use this option to specify the alternate port name.

  USB-only programmers normally do not need the port option be specified 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 into the host computer is the `-P` option needed, see below. Some `-c` programmers, e.g., pickit5, ignore the `-P` option altogether; these cannot distinguish multiple plugged-in programmers.

  Most USB programmers, however, support the command-line syntax `-P 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's 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) ist 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 also 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 77.

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,eeprom`; 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 96, 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 *file* 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 *0x* prepended.

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

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

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 Chapter 5 [Autogenerated Files], page 62, 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.

`-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`, `raspberry_pi_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`
`raspberry_pi_gpio`
`Parallel port programmers`

`help` Show help menu and exit.

`reset` The '/RESET' signal will be left activated at program exit, that is it will be held low, in order to keep the MCU in reset state afterwards. Note in particular that the programming algorithm for the AT90S1200 device mandates that the '/RESET' signal is active before powering up the MCU, so in case an external power supply is used for this MCU type, a previous invocation of AVRDUDE with this option specified is one of the possible ways to guarantee this condition. `flip2` will not exit bootloader mode at program exit if `reset` is used.

`noreset` The '/RESET' line will be deactivated at program exit, thus allowing the MCU target program to run while the programming hardware remains connected. `flip2` will exit bootloader mode at program exit and start the application if `noreset` is used, and this is the default behaviour for this bootloader.

`Parallel port programmers`

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

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

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

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

`Pickit 4 (MPLAB)`
`Pickit 5`

`vcc` 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. They accept the following parameters:

`init` Initialise memories with human-readable patterns. Flash memory will be randomly configured with respect to bootloader, data and code length. Patterns can best be seen with fixed-width font and the `:I` format by inspecting the generated hex file or by using, eg, `-U flash:r:-:I`. Patterns in flash memory are executable and represent benign AVR code, ie, no I/O memory access. Choose a fixed seed for reproducible results.

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

`random` Initialise memories with random code and values. Flash memory will be randomly configured with respect to bootloader, data and code length. Random code in flash will be benign, that is, not accessing I/O memories, SRAM or flash. Choose a fixed seed for reproducible results.

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

`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:`

Agressive 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, *string* 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.

`eepromrw` 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`
: Urclock 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 `DIO`*1..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 `DIO3` 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 `eeprom.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:eeprom.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:eeprom.hex:i
Reading 3328 bytes for eeprom from input file eeprom.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 ff  |Bonjour.........|
0010  ff ff ff ff ff ff ff ff  78 56 34 12 ff 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 -cusbasp -pattiny13 -Ueeprom:r:-:i

:20000000E2809954686520717569636B2062726F776E20666F78E280990AFFFFFFFFFFFFD3
:20002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
:00000001FF
```

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

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

:20000000E2809954686520717569636B2062726F776E20666F78E280990AFFFFFFFFFFFFD3
:20002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
: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 curently 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]
  ft2232h = [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 discribe 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*:

addr	*len*	**Memory interval**	**Comment**
0/positive	positive	[*addr*, *addr*+*len*-1]	Note: *len* = *end*-*start* + 1
0/positive	negative	[*addr*, *sz*+*len*]	End is |*len*| bytes below mem size *sz*
negative	positive	[*sz*+*addr*, *sz*+*addr*+*len*-1]	Start is |*addr*| bytes below mem size
negative	negative	[*sz*+*addr*, *sz*+*len*]	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:

addr	*len*	**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 = |*len*+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 under standing 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 62, 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).

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 `0xFFFFffffFFFFffffU`.

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 intervall 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 explictly 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 ensuinig 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 bootsz=bs_2048w # 0
config eesave=ee_erased # 1
config wdton=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 bootsz=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.rxdatah  # Receive data high byte
I/O 0x802: (1)   0x00 usart0.txdatal  # 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 ellipis ...** 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 ff  |................|
7fd0  ff 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 ff  |................|
7ff0  ff 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 05  |................|
7ff0  05 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.
```

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

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

Label3:                            ; Brne from L01c, L020
L01a:  sbiw    r30, 0x01           ; 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 bootloader, 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 73, 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 `avrddude.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
    baudrate = <num>;                           # baudrate for avr910-programmer
    vcc      = <pin1> [, <pin2> ... ];          # pin number(s) (3)
    buff     = <pin1> [, <pin2> ... ];          # pin number(s)
    reset    = <pin>;                           # pin number
    sck      = <pin>;                           # pin number
    sdo|pico = <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 new setting them for the new programmer.

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 AVRs
- `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`: Programer 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:

`arduino`	Arduino programmer for bootloading
`avr910`	Serial programmer using protocol from appnote AVR910
`avrftdi`	Interface to the MPSSE Engine of FTDI chips using libftdi
`avrftdi_jtag`	libftdi JTAG interface
`buspirate`	Bus Pirate's SPI interface
`buspirate_bb`	Bus Pirate's bitbang interface
`butterfly`	Atmel Butterfly evaluation board (AVR109, AVR911)
`butterfly_mk`	Mikrokopter.de Butterfly
`ch341a`	Chip CH341A: AVR must have min F_CPU of 6.8 MHz
`dryrun`	Dryrun programmer for testing avrdude
`dragon_dw`	Atmel AVR Dragon in debugWire mode
`dragon_hvsp`	Atmel AVR Dragon in HVSP mode
`dragon_isp`	Atmel AVR Dragon in ISP mode
`dragon_jtag`	Atmel AVR Dragon in JTAG mode
`dragon_pdi`	Atmel AVR Dragon in PDI mode
`dragon_pp`	Atmel AVR Dragon in PP mode
`flip1`	FLIP USB DFU protocol version 1 (doc7618)
`flip2`	FLIP USB DFU protocol version 2 (AVR4023)
`ftdi_syncbb`	FT245R/FT232R synchronous bit-bang programmer
`jtagmki`	Atmel JTAG ICE mkI
`jtagmkii`	Atmel JTAG ICE mkII
`jtagmkii_avr32`	Atmel JTAG ICE mkII in AVR32 mode
`jtagmkii_dw`	Atmel JTAG ICE mkII in debugWire mode
`jtagmkii_isp`	Atmel JTAG ICE mkII in ISP mode

`jtagmkii_pdi`	Atmel JTAG ICE mkII in PDI mode
`jtagmkii_updi`	Atmel JTAG ICE mkII in UPDI mode
`jtagice3`	Atmel JTAGICE3
`jtagice3_pdi`	Atmel JTAGICE3 in PDI mode
`jtagice3_updi`	Atmel JTAGICE3 in UPDI mode
`jtagice3_dw`	Atmel JTAGICE3 in debugWire mode
`jtagice3_isp`	Atmel JTAGICE3 in ISP mode
`jtagice3_tpi`	Atmel JTAGICE3 in TPI mode
`linuxgpio`	GPIO bitbanging using the Linux libgpiod or sysfs interface
`linuxspi`	SPI using Linux spidev driver
`micronucleus`	Micronucleus Bootloader
`par`	Parallel port bitbanging
`pickit2`	Microchip's PICkit2 Programmer
`pickit5`	Microchip's PICkit 5 Programmer/Debugger
`serbb`	Serial port bitbanging
`serialupdi`	Driver for SerialUPDI programmers
`serprog`	Program via the Serprog protocol from Flashrom
`stk500`	Atmel STK500 Version 1.x firmware
`stk500generic`	Atmel STK500, autodetect firmware version
`stk500v2`	Atmel STK500 Version 2.x firmware
`stk500hvsp`	Atmel STK500 V2 in high-voltage serial programming mode
`stk500pp`	Atmel STK500 V2 in parallel programming mode
`stk600`	Atmel STK600
`stk600hvsp`	Atmel STK600 in high-voltage serial programming mode
`stk600pp`	Atmel STK600 in parallel programming mode
`teensy`	Teensy Bootloader
`urclock`	Urclock programmer for urboot bootloaders (arduino compatible)
`usbasp`	USBasp programmer, see `http://www.fischl.de/usbasp/`
`usbtiny`	Usbtiny programmer (also as bootloading protocol)
`wiring`	Bootloader using the STK500v2 protocol (AVR068)
`xbee`	XBee Series 2 Over-The-Air (XBeeBoot)

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 AVRs
    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>, ...;
    hventerstabdelay = <num>;
```

```
progmodedelay    = <num>;                # PP only
latchcycles      = <num>;
togglevtg        = <num>;
poweroffdelay    = <num>;
resetdelayms     = <num>;
resetdelayus     = <num>;
hvleavestabdelay = <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_fcpu     = <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)
    pwroff_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`, `hvupdi_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 arithemtic and bitwise operators.

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

`a`*N* The bit is the *N*th 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, the only autogenerated files are urboot bootloaders from the `https://github.com/stefanrueger/urboot` project. They take the form `urboot:`*features*. Bootloader features are specified using an underscore-separated 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.

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

```
$ avrdude -qvc dryrun -p m328p -U urboot:i8mhz_56kbaud_uart0_best |& grep -i baud.rat
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.rat
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:                           ; Urboot marker bytes
L0002:  .byte   0x75, 0x72          ; ur

L0004:  .fill   16190, 2, 0xffff

; Rjmp from L0000
urboot:                             ; Entry point for bootloader
L7e80:  clr     r1
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:                        ; Serial port bootloading
L7e8e:  ldi     r24, 0x0e           ; 14
L7e90:  rcall   set_watchdog        ; L7f5c

[...]

Label27:
L7fea:  clr     r1
L7fec:  ret

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

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

L7ffc:  rjmp    pgm_write_page      ; L7f78

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

It is noteworthy that `-c dryboot` allows uploading of one urboot bootloader and, from that point onwards, emulates `-c urclock -x nometadata`. 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-uploaded sketch if the previously installed bootloader is an urboot vector bootloader just like `avrdude -c urclock -x nometadata` 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 upload through a self-written 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-SOIC	pwm2 pwm3 pwm2b pwm3b pwm216 pwm316
STK600-RCPWM-26	STK600-SOIC	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-ATXMEGAT0	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 AVRs 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 `AREF0` 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.

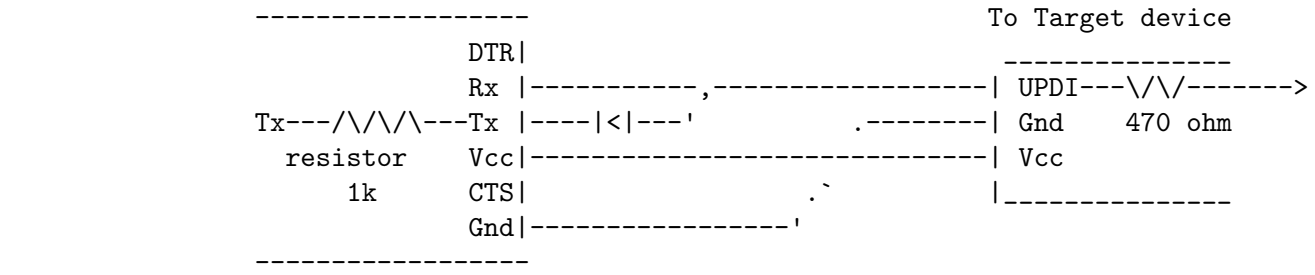
Sometimes, a bootloader with security mode already disabled seems to no longer respond with sensible configuration data, but only 0xFF for all queries. As these queries are used to obtain the equivalent of a signature, AVRDUDE can only continue in that situation by forcing the signature check to be overridden 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    ".avrdx"
    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	`/dev/ppi0`	`/dev/cuad0`
Linux	`/dev/parport0`	`/dev/ttyS0`
Solaris	`/dev/printers/0`	`/dev/term/a`

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 0x0eb) 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 0x0eb):

```
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 paticular, 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 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 subfamily.

- 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`	Mikrokopter.de Butterfly bootloader (SPM)
`mkbutterfly`	Mikrokopter.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_jtag	MPLAB(R) PICkit Basic (JTAG, XMEGAJTAG)
pickit_basic_dw	MPLAB(R) PICkit Basic (debugWIRE)
pickit_basic_ mplab_dw	MPLAB(R) PICkit Basic (debugWIRE)
pickit_basic_isp	MPLAB(R) PICkit Basic (ISP)

`pickit_basic_mplab_isp`	MPLAB(R) PICkit Basic (ISP)
`pickit_basic_pdi`	MPLAB(R) PICkit Basic (PDI)
`pickit_basic_mplab_pdi`	MPLAB(R) PICkit Basic (PDI)
`pickit_basic_tpi`	MPLAB(R) PICkit Basic (TPI)
`pickit_basic_mplab_tpi`	MPLAB(R) PICkit Basic (TPI)
`pickit_basic_updi`	MPLAB(R) PICkit Basic (UPDI)
`pickit_basic_mplab_updi`	MPLAB(R) PICkit Basic (UPDI)
`picoweb`	Picoweb Programming Cable (TPI, ISP)
`pkobn_updi`	Curiosity nano (nEDBG) (UPDI)
`pony-stk200`	Pony Prog STK200 (TPI, ISP)
`ponyser`	ponyprog serial: reset=!txd sck=rts sdo=dtr sdi=cts (TPI, ISP)
`powerdebugger`	Atmel PowerDebugger (JTAG, XMEGAJTAG, AVR32JTAG)
`powerdebugger_jtag`	Atmel PowerDebugger (JTAG, XMEGAJTAG, AVR32JTAG)
`powerdebugger_dw`	Atmel PowerDebugger (debugWIRE)
`powerdebugger_isp`	Atmel PowerDebugger (ISP)
`powerdebugger_pdi`	Atmel PowerDebugger (PDI)
`powerdebugger_tpi`	Atmel PowerDebugger (TPI)
`powerdebugger_updi`	Atmel PowerDebugger (UPDI)
`raspberry_pi_gpio`	Raspberry Pi GPIO via sysfs/libgpiod (ISP)
`serialupdi`	SerialUPDI (UPDI)
`serprog`	Program via the Serprog protocol from Flashrom (ISP)
`siprog`	Lancos SI-Prog (same as ponyser) (TPI, ISP)
`snap`	MPLAB(R) SNAP (JTAG, XMEGAJTAG)
`snap_jtag`	MPLAB(R) SNAP (JTAG, XMEGAJTAG)
`snap_dw`	MPLAB(R) SNAP (debugWIRE)
`snap_isp`	MPLAB(R) SNAP (ISP)
`snap_mplab`	MPLAB(R) SNAP (JTAG, XMEGAJTAG)
`snap_mplab_jtag`	MPLAB(R) SNAP (JTAG, XMEGAJTAG)
`snap_mplab_dw`	MPLAB(R) SNAP (debugWIRE)
`snap_mplab_isp`	MPLAB(R) SNAP (ISP)
`snap_mplab_pdi`	MPLAB(R) SNAP (PDI)
`snap_mplab_tpi`	MPLAB(R) SNAP (TPI)
`snap_mplab_updi`	MPLAB(R) SNAP (UPDI)
`snap_pdi`	MPLAB(R) SNAP (PDI)
`snap_tpi`	MPLAB(R) SNAP (TPI)
`snap_updi`	MPLAB(R) SNAP (UPDI)
`sp12`	Steve Bolt's Programmer (TPI, ISP)
`stk200`	STK200 starter kit (TPI, ISP)
`stk500`	Atmel STK500 (probes v2 first then v1) (ISP)
`stk500hvsp`	Atmel STK500 v2 (HVSP)
`scratchmonkey_hvsp`	Atmel STK500 v2 (HVSP)
`stk500pp`	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)
`ttl232r`	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 `eeprom`, `flash`, `signature` and `lock`:

`calibration`
: One or more bytes of RC oscillator calibration data

`efuse`
: Extended fuse byte

`fuse`
: Fuse byte in devices that have only a single fuse byte

`hfuse`
: High fuse byte

`lfuse`
: Low fuse byte

`prodsig`
: 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

`sigrow`
: Memory alias for prodsig

`sernum`
: The serial number part of prodsig; owing to scarce documentation this may not actually turn out to be a serial number or be readable by some programmers

`usersig`
: 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 usersig memory. Usersig is different to flash 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 usersig memories. As with all flash-type memories the `-U` option can only write 0-bits but not 1-bits. Hence, usersig needs to be erased before a file can be written to this memory region, e.g., using `-T "erase usersig" -U usersig:w:parameters.hex:i`

`io`
: 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

`sram`
: Volatile RAM memory; like `io` it cannot be accessed by external programming

E.2 ATxmegas

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

`application`
: Application flash area

`apptable`
: 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 `fuse`X 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

`fuse`*N* Other fuse bytes of ATxmega devices, where *N* 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 `eeprom`, `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 opcodes

`fuse5`	A.k.a. `syscfg0`: system configuration 0
`fuse6`	A.k.a. `syscfg1`: system configuration 1
`fuse7`	A.k.a. `append` or `codesize`: either the end of the application code section or the code size in blocks of 256/512 bytes
`fuse8`	A.k.a. `bootend` or `bootsize`: end of the boot section or the boot size in blocks of 256/512 bytes
`fusea`	A.k.a. `pdicfg`: configures/locks updi access; it is the only fuse that consists of two bytes
`fuses`	A logical memory of up to 16 bytes containing all fuseX of a part, which can be used to program all fuses at the same time
`osc16err`	Two bytes typically describing the 16 MHz oscillator frequency error at 3 V and 5 V, respectively (not all devices)
`osc20err`	Two bytes typically describing the 20 MHz oscillator frequency error at 3 V and 5 V, respectively (not all devices)
`osccal16`	One or two oscillator calibration bytes for 16 MHz (not all devices)
`osccal20`	One or two oscillator calibration bytes for 20 MHz (not all devices)
`prodsig`	Read-only memory section for factory programmed data such as the signature, calibration values and serial number
`sigrow`	Memory alias for prodsig
`sernum`	Serial number with a unique ID for the part (10 or 16 bytes)
`tempsense`	Temperature sensor calibration values
`bootrow`	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
`userrow`	Extra page of EEPROM memory that can be used for firmware settings; this memory is not erased during a chip erase
`sib`	Special system information block memory with information about AVR family, chip revision etc.
`io`	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
`sram`	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 20
`-x` Atmel-ICE 17
`-x` AVR Dragon 17
`-x` AVR109 20
`-x` AVR910 20
`-x` BusPirate 23
`-x` Curiosity Nano 19
`-x` dryboot 17
`-x` dryrun 17
`-x` flip2 16
`-x` jtag2updi 26
`-x` JTAG ICE mkII/3 17
`-x` linuxgpio 16
`-x` linuxspi 16, 27
`-x` Micronucleus bootloader 25
`-x` MPLAB(R) SNAP 17, 18
`-x` parallel port programmers 16
`-x` *parameter* 15
`-x` PICkit 4 ..... 17, 18
`-x` PICkit 5 18
`-x` PICkit2 26
`-x` pickit4_mplab, pickit5 16
`-x` Power Debugger 17
`-x` raspberry_pi_gpio 16
`-x` serialupdi 26
`-x` serprog 27
`-x` STK500 19
`-x` STK600 19
`-x` Teensy bootloader 25
`-x` Urclock 21
`-x` USBasp 26
`-x` Wiring 25
`-x` xbee 26
`-x` Xplained Mini 19

?

`?` (help) 44

1

1200 ... 87
128da28 ... 93
128da28s ... 93
128da32 ... 93
128da32s ... 93
128da48 ... 93
128da48s ... 93
128da64 ... 93
128da64s ... 93
128db28 ... 93
128db32 ... 93
128db48 ... 93
128db64 ... 93
16dd14 ... 93
16dd20 ... 94
16dd28 ... 94
16dd32 ... 94
16du14 ... 94
16du20 ... 94
16du28 ... 94
16du32 ... 94
16ea28 ... 94
16ea32 ... 94
16ea48 ... 94
16eb14 ... 94
16eb20 ... 94
16eb28 ... 94
16eb32 ... 94
16la14 ... 94
16la20 ... 94
16la28 ... 94
16la32 ... 94

2

2232h ... 82
2232hio ... 82
2313 ... 87
2323 ... 87
2333 ... 87
2343 ... 87

3

32da28 ... 94
32da28s ... 94
32da32 ... 94
32da32s ... 94
32da48 ... 94
32da48s ... 94
32db28 ... 94
32db32 ... 94
32db48 ... 94
32dd14 ... 94
32dd20 ... 94
32dd28 ... 94
32dd32 ... 94
32du14 ... 94
32du20 ... 94
32du28 ... 94
32du32 ... 94
32ea28 ... 94
32ea32 ... 94
32ea48 ... 94
32eb14 ... 94
32eb20 ... 94
32eb28 ... 94
32eb32 ... 94
32la14 ... 94
32la20 ... 94
32la28 ... 94
32la32 ... 94
32sd20 ... 94
32sd28 ... 95
32sd32 ... 95

4

4232h ... 82
4414 ... 87
4433 ... 87
4434 ... 87

6

64da28 ... 95
64da28s ... 95
64da32 ... 95
64da32s ... 95
64da48 ... 95
64da48s ... 95
64da64 ... 95
64da64s ... 95
64db28 ... 95
64db32 ... 95
64db48 ... 95
64db64 ... 95
64dd14 ... 95
64dd20 ... 95
64dd28 ... 95
64dd32 ... 95
64du28 ... 95
64du32 ... 95
64ea28 ... 95
64ea32 ... 95
64ea48 ... 95

8

8515 ... 87
8535 ... 87
89isp ... 82
89S51 ... 87
89S52 ... 87
8ea28 ... 95
8ea32 ... 95

A

a5505 ... 87
a6612c ... 87
a6613c ... 87
a6614q ... 87
a6616c ... 87
a6617c ... 87
a664251 ... 87
abcmini ... 82
ABCmini Board ... 82
abort ... 42
adafruit_gemma ... 82
alf ... 82
allow_subshells ... 53
Altera ByteBlaster ... 82
Amontec JTAGKey ... 83
append ... 98
application ... 96
apptable ... 96
arduino ... 82
Arduino bootloader ... 3, 20, 82
Arduino Gemma bootloader ... 82
Arduino-branded USBtiny ... 82
arduino-ft232r ... 82
Arduino: FT232R to ISP ... 82
arduino_as_isp ... 82
arduino_gemma ... 82
arduinoisp ... 82
arduinoisporg ... 82
At89isp cable ... 82
AT-ISP v1.1 cable ... 82
AT32UC3A0512, AT32UC3A0512AU ... 87
AT89S51 ... 87
AT89S52 ... 87
AT90CAN128 ... 87
AT90CAN32 ... 87
AT90CAN64 ... 87
AT90PWM1 ... 87
AT90PWM161 ... 87
AT90PWM2 ... 87
AT90PWM216 ... 87
AT90PWM2B ... 87
AT90PWM3 ... 87
AT90PWM316 ... 87
AT90PWM3B ... 87
AT90PWM81, AT90PWM81EP ... 87
AT90S1200, AT90S1200A ... 87
AT90S2313 ... 87
AT90S2323 ... 87
AT90S2333 ... 87
AT90S2343 ... 87
AT90S4414 ... 87
AT90S4433 ... 87
AT90S4434 ... 87
AT90S8515 ... 87
AT90S8535 ... 87
AT90USB1286 ... 87
AT90USB1287 ... 87
AT90USB162 ... 87
AT90USB646 ... 87
AT90USB647 ... 87
AT90USB82 ... 87
ATA5505 ... 87
ATA6612C ... 87
ATA6613C ... 87
ATA6614Q ... 87
ATA6616C ... 87
ATA6617C ... 87
ATA664251 ... 87
atisp ... 82
ATmega103, ATmega103L ... 87
ATmega128, ATmega128L ... 87
ATmega1280, ATmega1280V ... 87
ATmega1281, ATmega1281V ... 88
ATmega1284 ... 88
ATmega1284P ... 88
ATmega1284RFR2 ... 88
ATmega128A ... 88
ATmega128RFA1 ... 88
ATmega128RFR2 ... 88
ATmega16, ATmega16L ... 88
ATmega1608 ... 88
ATmega1609 ... 88
ATmega161, ATmega161L ... 88
ATmega162, ATmega162L, ATmega162V ... 88
ATmega163, ATmega163L ... 88
ATmega164A ... 88
ATmega164P, ATmega164PV ... 88
ATmega164PA ... 88
ATmega165, ATmega165V ... 88
ATmega165A ... 88
ATmega165P, ATmega165PV ... 88
ATmega165PA ... 88
ATmega168, ATmega168V ... 88
ATmega168A ... 88
ATmega168P, ATmega168PV ... 88
ATmega168PA ... 88
ATmega168PB ... 88
ATmega169, ATmega169L, ATmega169V ... 88
ATmega169A ... 88
ATmega169P, ATmega169PV ... 88
ATmega169PA ... 88
ATmega16A ... 88
ATmega16HVA ... 88
ATmega16HVB ... 88
ATmega16HVBrevB, ATMEGA16HVB ... 88

ATmega16M1 ... 88
ATmega16U2 ... 88
ATmega16U4, ATmega16U4RC ... 88
ATmega2560, ATmega2560V ... 88
ATmega2561, ATmega2561V ... 88
ATmega2564RFR2 ... 88
ATmega256RFR2 ... 88
ATmega32, ATmega32L ... 88
ATmega3208 ... 88
ATmega3209 ... 88
ATmega324A ... 89
ATmega324P, ATmega324PV ... 89
ATmega324PA ... 89
ATmega324PB ... 89
ATmega325, ATmega325V ... 89
ATmega3250, ATmega3250V ... 89
ATmega3250A ... 89
ATmega3250P, ATmega3250PV ... 89
ATmega3250PA ... 89
ATmega325A ... 89
ATmega325P, ATmega325PV ... 89
ATmega325PA ... 89
ATmega328 ... 89
ATmega328P ... 89
ATmega328PB ... 89
ATmega329, ATmega329V ... 89
ATmega3290, ATmega3290V ... 89
ATmega3290A ... 89
ATmega3290P, ATmega3290PV ... 89
ATmega3290PA ... 89
ATmega329A ... 89
ATmega329P, ATmega329PV ... 89
ATmega329PA ... 89
ATmega32A ... 89
ATmega32C1 ... 89
ATmega32HVB ... 89
ATmega32HVBrevB, ATMEGA32HVB ... 89
ATmega32HVE2 ... 89
ATmega32M1 ... 89
ATmega32U2 ... 89
ATmega32U4, ATmega32U4RC ... 89
ATmega406 ... 89
ATmega48, ATmega48V ... 89
ATmega4808 ... 89
ATmega4809 ... 89
ATmega48A ... 89
ATmega48P, ATmega48PV ... 89
ATmega48PA ... 89
ATmega48PB ... 89
ATmega64, ATmega64L ... 89
ATmega640, ATmega640V ... 89
ATmega644, ATmega644V ... 89
ATmega644A ... 89
ATmega644P, ATmega644PV ... 89
ATmega644PA ... 89
ATmega644RFR2 ... 90
ATmega645, ATmega645V ... 90
ATmega6450, ATmega6450V ... 90
ATmega6450A ... 90
ATmega6450P ... 90
ATmega645A ... 90
ATmega645P ... 90
ATmega649, ATmega649V ... 90
ATmega6490, ATmega6490V ... 90
ATmega6490A ... 90
ATmega6490P ... 90
ATmega649A ... 90
ATmega649P ... 90
ATmega64A ... 90
ATmega64C1 ... 90
ATmega64HVE2 ... 90
ATmega64M1 ... 90
ATmega64RFR2 ... 90
ATmega8, ATmega8L ... 90
ATmega808 ... 90
ATmega809 ... 90
ATmega8515, ATmega8515L ... 90
ATmega8535, ATmega8535L ... 90
ATmega88, ATmega88V ... 90
ATmega88A ... 90
ATmega88P, ATmega88PV ... 90
ATmega88PA ... 90
ATmega88PB ... 90
ATmega8A ... 90
ATmega8HVA ... 90
ATmega8U2 ... 90
ATmegaS128 ... 90
ATmegaS64M1 ... 90
Atmel at89isp cable ... 82
Atmel AVR Dragon ... 3, 17, 83
Atmel AVR JTAGICE3 ... 2, 17, 83
Atmel bootloader (AVR109, AVR911) ... 2, 20, 82
Atmel bootloader (Butterfly) ... 82
Atmel JTAG ICE mkI ... 2, 83, 84
Atmel JTAG ICE mkII ... 2, 17, 83, 84
Atmel low-cost programmer AVR910 ... 20, 82
Atmel PowerDebugger ... 17, 85
Atmel STK500 ... 2, 19, 85
Atmel STK500 v1 ... 86
Atmel STK500 v2 ... 85, 86
Atmel STK600 ... 2, 19, 68, 86
Atmel XplainedMini ... 3, 19, 86
Atmel XplainedPro ... 3, 86
Atmel-ICE ... 3, 82
`atmelice` ... 82
`atmelice_dw` ... 82
`atmelice_isp` ... 82
`atmelice_jtag` ... 82
`atmelice_pdi` ... 82
`atmelice_tpi` ... 82
`atmelice_updi` ... 82
ATtiny10 ... 90
ATtiny102, ATtiny102F ... 90
ATtiny104, ATtiny104F ... 90
ATtiny11, ATtiny11L ... 90
ATtiny12, ATtiny12L, ATtiny12V ... 90

ATtiny13, ATtiny13V ... 90
ATtiny13A ... 90
ATtiny15, ATtiny15L ... 90
ATtiny1604 ... 90
ATtiny1606 ... 90
ATtiny1607 ... 90
ATtiny1614 ... 90
ATtiny1614auto, ATtiny1614 ... 90
ATtiny1616 ... 90
ATtiny1616auto, ATtiny1616 ... 91
ATtiny1617 ... 91
ATtiny1617auto, ATtiny1617 ... 91
ATtiny1624 ... 91
ATtiny1626 ... 91
ATtiny1627 ... 91
ATtiny1634, ATtiny1634R ... 91
ATtiny1634R ... 91
ATtiny167 ... 91
ATtiny20 ... 91
ATtiny202 ... 91
ATtiny204 ... 91
ATtiny212 ... 91
ATtiny214 ... 91
ATtiny22, ATtiny22L ... 91
ATtiny2313, ATtiny2313V ... 91
ATtiny2313A ... 91
ATtiny24, ATtiny24V ... 91
ATtiny24A ... 91
ATtiny25, ATtiny25V ... 91
ATtiny26, ATtiny26L ... 91
ATtiny261, ATtiny261V ... 91
ATtiny261A ... 91
ATtiny28, ATtiny28L, ATtiny28V ... 91
ATtiny3216 ... 91
ATtiny3216auto, ATtiny3216 ... 91
ATtiny3217 ... 91
ATtiny3217auto, ATtiny3217 ... 91
ATtiny3224 ... 91
ATtiny3226 ... 91
ATtiny3227 ... 91
ATtiny4 ... 91
ATtiny40 ... 91
ATtiny402 ... 91
ATtiny404 ... 91
ATtiny406 ... 91
ATtiny412 ... 91
ATtiny414 ... 91
ATtiny416 ... 91
ATtiny416auto, ATtiny416 ... 91
ATtiny417 ... 91
ATtiny417auto, ATtiny417 ... 91
ATtiny424 ... 91
ATtiny426 ... 91
ATtiny427 ... 91
ATtiny4313 ... 91
ATtiny43U ... 91
ATtiny44, ATtiny44V ... 92
ATtiny441 ... 92
ATtiny44A ... 92
ATtiny45, ATtiny45V ... 92
ATtiny461, ATtiny461V ... 92
ATtiny461A ... 92
ATtiny48 ... 92
ATtiny5 ... 92
ATtiny804 ... 92
ATtiny806 ... 92
ATtiny807 ... 92
ATtiny814 ... 92
ATtiny814auto, ATtiny814 ... 92
ATtiny816 ... 92
ATtiny816auto, ATtiny816 ... 92
ATtiny817 ... 92
ATtiny817auto, ATtiny817 ... 92
ATtiny824 ... 92
ATtiny826 ... 92
ATtiny827 ... 92
ATtiny828, ATtiny828R ... 92
ATtiny828R ... 92
ATtiny84, ATtiny84V ... 92
ATtiny841 ... 92
ATtiny84A ... 92
ATtiny85, ATtiny85V ... 92
ATtiny861, ATtiny861V ... 92
ATtiny861A ... 92
ATtiny87 ... 92
ATtiny88 ... 92
ATtiny9 ... 92
ATxmega128A1 ... 92
ATxmega128A1revD ... 92
ATxmega128A1U ... 92
ATxmega128A3 ... 92
ATxmega128A3U ... 92
ATxmega128A4 ... 92
ATxmega128A4U ... 92
ATxmega128B1 ... 92
ATxmega128B3 ... 92
ATxmega128C3 ... 92
ATxmega128D3 ... 92
ATxmega128D4 ... 92
ATxmega16A4 ... 92
ATxmega16A4U ... 92
ATxmega16C4 ... 92
ATxmega16D4 ... 92
ATxmega16E5 ... 93
ATxmega192A1 ... 93
ATxmega192A3 ... 93
ATxmega192A3U ... 93
ATxmega192C3 ... 93
ATxmega192D3 ... 93
ATxmega256A1 ... 93
ATxmega256A3 ... 93
ATxmega256A3B ... 93
ATxmega256A3BU ... 93
ATxmega256A3U ... 93
ATxmega256C3 ... 93
ATxmega256D3 ... 93

ATxmega32A4 ... 93
ATxmega32A4U ... 93
ATxmega32C3 ... 93
ATxmega32C4 ... 93
ATxmega32D3 ... 93
ATxmega32D4 ... 93
ATxmega32E5 ... 93
ATxmega384C3 ... 93
ATxmega384D3 ... 93
ATxmega64A1 ... 93
ATxmega64A1U ... 93
ATxmega64A3 ... 93
ATxmega64A3U ... 93
ATxmega64A4 ... 93
ATxmega64A4U ... 93
ATxmega64B1 ... 93
ATxmega64B3 ... 93
ATxmega64C3 ... 93
ATxmega64D3 ... 93
ATxmega64D4 ... 93
ATxmega8E5 ... 93
Auto-detect mode ... 14
Auto-erase ... 9
Autogenerated files ... 62
`avr109` ... 82
`avr910` ... 82
`avr911` ... 82
AVR as programmer ... 82
AVR Dragon ... 3, 17, 83
AVR ISP programmer ... 83
AVR JTAGICE3 ... 2, 17, 83
AVR128DA28, AVR128DA28T ... 93
AVR128DA28S ... 93
AVR128DA32, AVR128DA32T ... 93
AVR128DA32S ... 93
AVR128DA48, AVR128DA48T ... 93
AVR128DA48S ... 93
AVR128DA64, AVR128DA64T ... 93
AVR128DA64S ... 93
AVR128DB28, AVR128DB28T ... 93
AVR128DB32, AVR128DB32T ... 93
AVR128DB48, AVR128DB48T ... 93
AVR128DB64, AVR128DB64T ... 93
AVR16DD14 ... 93
AVR16DD20 ... 94
AVR16DD28 ... 94
AVR16DD32 ... 94
AVR16DU14 ... 94
AVR16DU20 ... 94
AVR16DU28 ... 94
AVR16DU32 ... 94
AVR16EA28 ... 94
AVR16EA32 ... 94
AVR16EA48 ... 94
AVR16EB14 ... 94
AVR16EB20 ... 94
AVR16EB28 ... 94
AVR16EB32 ... 94
AVR16LA14, AVR16LA14T ... 94
AVR16LA20, AVR16LA20T ... 94
AVR16LA28, AVR16LA28T ... 94
AVR16LA32, AVR16LA32T ... 94
AVR32DA28, AVR32DA28T ... 94
AVR32DA28S ... 94
AVR32DA32, AVR32DA32T ... 94
AVR32DA32S ... 94
AVR32DA48, AVR32DA48T ... 94
AVR32DA48S ... 94
AVR32DB28, AVR32DB28T ... 94
AVR32DB32, AVR32DB32T ... 94
AVR32DB48, AVR32DB48T ... 94
AVR32DD14 ... 94
AVR32DD20 ... 94
AVR32DD28 ... 94
AVR32DD32 ... 94
AVR32DU14 ... 94
AVR32DU20 ... 94
AVR32DU28 ... 94
AVR32DU32 ... 94
AVR32EA28 ... 94
AVR32EA32 ... 94
AVR32EA48 ... 94
AVR32EB14 ... 94
AVR32EB20 ... 94
AVR32EB28 ... 94
AVR32EB32 ... 94
AVR32JTAG ... 54
AVR32LA14, AVR32LA14T ... 94
AVR32LA20, AVR32LA20T ... 94
AVR32LA28, AVR32LA28T ... 94
AVR32LA32, AVR32LA32T ... 94
AVR32SD20 ... 94
AVR32SD28 ... 95
AVR32SD32 ... 95
AVR64DA28, AVR64DA28T ... 95
AVR64DA28S ... 95
AVR64DA32, AVR64DA32T ... 95
AVR64DA32S ... 95
AVR64DA48, AVR64DA48T ... 95
AVR64DA48S ... 95
AVR64DA64, AVR64DA64T ... 95
AVR64DA64S ... 95
AVR64DB28, AVR64DB28T ... 95
AVR64DB32, AVR64DB32T ... 95
AVR64DB48, AVR64DB48T ... 95
AVR64DB64, AVR64DB64T ... 95
AVR64DD14 ... 95
AVR64DD20 ... 95
AVR64DD28 ... 95
AVR64DD32 ... 95
AVR64DU28 ... 95
AVR64DU32 ... 95
AVR64EA28 ... 95
AVR64EA32 ... 95
AVR64EA48 ... 95
AVR8EA28 ... 95

AVR8EA32 ... 95
avrdude.conf ... 53
AVRDUDE defaults ... 53
avrftdi ... 82
avrisp ... 82
avrisp-u ... 82
avrisp2 ... 82
avrispmkII ... 82
avrispv2 ... 82
aWire ... 55

B

backup *memlist file[:format]* ... 41
bascom ... 82
Bascom SAMPLE cable ... 82
Binary file mode ... 14
BitWizard ftdi_atmega ... 82
blaster ... 82
bodcfg ... 97
boot ... 96
bootend ... 98
Bootloader (AVR109, AVR911) ... 20, 82
Bootloader (Butterfly) ... 82
bootrow ... 41, 42, 98
bootsize ... 98
Brian S. Dean's programmer ... 82
bsd ... 82
BusPirate ... 23, 82
buspirate ... 82
buspirate_bb ... 82
butterfly ... 2, 82
butterfly_mk ... 82
bwmega ... 82

C

c128 ... 87
c232hm ... 82
c2n232i ... 83
c32 ... 87
c64 ... 87
C232HM cable from FTDI ... 82
calibration ... 1, 10, 13, 96, 97, 98
ch341a ... 83
CH341A programmer ... 3, 83
codesize ... 98
Command line options ... 6
config *[-f|-a|-v]* ... 42
config *[-f|-v] property [-f|-v]* ... 42
config *[-f|-v] property= [-f|-v]* ... 42
config *[-f|-v|-c] property=value [-f|-v|-c]* ... 42
Configuration files ... 7, 53, 73
Crossbow MIB510 ... 84
Curiosity nano ... 4, 19, 85

D

d (dump) ... 44
dapa ... 83
dasa ... 83
dasa3 ... 83
debugWIRE ... 2, 54
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 ... 70
Dick Smith HOTCHIP ... 82
diecimila ... 82
Digilent JTAG HS2 ... 83
digilent-hs2 ... 83
Direct AVR Parallel cable ... 83
disasm *[memory [addr [len]]]* ... 37
disasm example ... 51
Dontronics DT006 ... 83
Dragon ... 3, 17
dragon_dw ... 83
dragon_hvsp ... 83
dragon_isp ... 83
dragon_jtag ... 83
dragon_pdi ... 83
dragon_pp ... 83
dryboot ... 1, 67, 83
dryrun ... 1, 67, 83
dt006 ... 83
Dual boot ... 67
dump *[memory [addr [len]]]* ... 37
dump *memory [addr] ...* ... 37

E

eeprom ... 1, 8, 9, 10, 13, 21, 22, 29, 41, 42, 70, 71, 77, 96, 97, 98
ehajo-isp ... 83
ELF (Executable and Linkable Format) ... 14
Emulating a bootloader (dryboot) ... 1, 17, 67, 83
Emulating a HW programmer (dryrun) ... 1, 17, 67, 83
erase ... 41
erase *memory [addr len]* ... 41
ere-isp-avr ... 83
ERE ISP-AVR ... 83
Example Command line invocations ... 28

F

`factory reset` ... 42
`flash` ... 1, 2, 4, 8, 9, 13, 14, 15, 17, 21, 22, 23, 24, 28, 29, 30, 39, 41, 51, 61, 70, 96, 97
Flashcom serprog protocol ... 27, 85
`flip1` ... 83
`flip2` ... 83
FLIP bootloader ... 4, 16
FLIP bootloader DFU v1 (doc7618) ... 83
FLIP bootloader DFU v2 (AVR4023) ... 83
`flush` ... 41, 42
`flyswatter2` ... 83
`fosc` *freq*[`M`|`k`] ... 44
`fosc off` ... 44
FOSC_ADJ ... 55
Frank STK200 ... 83
`frank-stk200` ... 83
FreeBSD configuration files ... 73
FreeBSD USB permissions ... 74
`ft2232h` ... 83
`ft2232h_jtag` ... 83
`ft232h` ... 83
`ft232h_jtag` ... 83
`ft232r` ... 83
`ft245r` ... 83
`ft4232h` ... 83
FT2232H JTAG programmer ... 2, 83
FT2232H with buffer and LEDs ... 2, 82
FT2232H/D programmer ... 2, 82, 83
FT232H JTAG programmer ... 2, 83
FT232H programmer ... 2, 83
FT232R programmer ... 2, 83
FT232R Synchronous BitBang ... 2
FT245R programmer ... 2, 83
FT4232H programmer ... 2, 82, 83
FTDI TTL232R-5V ... 2, 86
`fuse0` ... 97
`fuse1` ... 97
`fuse6` ... 97
`fuses` ... 97, 98
`futurlec` ... 83
Futurlec.com cable ... 83

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 ... 54
HVSP ... 54
`hwmoncfg` ... 97

I

Immediate file mode ... 14
`include` *[opts] file* ... 43
Installation ... 73, 75
Instruction format ... 59
Intel Hex ... 13
Introduction ... 1
`io` ... 97, 98
`iseavrprog` ... 83
ISP ... 2, 54

J

Jason Kyle's pAVR ... 84
`jtag1` ... 84
`jtag1slow` ... 83
`jtag2` ... 83
`jtag2avr32` ... 84
`jtag2dw` ... 83
`jtag2fast` ... 83
`jtag2isp` ... 83
`jtag2pdi` ... 83
`jtag2slow` ... 83
`jtag2updi` ... 83
`jtag3` ... 83
`jtag3dw` ... 83
`jtag3isp` ... 83
`jtag3pdi` ... 83
`jtag3updi` ... 83
JTAG ICE mkI ... 2, 83, 84
JTAG ICE mkII ... 2, 17, 83, 84
JTAGICE3 ... 2, 17
`jtagkey` ... 83
`jtagmkI` ... 84
`jtagmkII` ... 84
`jtagmkII_avr32` ... 84
JTAG ... 2, 54
JTAGmkI ... 54
JTAGv2 to UPDI bridge ... 26, 83

K

Kanda AVRISP-U ... 82
KT-LINK FT2232H ... 84
`ktlink` ... 84

L

Lancos SI-Prog ... 85
LED management ... 72
lgt168p ... 95
lgt328p ... 95
lgt88p ... 95
LGT8F168P ... 95
LGT8F328P ... 95
LGT8F88P ... 95
Linux /dev/spidev* programmer ... 27, 84
Linux configuration files ... 73
Linux USB permissions ... 74
linuxspi ... 84
lm3s811 ... 84
Low-cost programmer AVR910 ... 20, 82
Luminary Micro LM3S811 ... 84

M

m103 ... 87
m128 ... 87
m1280 ... 87
m1281 ... 88
m1284 ... 88
m1284p ... 88
m1284rfr2 ... 88
m128a ... 88
m128rfa1 ... 88
m128rfr2 ... 88
m16 ... 88
m1608 ... 88
m1609 ... 88
m161 ... 88
m162 ... 88
m163 ... 88
m164a ... 88
m164p ... 88
m164pa ... 88
m165 ... 88
m165a ... 88
m165p ... 88
m165pa ... 88
m168 ... 88
m168a ... 88
m168p ... 88
m168pa ... 88
m168pb ... 88
m169 ... 88
m169a ... 88
m169p ... 88
m169pa ... 88
m16a ... 88
m16hva ... 88
m16hvb ... 88
m16hvbrevb ... 88
m16m1 ... 88
m16u2 ... 88
m16u4 ... 88
m2560 ... 88
m2561 ... 88
m2564rfr2 ... 88
m256rfr2 ... 88
m32 ... 88
m3208 ... 88
m3209 ... 88
m324a ... 89
m324p ... 89
m324pa ... 89
m324pb ... 89
m325 ... 89
m3250 ... 89
m3250a ... 89
m3250p ... 89
m3250pa ... 89
m325a ... 89
m325p ... 89
m325pa ... 89
m328 ... 89
m328p ... 89
m328pb ... 89
m329 ... 89
m3290 ... 89
m3290a ... 89
m3290p ... 89
m3290pa ... 89
m329a ... 89
m329p ... 89
m329pa ... 89
m32a ... 89
m32c1 ... 89
m32hvb ... 89
m32hvbrevb ... 89
m32hve2 ... 89
m32m1 ... 89
m32u2 ... 89
m32u4 ... 89
m406 ... 89
m48 ... 89
m4808 ... 89
m4809 ... 89
m48a ... 89
m48p ... 89
m48pa ... 89
m48pb ... 89
m64 ... 89
m640 ... 89
m644 ... 89
m644a ... 89
m644p ... 89
m644pa ... 89
m644rfr2 ... 90
m645 ... 90
m6450 ... 90
m6450a ... 90
m6450p ... 90
m645a ... 90

m645p ... 90
m649 ... 90
m6490 ... 90
m6490a ... 90
m6490p ... 90
m649a ... 90
m649p ... 90
m64a ... 90
m64c1 ... 90
m64hve2 ... 90
m64m1 ... 90
m64rfr2 ... 90
m8 ... 90
m808 ... 90
m809 ... 90
m8515 ... 90
m8535 ... 90
m88 ... 90
m88a ... 90
m88p ... 90
m88pa ... 90
m88pb ... 90
m8a ... 90
m8hva ... 90
m8u2 ... 90
Memories of ATxmegas ... 96
Memories of classic parts ... 96
Memories of modern AVR parts ... 97
Metadata ... 23
mib510 ... 84
Microchip PICkit 2 programmer ... 26, 84
micronucleus ... 84
Micronucleus bootloader ... 4, 25, 84
Mikrokopter.de Butterfly ... 82
mkbutterfly ... 82
Motorola S-Record ... 13
MPLAB(R) PICkit 4 ... 4, 17, 18, 84
MPLAB(R) PICkit 5 ... 4, 17, 18, 84
MPLAB(R) PICkit Basic ... 4, 17, 84, 85
MPLAB(R) SNAP ... 4, 17, 18, 85
ms128 ... 90
ms64m1 ... 90

N

nanoevery ... 83
NIBObee ... 84
nibobee ... 84
Nightshade ALF-PgmAVR ... 82

O

o-link ... 84
O-Link, OpenJTAG ARM JTAG USB ... 84
Octal file mode ... 14
openmoko ... 84
Openmoko debug board ... 84
Option --baud *baudrate* ... 6
Option --bitclock *bitclock* ... 6
Option --command *cmd* ... 12
Option --config *config-file* ... 7
Option --erase ... 8
Option --exitspecs *exitspec*[,...] ... 9
Option --extended *parameter* ... 15
Option --force ... 9
Option --help ... 15
Option --isp-clock-delay *delay* ... 9
Option --keep-trailing-0xff ... 8
Option --logfile *logfile* ... 10
Option --memory *mem*:*op*:*file*[:*fmt*] ... 12
Option --noconfig ... 8
Option --noerase ... 8
Option --noverify-memory ... 15
Option --osccal ... 10
Option --part *partname* ... 6
Option --part *wildcard/flags* ... 6
Option --port *port* ... 10
Option --programmer *programmer-id* ... 7
Option --programmer *wildcard/flags* ... 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 *baudrate* ... 6
Option -B *bitclock* ... 6
Option -c *config-file* ... 7
Option -c *programmer-id* ... 7
Option -c *wildcard/flags* ... 7
Option -D ... 8
Option -e ... 8
Option -E d_high ... 16
Option -E d_low ... 16
Option -E *exitspec*[,...] ... 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 *delay* ... 9
Option -l *logfile* ... 10
Option -n ... 10
Option -N ... 8
Option -O ... 10
Option -p *partname* ... 6
Option -p *wildcard/flags* ... 6

Option `-P` *port* ... 10
Option `-q` ... 12
Option `-r` ... 12
Option `-t` ... 12
Option `-T` *cmd* ... 12
Option `-U` *mem*:*op*:*file*[:*fmt*] ... 12
Option `-v` ... 15
Option `-V` ... 15
Option `-x` Arduino ... 20
Option `-x` Atmel-ICE ... 17
Option `-x` AVR Dragon ... 17
Option `-x` AVR109 ... 20
Option `-x` AVR910 ... 20
Option `-x` BusPirate ... 23
Option `-x` Curiosity Nano ... 19
Option `-x` dryboot ... 17
Option `-x` dryrun ... 17
Option `-x` flip2 ... 16
Option `-x` jtag2updi ... 26
Option `-x` JTAG ICE mkII/3 ... 17
Option `-x` linuxgpio ... 16
Option `-x` linuxspi ... 16, 27
Option `-x` Micronucleus bootloader ... 25
Option `-x` MPLAB(R) SNAP ... 17, 18
Option `-x` parallel port programmers ... 16
Option `-x` *parameter* ... 15
Option `-x` PICkit 4 ... 17, 18
Option `-x` PICkit 5 ... 18
Option `-x` PICkit2 ... 26
Option `-x` pickit4_mplab, pickit5 ... 16
Option `-x` Power Debugger ... 17
Option `-x` raspberry_pi_gpio ... 16
Option `-x` serialupdi ... 26
Option `-x` serprog ... 27
Option `-x` STK500 ... 19
Option `-x` STK600 ... 19
Option `-x` Teensy bootloader ... 25
Option `-x` Urclock ... 21
Option `-x` USBasp ... 26
Option `-x` Wiring ... 25
Option `-x` xbee ... 26
Option `-x` Xplained Mini ... 19
Option descriptions ... 6
Options (command-line) ... 6
`osc16err` ... 98
`osc20err` ... 98
`osccal16` ... 98
`osccal20` ... 98
`osccfg` ... 97
Other notes ... 60

P

Parent part ... 59
`parms` ... 45
`part` ... 46, 57
`part` *[opts]* ... 43
Part definitions ... 57
Patching the vector table ... 67
`pavr` ... 84
`pdicfg` ... 98
PDI ... 2, 54
`pgerase` *memory addr* ... 44
`pgm` ... 44
PICkit 2 programmer ... 26, 84
PICkit 4 ... 4, 17, 18, 84
PICkit 5 ... 4, 17, 18, 84
PICkit Basic ... 4, 17, 84, 85
`pickit_basic` ... 84
`pickit_basic_dw` ... 84
`pickit_basic_isp` ... 84
`pickit_basic_jtag` ... 84
`pickit_basic_mplab` ... 84
`pickit_basic_mplab_dw` ... 84
`pickit_basic_mplab_isp` ... 84
`pickit_basic_mplab_jtag` ... 84
`pickit_basic_mplab_pdi` ... 85
`pickit_basic_mplab_tpi` ... 85
`pickit_basic_mplab_updi` ... 85
`pickit_basic_pdi` ... 85
`pickit_basic_tpi` ... 85
`pickit_basic_updi` ... 85
`pickit2` ... 84
`pickit4` ... 84
`pickit4_dw` ... 84
`pickit4_isp` ... 84
`pickit4_jtag` ... 84
`pickit4_mplab` ... 84
`pickit4_mplab_dw` ... 84
`pickit4_mplab_isp` ... 84
`pickit4_mplab_jtag` ... 84
`pickit4_mplab_pdi` ... 84
`pickit4_mplab_tpi` ... 84
`pickit4_mplab_updi` ... 84
`pickit4_pdi` ... 84
`pickit4_tpi` ... 84
`pickit4_updi` ... 84
`pickit5` ... 84
`pickit5_dw` ... 84
`pickit5_isp` ... 84
`pickit5_jtag` ... 84
`pickit5_pdi` ... 84
`pickit5_tpi` ... 84
`pickit5_updi` ... 84
`picoweb` ... 85
Picoweb Programming Cable ... 85
`pincfg` ... 97
`pkobn_updi` ... 85
`PM_AVR32JTAG` ... 54
`PM_aWire` ... 55

PM_debugWIRE ... 54
PM_HVPP ... 54
PM_HVSP ... 54
PM_ISP ... 54
PM_JTAG ... 54
PM_JTAGmkI ... 54
PM_PDI ... 54
PM_SPM ... 54
PM_TPI ... 54
PM_UPDI ... 54
PM_XMEGAJTAG ... 54
Pony Prog STK200 ... 85
pony-stk200 ... 85
ponyser ... 85
PowerDebugger ... 17, 85
powerdebugger ... 85
powerdebugger_dw ... 85
powerdebugger_isp ... 85
powerdebugger_jtag ... 85
powerdebugger_pdi ... 85
powerdebugger_tpi ... 85
powerdebugger_updi ... 85
prodsig ... 97, 98
programmer ... 53
Programmer definitions ... 54
Programmer LED management ... 72
Programmer-specific information ... 68
Programmers accepting exitspec parameters ... 16
Programmers accepting extended parameters ... 17
Programmers supported ... 1, 82
Programming mode ... 43
Programming modes ... 54
pwm1 ... 87
pwm161 ... 87
pwm2 ... 87
pwm216 ... 87
pwm2b ... 87
pwm3 ... 87
pwm316 ... 87
pwm3b ... 87
pwm81 ... 87

Q

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

R

r (read) ... 44
raspberry_pi_gpio ... 85
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 ... 85

S

save *memory* {*addr len*} *file[:format]* ... 41
sck *period* ... 44
scratchmonkey ... 86
scratchmonkey_hvsp ... 85
scratchmonkey_pp ... 85
send *b1 b2 b3 b4* ... 44
Serial adapter definitions ... 56
Serial Atmel AVR ISP ... 82
Serial Atmel AVR ISPv2 ... 82
Serial port programmer ... 83, 85
serialadapter ... 56
serialupdi ... 85
SerialUPDI ... 4, 26, 70, 85
SerialUPDI programmer ... 70
sernum ... 97, 98
serprog ... 85
sib ... 98
signature ... 1, 9, 13, 43, 71, 96, 97, 98
sigrow ... 97, 98
siprog ... 85
snap ... 85
snap_dw ... 85
snap_isp ... 85
snap_jtag ... 85
snap_mplab ... 85
snap_mplab_dw ... 85
snap_mplab_isp ... 85
snap_mplab_jtag ... 85
snap_mplab_pdi ... 85
snap_mplab_tpi ... 85
snap_mplab_updi ... 85
snap_pdi ... 85
snap_tpi ... 85
snap_updi ... 85
SNAP ... 4, 17, 18, 85
sp12 ... 85
spi ... 44
SPM ... 54
sram ... 97, 98
Steve Bolt's Programmer ... 85
stk200 ... 85
stk500 ... 85
stk500hvsp ... 85
stk500pp ... 85

stk500v1 ... 86
stk500v2 ... 86
stk600 ... 86
stk600hvsp ... 86
stk600pp ... 86
STK200 starter kit ... 85
STK500 ... 2, 19, 85
STK500 v1 ... 86
STK500 v2 ... 85, 86
STK600 ... 2, 19, 68, 86
SUFFER ... 55
syscfg0 ... 97
syscfg1 ... 98

T

t10 ... 90
t102 ... 90
t104 ... 90
t11 ... 90
t12 ... 90
t13 ... 90
t13a ... 90
t15 ... 90
t1604 ... 90
t1606 ... 90
t1607 ... 90
t1614 ... 90
t1614auto ... 90
t1616 ... 90
t1616auto ... 91
t1617 ... 91
t1617auto ... 91
t1624 ... 91
t1626 ... 91
t1627 ... 91
t1634 ... 91
t1634r ... 91
t167 ... 91
t20 ... 91
t202 ... 91
t204 ... 91
t212 ... 91
t214 ... 91
t22 ... 91
t2313 ... 91
t2313a ... 91
t24 ... 91
t24a ... 91
t25 ... 91
t26 ... 91
t261 ... 91
t261a ... 91
t28 ... 91
t3216 ... 91
t3216auto ... 91
t3217 ... 91
t3217auto ... 91
t3224 ... 91
t3226 ... 91
t3227 ... 91
t4 ... 91
t40 ... 91
t402 ... 91
t404 ... 91
t406 ... 91
t412 ... 91
t414 ... 91
t416 ... 91
t416auto ... 91
t417 ... 91
t417auto ... 91
t424 ... 91
t426 ... 91
t427 ... 91
t4313 ... 91
t43u ... 91
t44 ... 92
t441 ... 92
t44a ... 92
t45 ... 92
t461 ... 92
t461a ... 92
t48 ... 92
t5 ... 92
t804 ... 92
t806 ... 92
t807 ... 92
t814 ... 92
t814auto ... 92
t816 ... 92
t816auto ... 92
t817 ... 92
t817auto ... 92
t824 ... 92
t826 ... 92
t827 ... 92
t828 ... 92
t828r ... 92
t84 ... 92
t841 ... 92
t84a ... 92
t85 ... 92
t861 ... 92
t861a ... 92
t87 ... 92
t88 ... 92
t9 ... 92
Tag-Connect TC2030 ... 86
Tagfile ... 38
tc2030 ... 86
tcd0cfg ... 97
teensy ... 86
Teensy bootloader ... 4, 25, 86
tempsense ... 97, 98
Terminal mode commands ... 36

Terminal mode examples ... 46
Terminal mode operation ... 36
The Bus Pirate ... 3, 23, 82
TIAO USB programmer ... 86
`tigard` ... 86
Tigard interface board ... 86
TinCan Tools Flyswatter 2 ... 83
TPI ... 54
Trinket Gemma bootloader ... 82
`ttl232r` ... 86
`tumpa` ... 86
`tumpa-b` ... 86
`tumpa_jtag` ... 86

U

`uc3a0512` ... 87
`um232h` ... 86
UM232H module from FTDI ... 86
`uncompatino` ... 86
Uncompatino programmer ... 86
Unix ... 73
Unix configuration files ... 73
Unix documentation ... 75
Unix installation ... 73
Unix port names ... 73
Unix USB permissions ... 73
UPDI ... 54
Urboot bootloader ... 3, 21, 86
`urboot:` ... 62
urclock ... 67, 86
Urclock programmer ... 3, 21, 86
Urprotocol ... 3, 21, 86
`usb1286` ... 87
`usb1287` ... 87
`usb162` ... 87
`usb646` ... 87
`usb647` ... 87
`usb82` ... 87
USB Atmel AVR ISP mkII ... 82
USB permissions ... 73, 74, 75
`usbasp` ... 86
Usbasp clones ... 86
USBasp ISP and TPI programmer ... 2, 26, 86
`usbasp-clone` ... 86
`usbtiny` ... 86
USBtiny simple USB programmer ... 2, 86
`userrow` ... 98
`usersig` ... 41, 42, 97

V

`varef` *[channel] voltage* ... 44
VAREF_ADJ ... 55
Variants of parts ... 43
Vector table ... 67
`verbose` *[level]* ... 43
`verify` *memlist file[:format]* ... 41
`vtarg` *voltage* ... 44
VTARG_ADJ ... 55
VTARG_READ ... 55
VTARG_SWITCH ... 55

W

`w` (write) ... 44
`wdtcfg` ... 97
Windows ... 75
Windows configuration file location ... 76
Windows configuration file names ... 76
Windows configuration files ... 75
Windows parallel ports ... 76
Windows port names ... 76
Windows serial ports ... 76
`wiring` ... 86
Wiring bootloader ... 3, 25, 86
`write` *memory addr data[,]* {*data[,]*} ... 39
`write` *memory addr len data[,]* {*data[,]*} ... 40
`write` *memory data* ... 40
`write` *memory file* ... 40

X

`x128a1` ... 92
`x128a1d` ... 92
`x128a1u` ... 92
`x128a3` ... 92
`x128a3u` ... 92
`x128a4` ... 92
`x128a4u` ... 92
`x128b1` ... 92
`x128b3` ... 92
`x128c3` ... 92
`x128d3` ... 92
`x128d4` ... 92
`x16a4` ... 92
`x16a4u` ... 92
`x16c4` ... 92
`x16d4` ... 92
`x16e5` ... 93
`x192a1` ... 93
`x192a3` ... 93
`x192a3u` ... 93
`x192c3` ... 93
`x192d3` ... 93
`x256a1` ... 93
`x256a3` ... 93
`x256a3b` ... 93
`x256a3bu` ... 93

x256a3u 93
x256c3 93
x256d3 93
x32a4 93
x32a4u 93
x32c3 93
x32c4 93
x32d3 93
x32d4 93
x32e5 93
x384c3 93
x384d3 93
x64a1 93
x64a1u 93
x64a3 93
x64a3u 93
x64a4 93
x64a4u 93
x64b1 93
x64b3 93
x64c3 93
x64d3 93
x64d4 93
x8e5 93
xbee 86
XBeeBoot OTA bootloader 26, 86
xil 86
Xilinx JTAG cable 86
XMEGAJTAG 54
XplainedMini 3, 19, 86
xplainedmini 86
xplainedmini_dw 86
xplainedmini_isp 86
xplainedmini_tpi 86
xplainedmini_updi 86
XplainedPro 3, 86
xplainedpro 86
xplainedpro_jtag 86
xplainedpro_pdi 86
xplainedpro_updi 86