software

Draisberghof | Software | USB_ModeSwitch | ModeSwitchForum

No ads - but links to sites I deem valuable and important:
stop-overpopulation.org - Life could be so good if there'd be fewer of us ...
metabunk.org - Remove the bunk from scary claims

This site is powered by green energy; for details see the provider's **page**

USB_ModeSwitch - Handling Mode-Switching USB Devices on Linux

If you don't like the text column width, adjust your browser window

Introduction

USB_ModeSwitch is (surprise!) a mode switching tool for controlling 'multi-mode' USB devices.

More and more USB devices (especially high-speed WAN stuff, based on cell

phone chipsets which are able to change their USB connection mode) have their MS Windows drivers onboard; when plugged in for the first time they act like a flash storage and start installing the driver from there. After installation (and on every consecutive plugging) the driver switches the mode internally, the storage device vanishes (in most cases), and a new device (like an USB modem) shows up. Modem maker "Option" calls that feature "ZeroCD (TM)" since it eliminates the need for shipping a separate driver carrier.

In the beginning, nothing of this was documented in any form and there was hardly any Linux support available.
On the good side, most of the known devices do work in both modes with the available Linux drivers like "usb-storage" or "option" (an optimized serial driver, the Linux standard for high-speed 3G modems).
That leaves only the problem of the mode switching from storage to modem or whatever the thing is supposed to do.

Fortunately there are things like human reason, USB sniffing programs and "libusb". It is possible to eavesdrop the communication of the MS Windows driver, to isolate the command or action that triggers the mode switching and to reproduce the same thing under the rule of Linux or the BSD variants.

USB_ModeSwitch makes this process easy to handle by taking the important parameters from a configuration file and doing all the initialization and communication stuff, with heavy help from "libusb".
It is mainly used automatically - via udev events and rules - to do the switch without any user interaction. But it can also be run as a command line tool, usually when trying to make unknown devices work with it.

This tool is part of most major distributions; you should not be having to install from the source packages here unless you run into problems and need the latest version.

Please read the information on this page carefully before you go around posting questions! If you encounter a new device, it really helps to understand the principle of what is happening, which in turn makes it easier to find out about the switching command and to add a new config entry.

For hints about doing your own sniffing see paragraph **Contribute** below.

Download

Important: For a working installation you need both program and data package

!

Changes and updates to the configuration data may happen more often than new program releases; most of the knowledge about devices is contained in these files. That's why it is provided separately.

- Download **usb-modeswitch-2.2.6.tar.bz2**, the source code release dated from 2015-11-01; a Debian package should be available soon at the **Debian Repository**.
- Download the **usb-modeswitch-data** package (2015-11-01). It contains the device database and the rules file, including full paths. You need program releases from 2.2.2 upward because of newly introduced parameters.
- The optional **device_reference.txt** from 2014-05-29; this is a collection of earlier device setups with their respective contributors; you can use it as a first resource if you want to make a new device working.
- Don't forget **libusb-1.x** (formerly on **libusb.org**) if it's not on your system. In most distributions there is most likely a package named "libusb1-dev" or "libusb1-devel" (or similar).
  Older versions were based on libusb-0.1.12 but since version 2.0.0 only libusb1.x is supported.
  As a small complication, there was a compatible fork of libusb1 called "libusbx" around for a while (**read more**).

If you have an Android Tablet with USB host port and at least Android version 2.2, you can try the "**PPP Widget**", available only on Google Play. It includes USB_ModeSwitch and PPPD, making it relatively easy to go online with your modem stick or your phone, even on WiFi-only devices.

If you are a system integrator (package maintainer) you can use this XML file to check for new releases:
usb-modeswitch-versions.xml.

The maintainer of the USB_ModeSwitch Debian package has set up a PPA providing the most recent release; see his **posting in the forum** for the link.

How to install

If you have an earlier version installed, de-installation is recommended ("make uninstall"). Several file locations changed in 1.1.0, old ones might be orphaned if not taken care of. If you have a version after 1.1.0 you can just update and

overwrite all existing files.

Unpack the source file of the program (who might have thought!). In the newly created directory run as root or superuser:

```
# make install
```

This installs a small shell script for udev, the larger wrapper script, a config file, the man page and the freshly compiled binary.
Important: if you install this way, you will need the "tcl" package for the large dispatcher script. There are more ways to install which do not depend on that package. Refer to the included README for **further install options** !

Now do the same procedure for the **data package**. It will install the config files in "/usr/share/usb_modeswitch" and the udev rule file in "/lib/udev/rules.d". The earlier "/etc/usb_modeswitch.d" is now reserved for custom config files (new or changed).
You are set already; if your device is known, you should be able to just plug it and use it. If it doesn't work right away we'll find out why.

For **manual** use just install the program. Work with the command line interface or use a custom config file. The "device_reference" file (see "Download") is a good starting point to create your own configuration. It is heavily commented and should tell you what to do.
Your custom config file can have any name and place; just tell usb_modeswitch how to find it with the **-c** parameter.

Manual use is intended for testing and analyzing. See next paragraph.

How to use

In most cases, you will be able to use your device without any interaction except plugging it in.

If you think your device is supported, but things are not working out as they should, turn on logging first as described in **Troubleshooting**.

For testing, debugging and taming new devices from the wild, you can use the binary part of USB_ModeSwitch in manual mode.
There are two ways for that: using a config file or using the command line.

Run "usb-modeswitch -h" to list the command line parameters. If any of them except -W, -D, -I and -q are used, a config file given with -c is ignored and all mandatory parameters have to be provided on the command line. See also the included man page.

To work with a config file, use one of the little files in "/usr/share /usb_modeswitch" or create one yourself. Then give the path and file name to usb_modeswitch with the -c option. You also can have a look into the **device_reference.txt** for hints about model families and an explanation of the parameters.

Important: USB_ModeSwitch - like all programs using libusb - has to be run as root (or with "sudo") when calling it manually. Otherwise strange error messages turn up and things won't work. When trying out switching commands and strategies, it's probably easier to work at a root shell for a while ("sudo bash" or "su -").

The automatic approach consists of several components working together, listed in the logical order of usage:

- **/lib/udev/rules.d/40-usb_modeswitch.rules** - the udev rules starting the wrapper if a known device ID (vendor/product) is recognized. To add a trigger for a new modem for which you have a working config file, append a line with its USB ID as seen in the existing entries.
  If the switched device provides standard serial ports, a second rule calls the wrapper again and adds a symbolic link to the **correct** connection port (see below)
- **/lib/udev/usb_modeswitch** - a shell script forking to the real wrapper script. Since version 1.1.6 the script is fully compatible with the "dash" shell used in Ubuntu as well as with older "bash" variants. Recent versions make use of upstart or systemd features in order to detach the usb_modeswitch run from the udev process
- **/usr/sbin/usb_modeswitch_dispatcher** - this is doing additional device checking and then using the binary to switch with the selected device config file.
  If no drivers are taking care of the device after the mode switch, the dispatcher will try to load and bind the "option" serial driver, in order to make the device usable.
- **/etc/usb_modeswitch.conf** - a global config file to enable extensive logging when troubleshooting, or to disable switching alltogether (mostly to access the install part of devices)
- **/usr/share/usb_modeswitch** - a folder containing the individual setup information files per device, named according to the IDs and possibly further identity tokens (to resolve known ambiguities). If your device ID shows up in one of the file names, chances are your device is supported even if the model or brand does not match.
- **/etc/usb_modeswitch.d** - a folder for customized config files. You can put new or modified config files here; they will take precedence over the

collection of configurations in /usr/share/usb_modeswitch.

- **/usr/sbin/usb_modeswitch** - the binary program effectively doing the switch. This is designed to be independent of kernel or system specifics, and should be portable to non-Linux platforms, wherever libusb is available.

After switching and driver-loading, it is the responsibility of the system to discover the new (mostly serial) device.

Current releases of **NetworkManager** (or its **ModemManager** component) are usually quite good at making use of wireless modems. Even newer models that don't use PPP interfaces anymore are recognized and integrated seamlessly.
There may be issues with older versions of these programs which may run into touble when trying to auto-detect USB modems and how to use them.
If you are stuck with an older system, try disabling NM and MM. Good results were reported by working with **wvdial**, **UMTSmon** and several tools providing a user interface to PPP like **kppp**; some of these programs may require a bit of basic knowledge though.

Starting from version 1.1.2, usb_modeswitch will add a symbolic link to the correct port with interrupt transfer if the device provides standard serial ports. The link will have the name **/dev/gsmmodem**, with a number appended if more than one device is attached.
You can use this name with connection helpers like **wvdial**. Note that in many cases you may have to edit the configuration file manually.

If you managed to get a new or badly supported device to switch correctly in manual mode, you can add a udev rule and a config file yourself. But **please** report it back to share it !!

See **Contribute**.

Known working hardware

Very important note:
Personally, I could not test the vast majority of supported devices; the list here - as well as the necessary data - relies on reports from third parties (people, that is). So it may happen that you hit sudden obstacles even with your device listed here.
That said, the user base of this little tool has grown considerably, so that any data related problems are generally surfacing quite soon.

There are hitherto three known methods for initiating the switching process:

1. sending a rarely used or seemingly weird standard storage command (equivalent to those of SCSI) to the storage device ("eject" for example)
2. sending one or more vendor-specific control messages to the device
3. actively removing (detaching) the storage driver from the device (only some early devices)

Again, **if you don't find the name of your device in the list, it may still be supported**.
The important thing is that you find your device's USB ID in the config file folder. Have a look into the latest data package (See **Download**).

Here is the list of devices, together with the respective contributors: **device_reference.txt**.

Troubleshooting

Note: if you still need support after having followed the advice on this page, please use the forum!
E-mail may be used for device/config contributions only - **no** support questions!

Known issues:

- Automatic serial driver assignment will work with the 3G-optimized driver only for kernels from 2.6.27 and up. If you have an older kernel and your modem is not recognized by any driver after mode switching, then the generic "usbserial" driver is used as a fallback.
- There is a problematic handling of devices with ID **19d2:2000** in kernels 2.6.26 to 2.6.28. This affects mostly ZTE devices and makes the "usb-storage" driver ignore the ID. In turn this will prevent proper initialization and may cause switching to fail. There is no other way around this than compiling your own kernel with some tiny edits. See **Kernel related issues** below for details.

For debugging of the automated system integration, edit (as root or su) **/etc/usb_modeswitch.conf** in a text editor and change the line

```
EnableLogging=0
```

to

```
EnableLogging=1
```

This gives you a verbose output of the hotplug activity to **/var/log**

/usb_modeswitch_<device>.

If you're next to certain that you have the right values for your device, followed all the hints (see **Known working hardware**), and USB_ModeSwitch seems to do something run after run but to no effect, there are most likely system issues involved.

The first suspects are **existing system rules** for modems which handle things not quite correctly.
If you own a device with the unswitched ID of **05c6:1000**, it may get a wrong switching command in older systems. There are four different types of switching devices, all with that same ID; in the big distributions they were all treated alike as a model from "Option" (the manufacturer) which is wrong in four out of five cases. There are even cell phones with that ID which wrongly get the same treatment when connected to an USB port.

To fix problems like that you can try to remove rules files from "/lib/udev /rules.d" which contain calls to "modem-modeswitch".
USB_ModeSwitch will do additional checks beside the USB ID and treat all known ambiguous devices in the right way. For example, it will leave unknown devices with the **05c6:1000** ID alone.

Annother notorious candidate is again **19d2:2000**. It may be switched O.K. by an existing rule, but there is no driver loading if your model is new and its ID is not yet added to the "option" module.
Disable the rule running "eject" and the ID will be handled by usb_modeswitch.

Kernel related issues

In some newer kernels, certain devices (some Option, some Huawei, some ZTE as mentioned above) get a special treatment in the usb-storage code to enable switching right away. You would not need USB_ModeSwitch anymore for these specific devices; on the other hand you have no choice of accessing the "CD-ROM" part of your device. Plus, there were cases when the special treatment brought no results and furthermore prevented USB_ModeSwitch to work properly afterwards (happened with ZTE devices, error "-2").

In case of trouble, look into "unusual_devs.h" in the "drivers/usb/storage" folder of your kernel source. If your default ID (vendor and product ID of the storage part) can be found there and you get errors when running USB_ModeSwitch, try first to blacklist "usb-storage".
If that helps, you should consider rebuilding your kernel with the entry in "unusual_devs.h" deactivated. The only thing that will happen is that usb-storage works in the default way afterwards.
I found a **tip** in the Russian Gentoo wiki to do exactly what I just suggested for the **ZTE MF626**.
By the way, at one point there was an agreement among USB developers to keep all future mode switching code out of the kernel drivers if the neccessary

steps can be taken in "user space".

Annother way of influencing the kernel behaviour is the parameter "delay_use" of "usb-storage" which sets the time in seconds after plugging when the storage device will actually be used (and probably automounted). The default value is 5; this might affect the switching result under certain conditions.
To change the default add in /etc/modprobe.conf:

```
options usb-storage delay_use=1 (or 10, or other)
```

Old systems (e.g. CentOS 5 or Xandros 6)

If you are running a not-quite-fresh system (with a kernel below 2.6.27), you may run into incompatibilities between versions of "udev", the device manager. If you get no indication of the usb_modeswitch components doing anything at all (no log files), follow these steps:

- Check if there are other files alongside "40-usb_modeswitch.rules" in the folder "/lib/udev/rules.d".
  If this is **not** the case, move the file to the folder "/etc/udev/rules.d".
- If there is still no action, see if other files in "/etc/udev/rules.d" contain the string "ATTRS".
  If this is **not** the case, edit the file "40-usb_modeswitch.rules" and replace all occurences of "ATTR" and "ATTRS" with "SYSFS". Save the changes and see if something happens when you plug in again.
- In particularly tough cases of "non-action", it might be necessary to analyze udev's actions; this is done by editing "/etc/udev/udev.conf" and change the logging level to "debug".

Contribute

USB_ModeSwitch comes quite handy for experimenting with your own hardware if not supported yet.
The first step would be to try widely used switching methods from known devices, like the "eject" sequence found in the BandLuxe configuration "1a8d:1000". If you have a hint that your device may be made by Huawei originally, try the sequence from "12d1:1446".
Don't worry, it's almost impossible to break anything by experimenting with possibly wrong sequences. Just make sure you re-plug your device after each attempt!

If all this has no effect you can try this approach:
Note the device's Vendor and Product ID from /proc/bus/usb/devices (or from

the output of "lsusb"); the assigned driver is usually "usb-storage". Then spy out the USB communication to the device with the same ID inside M$ Windoze, with the on-board driver installed. The device must be switched there too, and you want to log that moment.
I recommend this tool: "SniffUSB" (**http://www.pcausa.com/Utilities/UsbSnoop/default.htm**).

This is the short version. There is a very good case example from Mark A. Ziesemer here:
Alltel UM175AL USB EVDO under Ubuntu Hardy Heron

Please post any improvements, new device information and/or bug reports to the **ModeSwitchForum** !
If you don't need support you can also send me an old-fashioned - and at your demand confidential - e-mail (see below).

Whodunit

- Josua Dietze (usb_admin at this domain)

Please use only the forum for support questions! For other messages like tested new device configurations or personal/confidential stuff you can mail me directly.

Other contributors

Command line parsing, decent usage/config output and handling, bugfixes added by:

- Joakim Wennergren

TargetClass parameter implementation to support new Option devices/firmware:

- Paul Hardwick (**http://www.pharscape.org**)

Created with initial help from:

- "usbsnoop2libusb.pl" by Timo Lindfors (**http://iki.fi/lindi/usb/usbsnoop2libusb.pl**)

Config file parsing stuff borrowed from:

- Guillaume Dargaud (**http://www.gdargaud.net/Hack/SourceCode.html**)

Hexstr2bin function borrowed from:

- Jouni Malinen (**http://hostap.epitest.fi/wpa_supplicant**, from "common.c")

A special "Thank You" goes to

- **Lars Melin**, **DD-WRT** maintainer,

who tirelessly traces and collects new device configurations from around the electronic universe, then lists them for me to make updates and corrections so much easier !!

Code, ideas and other input from:

- Aki Makkonen
- Denis Sutter
- Lucas Benedičič
- Roman Laube
- Luigi Iotti
- Vincent Teoh
- Tommy Cheng
- Daniel Cooper
- Andrew Bird
- Yaroslav Levandovskiy
- Sakis Dimopoulos
- Steven Fernandez
- Christophe Fergeau
- Nils Radtke
- Andrei Nazarenko
- Filip Aben
- Amit Mendapara
- Roman S. Samarev
- Chi-Hang Long
- Andrey Tikhomirov

- Nicholas Carrier
- Adam Goode
- Daniel Mende
- Leonid Lisovskiy
- Vladislav Grishenko

More contributors (device specific) are listed in the **device_reference.txt**. Thanks to everyone at the **forum** too! Sometimes it takes a considerable reservoir of patience on the road to success ...

History

See the **ChangeLog** of the program package or the **ChangeLog** of the data package which includes the names of supported device models.

Legal

Page last revised: 2015-07-16