

# **USBprog User's Manual**

Bernhard Walle  
[bernhard@bwalle.de](mailto:bernhard@bwalle.de)

5th January 2010

# Contents

<b>1</b>	<b>Overview</b>	<b>3</b>
1.1	The USBprog Hardware . . . . .	3
1.2	The USBprog Software . . . . .	3
1.3	About this Document . . . . .	4
1.4	Terms and Definitions . . . . .	4
1.5	Getting Support . . . . .	4
<b>2</b>	<b>Getting Started with the Hardware</b>	<b>5</b>
2.1	Connectors, Jumpers and LEDs . . . . .	5
2.2	Initialise the Hardware . . . . .	7
<b>3</b>	<b>Getting Started with the Software</b>	<b>10</b>
3.1	Installation . . . . .	10
3.2	The Update Mode . . . . .	20
3.3	Upload the First Firmware . . . . .	21
<b>4</b>	<b>USBprog Software Reference</b>	<b>23</b>
4.1	The Firmware Pool . . . . .	23
4.2	Configuring a Proxy . . . . .	23
4.3	Graphical User Interface . . . . .	24
4.4	USBprog Command Line . . . . .	26
<b>5</b>	<b>Common Firmwares</b>	<b>33</b>
5.1	Conventions . . . . .	33
5.2	Atmel AVR ISP mkII Clone . . . . .	33
	<b>Bibliography</b>	<b>37</b>

# 1 Overview

## 1.1 The USBprog Hardware

If you read this document, you have probably already an USBprog device. In the past, every microcontroller has its own programming hardware that is mostly relatively expensive. If you work with multiple microcontroller environments, you end up with plenty of programmers on your desk that are not only expensive but also waste space.

On the other side, most self-built programming hardware (for example several ISP programmers for the Atmel AVR controllers) was for the parallel port. However, modern PCs have no parallel port. While you can extend a PC with a parallel port PCI card, you're lost on notebooks. Building and programming USB hardware is not really difficult but still more work than for the parallel port.

The most important thing is the firmware. Once programmed with a so-called *bootloader* the firmware can be exchanged. While it's necessary to have an ISP programming device<sup>1</sup> once to program the USBprog, a normal PC or Mac with the *USBprog software* is sufficient to change the firmware. So it takes only a few seconds to make a JTAG device from an Atmel MTK II clone, for example.

**Warning:** This document only describes the USBprog hardware in version 3.0. If you still have an USBprog 2.0 device, please refer to the online documentation! If you are unsure, look at picture [2.1 on page 5](#).

## 1.2 The USBprog Software

As already mentioned in the section above, you need a special PC software to exchange the firmware of the USBprog. This software is available as both GUI and command line version that can be also used in scripts and Makefiles. For following systems are supported:

- Microsoft Windows 2000, XP, Vista and 7
- Linux (tested on openSUSE and Ubuntu)
- MacOS X (tested on 10.5)

---

<sup>1</sup>That can be a second, already initialised USBprog or another ISP programmer.

## 1.3 About this Document

This documentation should be both: A guide to get started for people that just bought the hardware and want to install the software and use the hardware and also a reference documentation for both the software and the most common firmwares.

If you have any suggestions how to improve the software and the documentation or if you just find problems, please don't hesitate to contact the author via e-mail at [bernhard@bwalle.de](mailto:bernhard@bwalle.de). However, I probably cannot help with hardware or firmware problems. See section 1.5 in that case.

This document is available under the terms of the Creative Commons license " Attribution-Share Alike 3.0 Unported" [3].

## 1.4 Terms and Definitions

At first we introduce the term *firmware* and *bootloader* as we use it in the rest of the document.

**Firmware** The main advantage of the USBprog device is that the firmware can be exchanged. The firmware is the program which is loaded into the flash memory of the USBprog device and which is used to perform a specific task: Programming AVR microcontrollers, emulating a JTAG debugger and so on.

**Bootloader** While the main firmware is exchanged with the computer program, the flash contains also a part (at the highest possible location) that can load the other firmware part. This part is—in the ideal world—only loaded once into the USBprog flash and then never gets overwritten.

## 1.5 Getting Support

If you have problems with the document here, there are several places where you can get support:

1. There's a web-based forum at <http://forum.embedded-projects.net> which is quite well-visited. Also the guy who has developed USBprog, Benedikt Sauter, reads that forum regularly.
2. Additionally, there's also a mailing list at Berlios ([usbprog-pub@lists.berlios.de](mailto:usbprog-pub@lists.berlios.de)). You can subscribe, unsubscribe or read the archive at <https://lists.berlios.de/pipermail/usbprog-pub/>.
3. For general questions about microcontroller programming, <http://www.mikrocontroller.net> is always worth looking at. There are also plenty of USBprog users out there.
4. Especially for problems with the USBprog software, you can also send me an email directly at [bernhard@bwalle.de](mailto:bernhard@bwalle.de).

## 2 Getting Started with the Hardware

### 2.1 Connectors, Jumpers and LEDs

At first we have to introduce the jumpers, connectors and LEDs which we talk about in the next sections. All figures in that document are meant to be drawn in the same perspective (USB connector on the right) as figure 2.1.

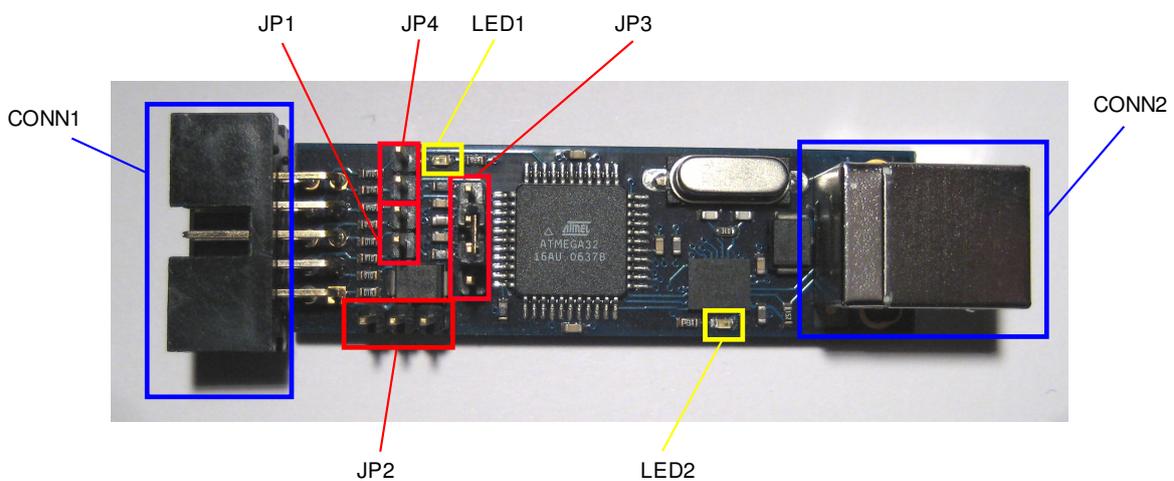


Figure 2.1: The USBprog device (version 3.0) with all connectors, jumpers and LEDs

#### 2.1.1 Connectors

**CONN1** is the output interface which is used to do something useful with the USBprog beside from blinking LEDs. For example if you use the *avrispmk2* firmware, this is an ISP interface which is used to program microcontrollers.

**CONN2** is obviously the USB connector which is used to connect your USBprog to the computer. You have to use a *Type B* cable which is the normal cable to connect USB devices—apart from micro or mini USB.

#### 2.1.2 Jumpers

**JP1** (the lower two of the four pins) is the jumper that must be connected if the bootloader of the USBprog should be updated. See also section 2.2 on [page 7](#) how to flash the bootloader.

**JP2** controls how the power supply of the circuit that should be programmed (connected to CONN1). Figure 2.2 shows the possible connections:

The default is no power supply for the programmed circuit. In that case you must ensure that the device is supplied with power by other means. This is the safest possibility.

If you connect the two leftmost pins, that means that the 5 V VCC is directly from the USB port. This setting is dangerous because an error (short-circuit) in your circuit can damage the computer.

An alternative is the setting of the two leftmost pins: In that case, the 5 V VCC is not directly from the USB port but with a Schottky diode. This is safer than the direct connection.

**JP3** has two functions: At first it can be used as 5 V serial interface for debugging. You cannot directly connect this jumper to the computer but you need some level converter in between<sup>1</sup>. This functionality is only needed by firmware developers.

More important is another function which is used by the bootloader at startup. To put the device in update mode, disconnect the device, then connect pins 2 and 3 as shown in figure 2.3 and connect the device again.

**JP4** is application specific, i. e. used directly by the firmware that is flashed into the USBprog. Currently, JP4 is not used by any of the public firmwares.

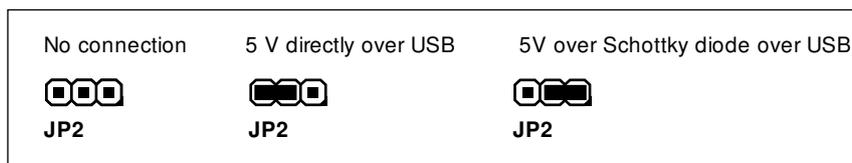


Figure 2.2: Possible connections of JP2

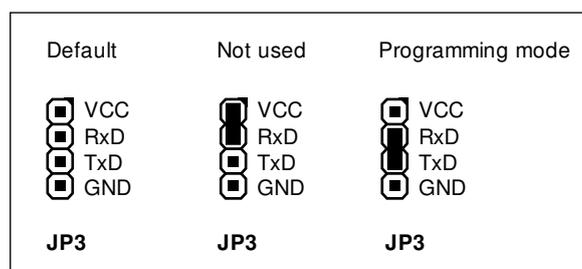


Figure 2.3: Possible connections of JP3

<sup>1</sup>If you want to build such a level converter yourself, look for example at <http://www.nslu2-linux.org/wiki/HowTo/AddASerialPort>. If you want to buy such a cable in Germany, the [Embedded Project Shop](#) also has one. Look for "FTDI-Kabel TTL-232R USB zu TTL serielles Kabel (5.0 V)".

### 2.1.3 LEDs

**LED1 (red)** is used by the firmware. There are two common scenarios:

If the USBprog is in *update mode*, the LED blinks slowly. In the wide-spread Atmel MTK II clone firmware, this LED shines while the programming of the microcontroller is ongoing.

**LED2 (green)** is just the Power LED.

## 2.2 Initialise the Hardware

This section describes how to install the bootloader. If you bought a version of USBprog which already contains the bootloader, you can safely skip that step. If you don't know if you have a version with bootloader, please read section 2.2.1.

### 2.2.1 Check if the Bootloader is Installed

To check if the device already contains a bootloader, just put the device into programming mode manually. For this step, you don't require any specific PC software. You just need a working USB port.

1. Disconnect your USBprog from the computer if it's connected.
2. Close the jumper JP1 as described in section 2.1.2 on page 5.
3. Connect the USBprog to a USB port.

If LED1 (see figure 2.1 on page 5) now blinks, everything is fine. If not, there's no working bootloader and you need to flash it.

### 2.2.2 Flashing the Bootloader

When buying a USBprog at the [Embedded Projects Webshop](#) you can choose between a slightly cheaper version without an already flashed bootloader and a slightly more expensive version that already has flashed the bootloader.

Flashing the bootloader is the only step where you need a second programming device. It's possible to use an already initialised USBprog with the "AVRISP mk2 Clone" firmware flashed. Any other programmer for Atmel AVR microcontrollers will do it, too. You need the standardised *ISP* interface.

We need not only a programmer but also a programming software. While there are plenty of programmers for Atmel AVR microcontrollers, we use [AVRdude](#) because it supports quite a lot of hardware devices and it's available on basically any platform that is around there. On Windows, we suggest to use [WinAVR](#) which is quite easy to install.

## Setting the Jumpers

To program the microcontroller that is on the USBprog board, you have to connect jumper JP1 as described in section 2.1.2 on page 5.

There are two possibilities how the programmer gets its power:

1. The programmer has its own power supply. That is the case for every USB-based programmer since USB can supply the device with up to 500 mA.
2. The programmer takes the power from the circuit that should be programmed. That is the case for most “self-made” parallel port programmers since the parallel port is not able to supply devices with power.

If you have a device of the second category, you have to set the jumper JP2 as described in section 2.1.2 on page 5 and shown in figure 2.3 on page 6. We suggest the 3rd method with the Schottky diode.

## Wiring

After the jumpers are right, connect the USBprog with a free USB port of your computer which is necessary to supply the USBprog with power. The green power LED (LED1, see section 2.1.3 on the preceding page) indicates that everything is okay. If necessary connect your programmer with power and with your computer.

For the connection between your programmer and your computer you need a 10-pole ribbon cable. The pin assignment is the standardised ISP interface which is described in section 5.2.2 on page 33.

## Flashing the Firmware

Now it's time to flash the firmware with AVRdude. At first you have to find out how your programmer is named in AVRdude which is listed in [1] where the `-c` option is described. This symbolic name is spelled as PROGRAMMER in the commands below.

At first you have to download the firmware file at [http://svn.berlios.de/svnroot/repos/usbprog/trunk/usbprog\\_base/firmware/usbprog\\_base.hex](http://svn.berlios.de/svnroot/repos/usbprog/trunk/usbprog_base/firmware/usbprog_base.hex). Save the file as `usbprog_base.hex`.

Now flash the firmware with the command

```
% avrdude -p m32 -c PROGRAMMER -U flash:w:usbprog_base.hex
```

The second step requires to set the fuse bits with following interactive AVRdude session:

```
% avrdude -p m32 -c PROGRAMMER -t
avrdude> write lfuse 0 0xe0
avrdude> write hfuse 0 0xd8
avrdude> quit
```

After everything was successful, disconnect the USBprog, remove the jumper JP1 and connect the USBprog again. The red LED blinks now. This is a sign that you're ready to upload a firmware which is described in [section 3.3 on page 21](#)—just the next section!

## 3 Getting Started with the Software

### 3.1 Installation

#### 3.1.1 Binary Packages for Linux

This section describes the installation of binary packages on wide-spread Linux distributions. If you have a more exotic Linux distribution, another Unix flavour or if you just want to use the latest and greatest version, proceed by reading section [3.1.4 on page 16](#).

In any case: After the installation has finished, the program can be started using `usbprog` in a shell for the command line version of the USBprog software and `usbprog-gui` for the graphical user interface (if installed). In the last case, USBprog should also appear in the start menu of your desktop environment<sup>1</sup>, at least after logging out and logging in again.

#### Ubuntu/Debian

That's the easiest distribution because Uwe Herrman ([uwe@debian.org](mailto:uwe@debian.org)) was so kind to provide Debian packages. Because everything that is in Debian is also in *Universe*, you also have that advantage on Ubuntu.

So: Just open a Terminal and install USBprog by entering

```
% sudo aptitude install usbprog
```

If you also want to use the graphical interface of the USBprog software, use

```
% sudo aptitude install usbprog usbprog-gui
```

The only “disadvantage” of installing the GUI is that it probably will install some dependencies like `wxGtk`.

If you want to access the USBprog as user, that user has to be put in the `plugdev` group. To do so, just execute (replace `@@USERNAME@@` with the username that you want to put into that group):

```
% sudo usermod -aG plugdev @@USERNAME@@
```

---

<sup>1</sup>At least if you use KDE, GNOME or Xfce.

## openSUSE and SLES

There are binary packages in the [openSUSE Build Service](#), directly from the author. So they should be always up to date. They are in the *electronics* repository. To add the repository to your package list, open a shell and then enter

```
% sudo zypper ar -r http://repos.opensuse.org/electronics/@@DIST@@/electronics.repo
```

The term @@DIST@@ has to be substituted version of your openSUSE or SLES as shown in table 3.1.

String	Distribution
openSUSE_11.0	openSUSE 11.0
openSUSE_11.1	openSUSE 11.1
openSUSE_11.2	openSUSE 11.2
SUSE_Linux_Factory	openSUSE Factory (the unstable development version)
SLE_10	SUSE Linux Enterprise (Desktop/Server) 10
SLE_11	SUSE Linux Enterprise (Desktop/Server) 11

Table 3.1: SUSE Distribution strings for the *electronics* Build Service repository

After that, refresh the package list with

```
% sudo zypper ref
```

You're now ready to install USBprog:

```
% sudo zypper install usbprog usbprog-gui
```

If you don't need the graphical user interface, just omit the `usbprog-gui` package. Of course, since the repository is now known by the package manager, you can also install or uninstall the USBprog packages using YaST.

If you want to access the USBprog as user, that user has to be put in the `uucp` group on SLES and openSUSE until 11.2. Starting with openSUSE 11.2, the group is `dialout`. The group name has been chosen to match the group of the serial devices (`/dev/ttyS*`). See also `/usr/share/doc/packages/usbprog/README.SUSE`.

Just use YaST to perform that task: Start `yast users` on a root shell, select the user with the cursor keys, press F4 (Edit), press Alt-d (Details) press TAB until "Additional Groups" is selected on the right, select the group, press SPACE to mark that group and leave YaST by pressing F10 twice.

## Fedora, CentOS and RHEL

You may be surprised, but the binary packages for Fedora, Red Hat and CentOS are also built in the [openSUSE Build Service](#). The only difference of that section compared with the previous section is that we describe `yum` instead of `zypper` here.

There's no command so add a repository in yum. Instead, you have to change to the directory `/etc/yum.repos.d` and download the repository file. In that case, you have to substitute `@@DIST@@` with values of table 3.2.

```
% su
# cd /etc/yum.repos.d
# wget http://repos.opensuse.org/electronics/@@DIST@@/electronics.repo
```

String	Distribution
Fedora_11	Fedora 11
Fedora_12	Fedora 12
CentOS_5	CentOS 5
RHEL_5	Red Hat Enterprise Linux 5

Table 3.2: Distribution strings for Red Hat-like distributions in the Build Service

Since the refresh is done automatically by yum, just install the package(s) with

```
# yum install usbprog usbprog-gui
```

As with SUSE and Ubuntu, you can omit the `usbprog-gui` package if you don't plan to use the graphical user interface.

If you want to access the USBprog as user, that user has to be put in the `uucp` group on RHEL/CentOS 5 or in the `dialout` group on recent Fedora versions. Just execute

```
% sudo usermod -aG uucp @@USERNAME@@ # RHEL 5, CentOS 5
% sudo usermod -aG dialout @@USERNAME@@ # Fedora > 10
```

where `@@USERNAME@@` has to be replaced with the login of the user that should be in that group. See also `/usr/share/doc/usbprog/README.{CentOS,Fedora,RedHat}`.

### 3.1.2 Windows

#### Supported Systems

This section describes the installation of USBprog on Windows. Although binary packages are provided, the installation is not trivial since we need to install a device driver—unlikely the other operating systems that are able to handle USB devices in userspace without driver installation.

At first, make sure that you're running one of following operating systems:

- Windows 2000 with SP4
- Windows XP with SP3 (32 bit only)
- Windows Vista with SP1 (32 bit or 64 bit)<sup>2</sup>

<sup>2</sup>I was not able to test Windows Vista since I don't have it. The problem that prevented other USB devices from working after the installation of the USBprog software should be resolved at Windows 7 also was affected and Windows 7 works now.

- Windows 7 (32 bit or 64 bit)

Although the installation package is prepared to run on a 64 bit Windows, I was not able to test it. If you are successfully able to use it, please drop me a [note](#).

## Getting the Software

**Note:** Please install the software *before* attaching the hardware the first time. That makes it easier for you. If you already attached the hardware, abort the driver installation process.

You can download the latest Windows installer package at following site:

➡ [http://developer.berlios.de/project/showfiles.php?group\\_id=7642](http://developer.berlios.de/project/showfiles.php?group_id=7642)

There's only one installer package for both 32 and 64 bit Windows. While the userspace program is always 32 bit, the driver is available as 32 bit and 64 bit version. Windows (like Linux) can handle 32 bit software on x86-64<sup>3</sup>, but drivers have to be 64 bit.

## Installation of the Software

Installation of the *userspace software* (that excludes the driver part) is as easy as installing any other Windows program. Just start the installer with a double click, confirm the license, confirm the installation path and you're done.

To start the graphical user interface, select "Start" → "USBprog" → "USBprog GUI". The command line interface has the name "USBprog Commandline".

To uninstall the software, just use the "Uninstall" entry in the start menu or go to "Control Panel" → "Software".

## Driver Installation on Windows 2000 and XP

After you downloaded the software, the driver is located at `c:\Program Files\USBprog\driver` on an English Windows. Replace the "Program Files" with something locale-specific like "Programme" in a German Windows installation.

To install the driver, perform these steps. We assume that you successfully installed the bootloader on the device as described in [section 2.2 on page 7](#).

1. Insert jumper JP1 (see [section 2.1.2 on page 5](#)) to put the device into programming mode after power up.
2. Connect the device.
3. After a few seconds, the "New Hardware Wizard" should appear.
4. Don't connect with *Windows Update*—you don't find a driver for the USBprog there.

---

<sup>3</sup>also known as "x64" or "amd64"

5. Choose “Install from a list or specific location (Advanced)”.
6. Select “Search for the best driver in these locations” and tick “Include this location in the search”.
7. Choose “Browse” and select the driver directory mentioned above (c:\Program Files\USBprog\driver).

That’s it. Figure 3.1 on the following page shows screenshots of the dialog (German only). To test your installation, fire up the USBprog software. In the command line mode, devices should list exactly one USBprog device and in GUI mode, the combo box should have also an entry.

## Driver Installation on Windows Vista and 7

The following procedure describes only Windows 7, but Windows Vista should be similar. To install the driver, perform these steps. We assume that you successfully installed the bootloader on the device as described in section 2.2 on page 7.

**Note:** If you have a 64 bit version of Windows Vista or 7, you cannot install unsigned device drivers by default. However, there’s a software called [Driver Signature Enforcement Overrider](#) that should solve that problem. Use the software at own risk—sorry!

Okay, let’s start: At first you need to insert jumper JP1 (see section 2.1.2 on page 5) to put the device into programming mode after power up. Then connect the device. Now Windows looks for a matching driver and doesn’t find one. This procedure takes about one minute<sup>4</sup> and then the dialog shown in figure 3.2 on page 16 appears.

So we have to install the driver manually:

1. Open the “Device Manager”. You find it in “Control Panel” → “Hardware and Sound” → “Devices and Printers” → “Device Manager”.
2. The device “usbprogBase Mode” should appear there with a little warning sign as shown in figure 3.3 on page 17 in the first picture. Select “Update driver”.
3. Windows asks you where you want to search for the driver (figure 3.3 on page 17, second picture). Choose to manually specify the location and select c:\Program Files\USBprog\driver then.
4. A red warning that the driver is not signed appears (figure 3.3 on page 17, third picture). Confirm the driver installation!

That’s it. To test your installation, fire up the USBprog software. In the command line mode, devices should list exactly one USBprog device and in GUI mode, the combo box should have also an entry.

---

<sup>4</sup>at least in my installation in VirtualBox, maybe a native Windows is much faster

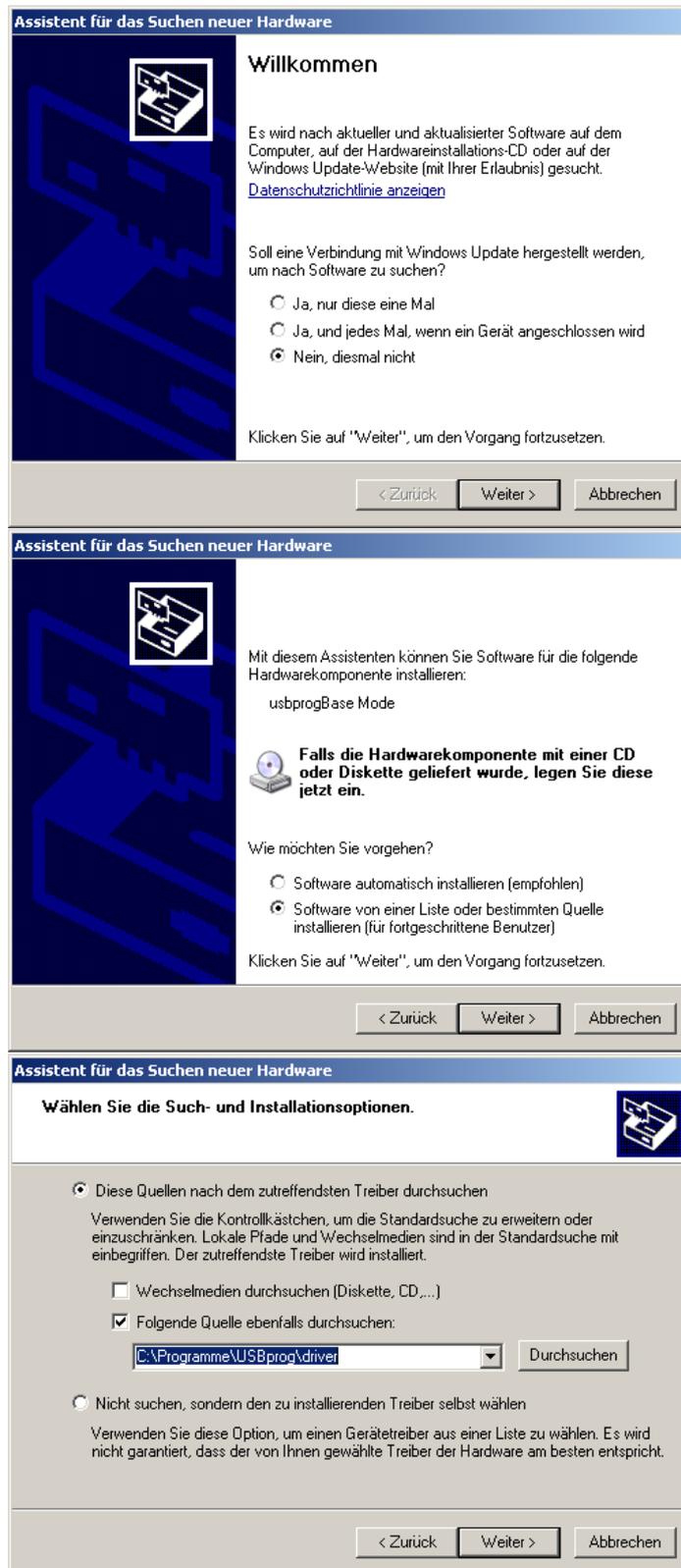


Figure 3.1: Installation workflow on Windows XP (German only)

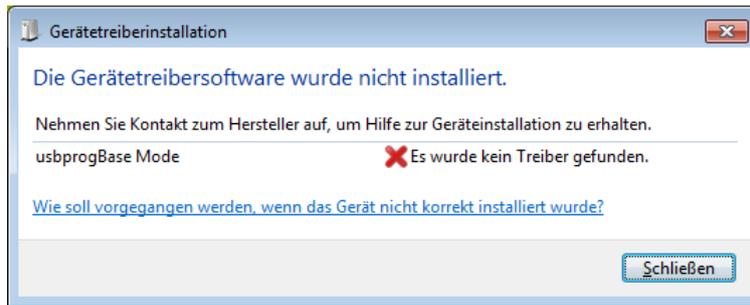


Figure 3.2: No driver has been found (German only)

### 3.1.3 MacOS

Porting Unix software to the Mac is easy, but developing cross-platform software that feels really MacOS-like is hard—especially if you are software developer and not designer. Therefore, we don't provide a software package for the Mac. But nevertheless: It's easy to install the USBprog software on the Mac.

You can compile USBprog directly from the source as described in section 3.1.4. However, it's more easy if you use *MacPorts*. Just install MacPorts on your computer as described in [2].

After that, open a Terminal and execute

```
% sudo port install usbprog
```

After that, you can start the command line tool with `usbprog` in a shell and the GUI with `usbprog-gui`.

### 3.1.4 Compiling USBprog from Source

#### Operating Systems

This section describes how to build USBprog from source. Sometimes there are valid reasons that even “normal” users (and not developers) wish to compile the program from the source code:

- a newer version is available that fixes a bug but the distribution package is too old;
- a more “exotic” Linux distribution is used for which no binary packages are available;
- even another operating system (not Linux or MacOS) is used.

Therefore, that section describes how to compile USBprog on Unix (which includes MacOS). Compiling it on Windows is completely different. Since there is always a binary setup package is available on that platform, the reasons mentioned above don't apply here. If you want to compile USBprog on Windows, refer to the file `win32/README` in the USBprog source distribution.

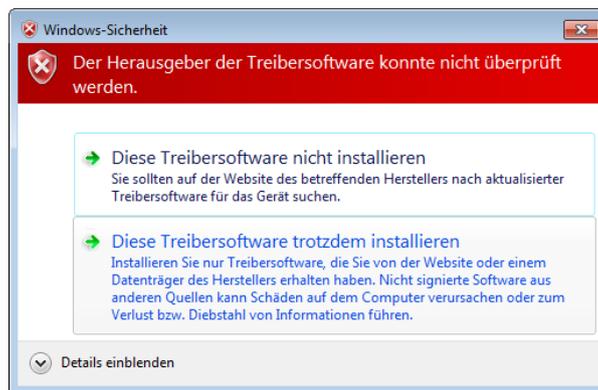
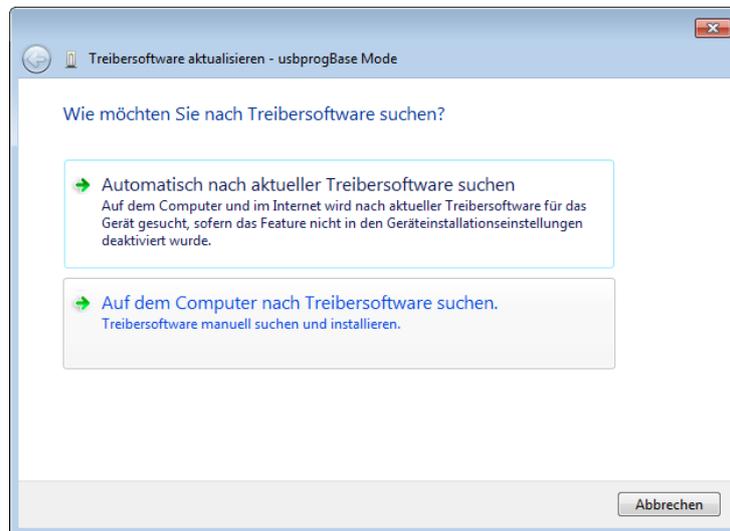
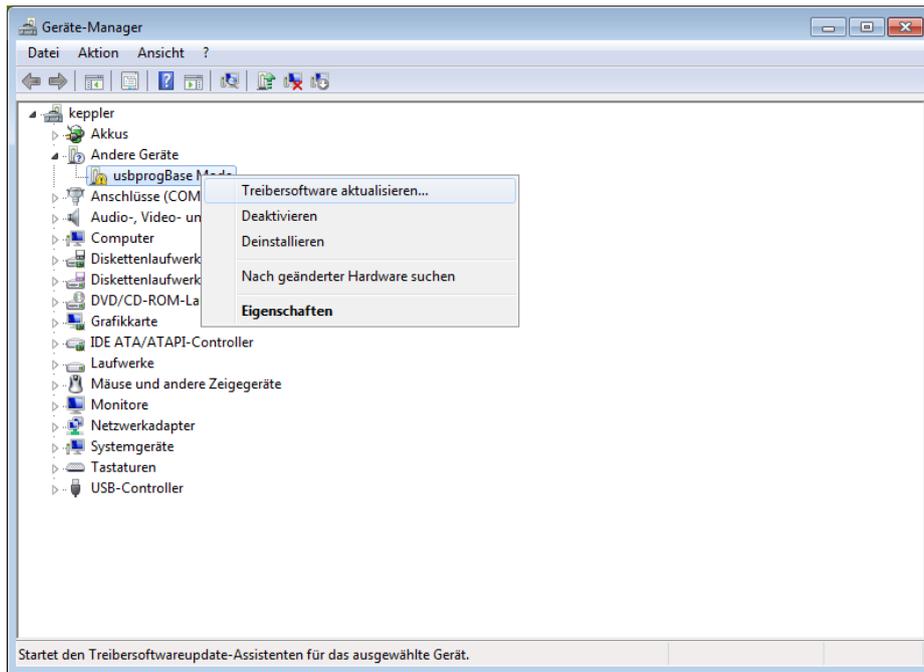


Figure 3.3: Manual driver installation on Windows (German only)

## Dependencies

When compiling software from source, care must be taken to ensure that all dependencies are installed on the computer. The USBprog software needs following other software components. If you use distribution packages to install these dependencies, ensure that the header files (usually in a subpackage `-dev`<sup>5</sup> or `-devel`<sup>6</sup>) are installed, too.

- a working C++ compiler like `g++`;
- Perl (<http://www.perl.org>) to build the manual pages from the POD sources;
- `libusb` (<http://www.libusb.org>);
- `libxml` (<http://www.xmlsoft.org>);
- `libcurl` (<http://curl.haxx.se>);
- GNU Readline (<http://tiswww.case.edu/php/chet/readline/rltop.html>) only if you want to use command-line completion with TAB;
- `wxWidgets` (<http://www.wxwidgets.org>) only if you want to build a graphical user interface.

## Download, Build and Install

After everything is fine, download the source tarball from following source:

➡ [http://developer.berlios.de/project/showfiles.php?group\\_id=7642](http://developer.berlios.de/project/showfiles.php?group_id=7642)

You should have a file called `usbprog-@@VERSION@@.tar.bz2` on the disk where `@@VERSION@@` represents the version of the program. At first, extract that tarball and change into the newly created directory afterwards:

```
% tar xvfj usbprog-@@VERSION@@.tar.bz2
% cd usbprog-@@VERSION@@/
```

Now configure the software using

```
% ./configure
```

Lots of text messages appear. If the last lines look like

```
-----
                Build summary
-----
Readline          : enabled
GUI (wxWidgets)  : enabled
-----
```

---

<sup>5</sup>on Debian/Ubuntu

<sup>6</sup>on RPM-based distributions like Red Hat or SUSE

everything is fine. Of course “disabled” is also okay, but then the specific feature is missing!

Build and install the software with

```
% make
% sudo make install # or su -c "make install"
```

You can now start the software with `usbprog` (the CLI variant) or `usbprog-gui` (the graphical user interface, if built) and should also find it in the menu of your desktop environment. Also, a manual page should be available, both *usbprog(1)* and *usbprog-gui(1)*.

## Hardware Access for Normal Users

**Note:** This applies only to Linux. For other operating systems please refer to the documentation of the system vendor or simply stick to `root`.

By default, only the `root` user has “raw” access to USB devices. Since it’s not desirable to start USBprog—especially the graphical user interface—always as `root`, we have to change system configuration in a way that it allows non-`root` users the access to the device.

Whenever a new device is detected on the system, the kernel asks a superspace daemon called *udev* to create a new device node below `/dev`. For USB devices, the raw device file (that gets accessed by *libusb* applications like USBprog) is below `/dev/bus/usb`. The permission of that device file determines which user has access to that device file.

A simple solution is now to create a *udev rule* that gives that device file not `root:root` but `root:@@GROUP@@` with write permission for that group. All users that should be allowed to use USBprog can not be put in `@@GROUP@@`.

A template for that *udev rule* is distributed with the USBprog source distribution called `usbprog.rules.in`. Replace the term `@@USBPROG_GROUP@@` with the name of the group you want to use<sup>7</sup> and copy the file in `/etc/udev/rules.d`. Since the lexicographic order in that directory determines the priority (processing order) of rules, it’s quite common to prefix it with a number. Example:

```
% sed -e 's/@@USBPROG_GROUP@@/plugdev/g' usbprog.rules.in \
> /etc/udev/rules.d/98-usbprog.rules
```

Of course “modern” Linux distributions (especially Red Hat and SUSE flavours) offer more complicated solutions to give the current “desktop” user more permission to the devices. However, that solutions are not standardised, tend to change every year and sometimes even are poorly documented. Therefore, we don’t describe that here. For the average user, this group-based solution is sufficient.

However: You don’t want to use that solution in a centrally administered company network. Contact your system administrator instead. :-)

---

<sup>7</sup>You can create a new group called `usbprog` or you can simple re-use an existing group like `plugdev` on Ubuntu for storage devices.

## 3.2 The Update Mode

### 3.2.1 How USB Works

#### USB Device Identification

You don't know much about USB to use USBprog. However, one fact is important to understand the limitations of the USBprog especially on Microsoft Windows: How a USB device is identified.

Whenever you connect a USB device to the computer, the USB *host* (the computer) asks the USB *device* a few things. This process is called *enumeration*. After the enumeration process has finished, the USB host assigns the USB device an address. On Linux, this process can be watched in the kernel log with the `dmesg` command:

```
usb 6-1: new full speed USB device using uhci_hcd and address 2
usb 6-1: New USB device found, idVendor=1781, idProduct=0c62
usb 6-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 6-1: Product: usbprogBase Mode
usb 6-1: Manufacturer: USBprog EmbeddedProjects
usb 6-1: configuration #1 chosen from 1 choice
```

As already mentioned in section 1.4 on page 4, the USBprog device has a special *bootloader* that allows the *firmware* to be updated without another device. When you connect the USBprog to the computer, the device normally starts the firmware. If you have flashed for example the `avrispmk2` firmware, then the device identifies as “Atmel AVR ISP mkII”<sup>8</sup>.

**Important:** Although the USBprog is always the same hardware device, depending on the operation mode and the firmware, it is seen as totally different devices from the computer's point of view. It can even change its identity at runtime without re-attaching!

#### USB Drivers on Windows

Two important things have to be considered when using USB Devices on Microsoft Windows:

- It *does* matter on which USB port a device is connected for a driver. So if you connect the device to a different USB port, the driver installation process has to be repeated. However, since Windows knows the driver from the last time, everything goes automatically—which doesn't Windows prevent from displaying lots of dialogues.
- You need a device driver<sup>9</sup> for each device.

---

<sup>8</sup>USB Vendor ID: 0x03eb, USB Device ID: 0x03eb

<sup>9</sup>One exception are HID drivers which can be accessed from a userspace application.

While the first fact is only a bit annoying, the second fact matters when using the USBprog: Since the device can change its identity at runtime, after the identity changes, a new device driver has to be installed. And it's not possible to access the USBprog device while it emulates another device like the Atmel AVR ISP mkII.

However: In normal operation, that's not a problem. I just wanted to explain that fact so that you understand why the manual switch to the update mode is necessary.

### 3.2.2 Switching to the Update Mode

Whenever the firmware (not the bootloader) of the USBprog needs to be updated, the device must be switched from normal operation mode to update mode. There are two ways of putting the USBprog in update mode:

1. Sending some magic stuff to the device that puts the USBprog in update mode at normal operation runtime.
2. Disconnecting the device, setting jumper JP1 (see section [2.1.2 on page 5](#)) and connecting the device again.

While the first variant seems to be easier (especially when developing a new firmware where you have to re-flash the USBprog every few minutes), it clearly has its disadvantages: It doesn't work on Windows for the reasons already explained and it sometimes doesn't seem to work reliably at all.

Therefore: As ordinary user, we always recommend to use the jumper to switch to update mode! After programming, the USBprog automatically switches from update mode to normal mode<sup>10</sup>. So don't forget to remove the jumper to prevent the USBprog from starting in update mode at next time you connect the USBprog again or the computer boots.

To detect if the firmware is in update mode—you may already have noticed it—the red LED flashes in a sequence “flash, flash, pause, flash, flash, ...”.

## 3.3 Upload the First Firmware

We describe the command line interface here as “first step” for a simple reason: It's exactly the same on every platforms and a command can be reproduced by the user more easily than finding GUI elements.

So open a shell on Unix (Linux/MacOS) or open the “USBprog Commandline” in the “Start” menu on Windows. You should see a prompt that looks like

```
(usbprog) _
```

---

<sup>10</sup>yes, even on Windows

Now connect the USBprog device with the JP1 (update mode) jumper set. After you connected the USBprog to the USB port of your computer, wait a few seconds to make sure that the operating system detected the device.

Back to the USBprog software, the devices command should display a single USBprog device:

```
(usbprog) devices
[ 0] * Bus 003 Device 002: 1781:c620
      usbprog: USBprog in update mode
```

Since it's the only device and since the device is already in *update mode*, the star (\*) after the number shows you that this device is already selected. So you can proceed and upload the `blinkdemo` firmware. That firmware does nothing but blinking the LED. Blinking the LED you ask? It's already blinking! Well, the sequence is different, so it should be easily possible to distinguish the `blinkdemo` blinking from the update mode blinking.

Enough talking. Let's now upload the firmware:

```
(usbprog) upload blinkdemo
Opening device ...
Writing firmware ...
#####
Starting device ...
Detecting new USB devices ...
```

Okay, that's it! The next chapters describe both the GUI and the command line in detail. If you then want to do something real with your USBprog like programming an AVR microcontroller, read [chapter 5 on page 33](#).

## 4 USBprog Software Reference

### 4.1 The Firmware Pool

#### 4.1.1 Overview

Both the GUI and the command line use the firmware pool available in the internet. Therefore, you need an internet connection to use USBprog, at least at the first time. Since the firmware pool is cached on the disk, you can also use USBprog without an internet connection later.

To get the list of available firmware files, USBprog will connect to the internet at startup and download the file <http://www.ixbat.de/usbprog/versions.xml>. So if you run some “personal fire-wall” software on Windows and that software asks you if it’s okay to connect to the internet, that’s because of that file.

Right of the label “Online Pool”, there’s a combo box with all the firmwares available at the firmware pool. Before uploading a firmware to the USBprog device, you probably want to get some information for the firmware first.

#### 4.1.2 Location of the Cache

For the curious, here’s the location of the firmware cache where also the `version.xml` file is downloaded:

- **Linux/MacOS:** `$HOME/.usbprog`
- **Windows:** `%APPDATA%` (just enter that in the Explorer and it shows you the directory in your Windows installation), for example:  
`c:\Documents and Settings\bwalle\Anwendungsdaten`

### 4.2 Configuring a Proxy

The USBprog software uses *libcurl* for internet access. Therefore, it honours the environment variable `http_proxy` that simply can be set to the hostname of the proxy. Read the manpage *curl(1)* and the “ENVIRONMENT” section in specific for more information.

In addition to that settings, the Windows version of USBprog also honours the Internet Explorer proxy settings. However, also on Windows, the environment variables overwrite the Internet Explorer settings. If you are in doubt, set the environment since that’s better documented and supported directly by *libcurl*.

## 4.3 Graphical User Interface

The installation instructions in section 3.1 on page 10 explain also how to start the USBprog GUI. After the program has started, a window like figure 4.1 on the following page should appear. Of course the GUI looks different on the different operating system, but since the program is built from the same sources using the cross-platform toolkit *wxWidgets* the differences should be small.

### 4.3.1 Getting Information for a Firmware

Some very common firmwares are described in chapter 5 on page 33 in that documentation. However, for any firmware available, you can display some brief information. Just select the firmware in the combo box and choose “Firmware” → “Info” or press F2. A dialog like figure 4.2 on the following page appears.

If you want to get the pin assignment, choose “Firmware” → “Pins” or press F3. A dialog like figure 4.2 on the next page appears.

### 4.3.2 Cache Management Functions

If you want to have all firmware files available for offline usage, just choose “Cache” → “Download all”.

For cleanup, there are two different entries:

- The “Clean” function only deletes unused firmware files that are still in the cache but are useless: Files that are not referenced in the index file (`versions.xml`) any more. This can be old versions of firmwares still in the index file or removed firmwares.
- The “Delete cache” function removes all firmware files from the cache.

### 4.3.3 Uploading a Firmware

Uploading a firmware from the online pool is easy:

1. Connect the device in update mode (see section 3.2.2 on page 21).
2. Select a device in the “Device” list. Maybe you need to click on “Refresh” first.
3. Select a firmware in the “Firmware” list.
4. Click on “Upload”.

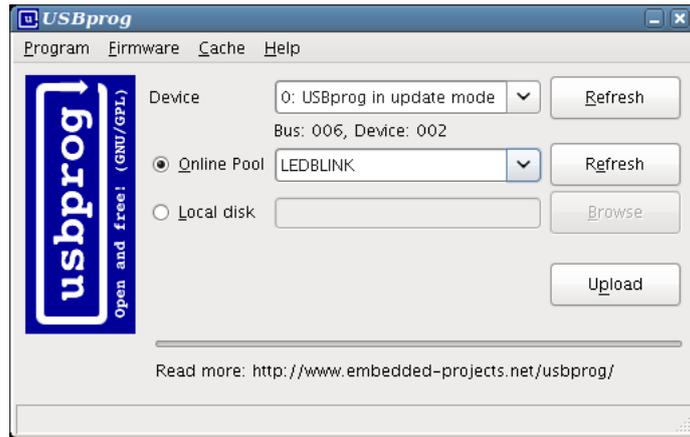


Figure 4.1: Screenshot of USBprog on Linux



Figure 4.2: Information about the avrispmk2 firmware

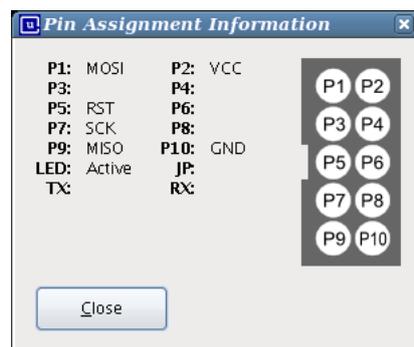


Figure 4.3: Pin assignment of the avrispmk2 firmware

The progress bar should tell you that there's a upload in process and the string "Upload successful" should show up in the status bar. The software also starts the firmware immediately. Don't forget to remove the JP1 jumper!

If you want to upload a firmware that is not in the cache, click on the radio button "Local disk" and use "Browse" to select a full path to the firmware file (binary format). The rest is the same.

### 4.3.4 Debugging

If something doesn't work and you need support from the author (or any other party listed in section 1.5 on page 4), a logfile is always helpful. To write a log file in the GUI, click on "Program" → "Logging" and select an appropriate path.

You can also start the application in logging mode:

```
% usbprog-gui -D
```

This logs to standard error. To redirect that to a file, just use the redirection operator of the shell (Unix only):

```
% usbprog-gui -D 2> /path/to/logfile.log
```

## 4.4 USBprog Command Line

### 4.4.1 Overview

Why use the command line version if there's a GUI (section 4.3 on page 24)? There are several reasons:

- The command line version is available everywhere (for example on RHEL there's no *wx-Widgets* package, therefore it's much more work to get the GUI running).
- The command line version can be scripted.
- Some people like me prefer CLIs<sup>1</sup> over GUIs. :-)

Because the GUI and the command line use the same library, they should share all bugs and features. They also share the cache.

The USBprog command line client works like a shell. Each command takes a fixed number of arguments. If the argument is not specified, the shell asks for it:

```
(usbprog) helpcmd  
command> _
```

Also, some commands can be accessed via multiple names:

---

<sup>1</sup>Command Line Interface

```
(usbprog) helpcmd bla
Invalid command: bla
(usbprog) ? bla
Invalid command: bla
```

The best: If compiled with readline support which is the case for all binary packages listed in section 3.1.1 on page 10, TAB completion works like you would expect it.

## 4.4.2 Getting Help

Because a CLI cannot be “self-explanatory”, there are several ways to get a quick help beside from that document. The command

```
(usbprog) help
```

displays a list of all available commands. To get more help for a specific command, use `helpcmd COMMAND` or `? COMMAND`. For example:

```
(usbprog) ? pin
Name:          pin
Aliases:       pins
Argument:      firmware

Description:
Prints a list about pin usage. This might help you when connecting
something to your USBprog.
```

Last, but not least, the manual page `usbprog(1)` briefly lists all commands and invocation parameters.

## 4.4.3 Exiting

Well, it's quite obvious, but since the documentation should be complete, there are three ways to exit the shell:

1. The command `exit`.
2. The alias `quit`.
3. Sending a end-of-line (EOL) with `Ctrl-D` on Unix and `Ctrl-Z` on Windows.

#### 4.4.4 Getting Information for a Firmware

To view a list of firmware files in the pool (see section 4.1.1 on page 23 for an explanation of the firmware pool), use the command `list` like here:

```
(usbprog) list
JTAGICE2      [ ] JTAGICE2
XSVF Player   [ ] XSVF Player
at89prog      [*] at89prog
avrispmk2     [*] AVRISP mk2 Clone
blinkdemo     [ ] LEDBLINK
openocd       [*] OpenOCD Debugger
simpleport     [*] SimplePort
usbprogrs232  [*] usbprogRS232

*: Firmware file downloaded
```

The star (\*) shows if the firmware is already downloaded for offline usage. To display information about a specific firmware, use `info FIRMWARE` and `pins FIRMWARE`:

```
(usbprog) info blinkdemo
Identifier    : blinkdemo
Name         : LEDBLINK
URL          : http://www.ixbat.de/usbprog/
File name    : blinkdemo.bin
Author       : Benedikt Sauter
Version      : 1 [2007-08-13]
Device ID(s) : Vendor: 0x1781, Product: 0x0c62, BCDDDevice: 0x0001
```

##### Description

A simple LED blink demo.

For information about the Pin assignment, use the "pin blinkdemo" command.

```
(usbprog) pins blinkdemo
+-----+
| 9 7 5 3 1 |
| 10 8 6 4 2 |
+-----+

[ P1]          [ P2]
[ P3]          [ P4]
[ P5]          [ P6]
[ P7]          [ P8]
[ P9]          [ P10]
[ LED] Blink LED
```

## 4.4.5 Uploading a Firmware

Before you can upload a firmware, you must select a device first and—contrary to the GUI—you must download the firmware file to the cache.

### Selecting a Device

The `devices` command gives you a list of available updatable devices:

```
(usbprog) devices
[ 0] * Bus 006 Device 004: 1781:c620
      usbprog: USBprog in update mode
```

In that case, the device is already selected as update device. This is the case if there's only one device available and if that device is already in update mode (see section 3.2.2 on page 21). Otherwise, select the device with the `device NUMBER` command as update device:

```
(usbprog) device 0
```

### Downloading the Firmware

The `list` command shows you with the missing star (\*) that the `blinkdemo` is not downloaded. Just download the firmware with

```
(usbprog) download blinkdemo
#####
Firmware blinkdemo has been downloaded successfully.
```

### Uploading the Firmware

It's recommended that you manually put the hardware in update mode as described in section 3.2.2 on page 21. If that's not done and you selected another device as update device, the software tries to put the device in update mode first<sup>2</sup>.

In any case, just upload the firmware with `upload FIRMWARE`:

```
(usbprog) upload blinkdemo
Opening device ...
Writing firmware ...
#####
Starting device ...
Detecting new USB devices ...
```

As the output indicates, the firmware starts immediately—in that case, the LED blink frequency changes.

---

<sup>2</sup>that does not work on Microsoft Windows

## Starting a Firmware

If you now disconnect the device, leave the JP1 jumper connected and connect the device again, the device again shows in *update mode*. However, we know that we flashed the `blinkdemo` firmware. We could now remove the jumper and start it, or flash the firmware again. But we could also use the `start` command:

```
(usbprog) devices
[ 0] * Bus 006 Device 006: 1781:c620
      usbprog: USBprog in update mode
(usbprog) start
Device successfully started.
```

The firmware starts, we see the blink frequency changing!

## 4.4.6 Managing the Cache

### Downloading Everything

Of course it's also possible to download every firmware file with the command line interface. Just use `download all`:

```
(usbprog) download all
Downloading at89prog ...
#####
Downloading AVRISP mk2 Clone ...
#####
...
```

### Cleanup

A cache cleanup can be done with the `cache OPERATION` command. It has two functions:

- If you just want to delete unused files, execute `cache clean`. That can be firmware files that are not referenced by the index file any more because they have been removed or the version has been increased.
- If you want to delete the full cache, use `cache delete`.

This example shows the effect of the last command:

```
(usbprog) list
JTAGICE2      [ ] JTAGICE2
XSVF Player  [ ] XSVF Player
at89prog     [*] at89prog
avrispmk2    [*] AVRISP mk2 Clone
blinkdemo    [*] LEDBLINK
openocd      [*] OpenOCD Debugger
```

```

simpleport    [*] SimplePort
usbprogrs232 [*] usbprogRS232

(usbprog) cache delete

(usbprog) list
JTAGICE2     [ ] JTAGICE2
XSVF Player  [ ] XSVF Player
at89prog     [ ] at89prog
avrispmk2    [ ] AVRISP mk2 Clone
blinkdemo    [ ] LEDBLINK
openocd      [ ] OpenOCD Debugger
simpleport    [ ] SimplePort
usbprogrs232 [ ] usbprogRS232

```

As you see, after the `cache delete` operation, no firmware is in the cache any more.

#### 4.4.7 Offline Mode

While the GUI starts in offline mode automatically when it detects no working internet connection, the command line client doesn't do that. If you don't have an internet connection, the program refuses to work:

```

% usbprog
Error: CURL error: Couldn't resolve host 'www.ixbat.de'

```

However, you can activate the offline mode with the `-o` or `--offline` flag:

```

% usbprog -o
WARNING: You're using usbprog in offline mode!
(usbprog) _

```

#### 4.4.8 Expert Options

##### Debugging

To get debugging output on standard error, use `-D` or `--debug`. You can redirect that debugging output to a file using the standard shell feature:

```

% usbprog --debug 2> debug.log
(usbprog) _

```

##### Alternative Data Directory

Although section 4.1.2 on page 23 mentions that the cache is stored in a fixed directory location, that location can be overwritten with `-d DATADIR` or `--datadir DATADIR`.

## 4.4.9 Batch Mode

As already mentioned, you can use `usbprog` not only in interactive shell mode but also in batch mode. Just specify the commands as arguments of `usbprog` like in that example:

```
% usbprog devices
No devices found.
```

Since the number of arguments for a command is always fixed<sup>3</sup>, you can specify multiple commands without a separator:

```
% usbprog device 0 download blinkdemo upload blinkdemo
===> device 0

===> download blinkdemo
Firmware blinkdemo has been downloaded successfully.

===> upload blinkdemo
Switching to update mode ...
Opening device ...
Writing firmware ...
Starting device ...
Detecting new USB devices ...
```

To be able to read the error messages correctly, if specified multiple commands, the `usbprog` program adds the command before showing the output, prefixed with `===>`.

---

<sup>3</sup>That's why `help` doesn't take a command as argument and you have to use either `?` or `helpcmd`.

# 5 Common Firmwares

## 5.1 Conventions

When describing pinouts, the we use always the pin numbering scheme in figure 5.1.

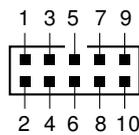


Figure 5.1: Pin numbering used in that document

## 5.2 Atmel AVR ISP mkII Clone

### 5.2.1 Purpose

The *AVR ISP mkII* is a device from Atmel to program AVR microcontrollers. You can view a picture online at following site:

➡ [http://www.atmel.com/dyn/products/tools\\_card.asp?tool\\_id=3808](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3808)

This firmware emulates that device. The advantage of emulating this device rather than inventing an own protocol (like *USBasp*<sup>1</sup>) is that existing programming software can be used. The most common programming tools

- *AVRDUDE* (<http://www.nongnu.org/avrdude/>)
- *AVR Studio* ([http://www.atmel.com/dyn/products/tools\\_card.asp?tool\\_id=2725](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725))

support that hardware device on all operating systems they run on.

### 5.2.2 Pinout

The pinout of the AVR ISP mkII (and of course also of this clone) is the standardised ISP pinout as shown in table 5.1 on the next page.

---

<sup>1</sup><http://www.fischl.de/usbasp/>

Pin	Description	Pin	Description
1	MOSI (data out)	2	VCC (+5 V)
3	–	4	GND (signal ground)
5	RESET	6	"
7	SCK (clock)	8	"
9	MISO (instruction in)	10	"

Table 5.1: ISP Pin Assignment

### 5.2.3 Programming AVR with AVRDUDE

Although this section should be helpful if it's the first time you're using AVRDUDE together with USBprog, this document cannot replace the documentation of AVRDUDE [1]. If you have not already downloaded AVRDUDE, you should do now. On Windows we suggest the excellent *WinAVR* software that can be obtained at

► <http://winavr.sourceforge.net>

Like for the USBprog device itself, you also have to install a device driver for AVRDUDE. When using the `avrispmk2` firmware, the USBprog emulates a different device and so it needs a different driver. The documentation [4] helps here.

If everything is fine and you started the `avrispmk2` firmware on your USBprog, connect it to a microcontroller circuit using the ISP interface and then you can test the hardware and the installation with

```
% avrdude -c avrispmkII -P usb -p m8
avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.01s

avrdude: Device signature = 0x1e9307
avrdude: current erase-rewrite cycle count is 68 (if being tracked)

avrdude: safemode: Fuses OK

avrdude done. Thank you.
```

Don't forget to replace the string `m8` with the type of your microcontroller. If you don't want to consult the documentation, AVRDUDE also gives you a list of valid parts when using an invalid part:

```
% avrdude -p ?

Valid parts are:
x128a4 = ATXMEGA128A4    [/opt/cross/etc/avrdude.conf:14931]
x64a4 = ATXMEGA64A4    [/opt/cross/etc/avrdude.conf:14834]
x32a4 = ATXMEGA32A4    [/opt/cross/etc/avrdude.conf:14737]
...
```

To upload a program called `blink.hex`, just use

```
% avrdude -p m8 -e -U flash:w:blink.hex -y
```

#### 5.2.4 Programming AVR's with AVR Studio

It's a matter of taste whether you use AVRDUDE or AVR Studio to program your AVR's. If you don't use Microsoft Windows, the decision is easier since AVRDUDE runs on basically any operating system while AVR Studio only runs on Windows. The advantage of AVR Studio is clearly that it's integrated in the rest of the IDE, so you don't have to fire a command prompt to program the device.

If you have not already downloaded and installed AVR Studio we suggest to visit following site that contains direct links to the latest AVR Studio versions. You can download the program then without registering at Atmel.

➡ <http://www.mikrocontroller.net/articles/AVR-Studio>

After you installed AVR Studio, don't forget to install the *Jungo* USB driver needed for the AVR ISP mkII device (regardless whether it's the original or the clone). Read the included documentation if you have problems.

If that's done, you should be able to connect with the device using "Tools" → "Program AVR" → "Connect...". Select "AVRISP mkII" in the left list and "USB" in the right list as shown in figure 5.2 on the following page. Now it takes a few seconds and the programming dialog shown in figure 5.3 on the next page appears.

To upload the same `blink.hex` file as with AVRDUDE, click on the "..." in the "Flash" group right of "Input HEX file", select that file and click on "Program" and "Verify".

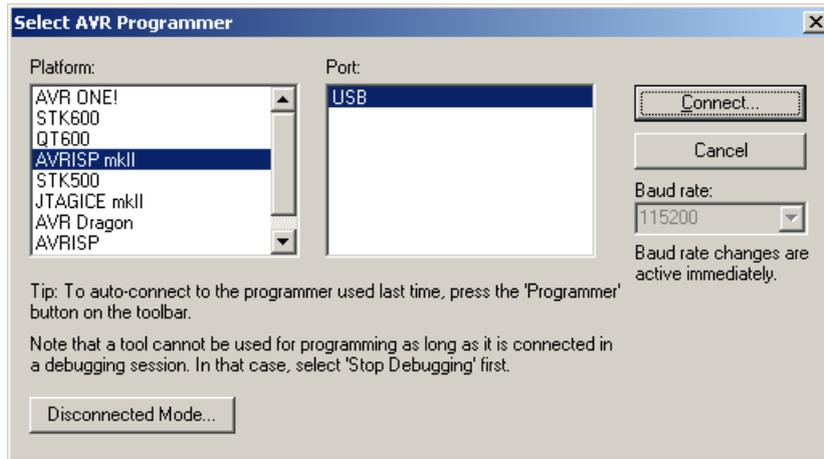


Figure 5.2: Device selection dialog of AVR Studio

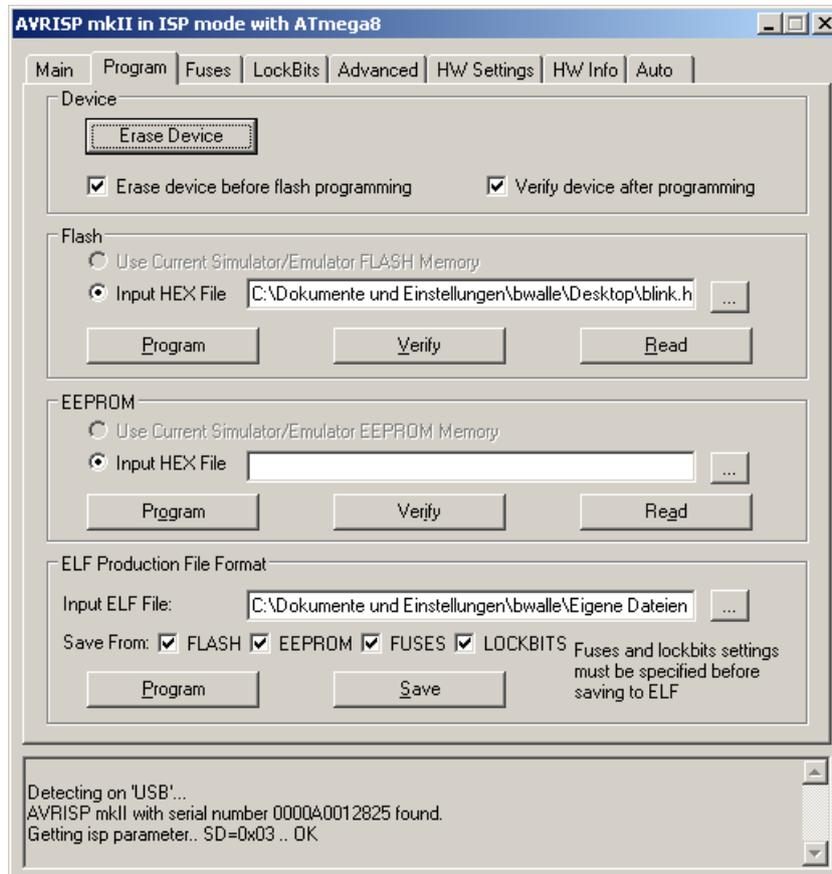


Figure 5.3: Programming dialog of AVR Studio

# Bibliography

- [1] AVRDUDE Manual, <http://www.nongnu.org/avrdude/user-manual/avrdude.html>
- [2] "Installing MacPorts", <http://www.macports.org/install.php>
- [3] Creative Commons license "Attribution-Share Alike 3.0 Unported",  
<http://creativecommons.org/licenses/by-sa/3.0/>
- [4] "WinAVR User's Manual"  
<http://winavr.sourceforge.net/WinAVR-user-manual.html>