White Rabbit Switch: User's Manual

Information about configuring the White Rabbit switch, for final users
October 2014 (wr-switch-sw-v4.1.1)

Alessandro Rubini, Benoit Rat, Federico Vaga et al.

Table of Contents

Introduction

The White Rabbit switch (or WRS) is a major component of the White Rabbit (WR) network. Like any modern managed switch, the WRS includes a CPU with its own operating system.

This manual is for people installing WRS devices, who need to configure them in their network.

1 WRS Documentation

Up to and including release 4.0 of WRS software this manual didn't exist, and the "WRS Build Manual" included some information about configuration.

1.1 The Official Manuals

This is the current set of manuals that accompany the WRS:

- *White Rabbit Switch: Startup Guide*: hardware installation instructions. This manual is provided by the manufacturer: it describes handling measures, the external connectors, hardware features and the initial bring-up of the device.
- *White Rabbit Switch: User's Manual*: documentation about configuring the WRS, at software level. This guide is maintained by software developers. The manual describes configuration in a deployed network, either as a standalone device or as network-booted equipment. The guide also describes how to upgrade the switch, because we'll release new official firmware images over time, as new features are implemented.
- *White Rabbit Switch: Developer's Manual*: it describes the build procedure and how internals work; use of scripts and WRS-specific executables and so on. The manual is by developers and for developers. This is the document to check if you need to customize your *wrs* rebuild software from new repository commits that are not an official release point, or just install your *wrs* with custom configuration values.

The official PDF copy of the three manuals at each release is published in The *files* tab of the software project in `ohwr.org`: (`http://www.ohwr.org/projects/wr-switch-sw/files`). This doesn't apply to release 4.0 and earlier.

The source form of all three manuals is maintained in `wr-switch-sw/doc`. Within the repository, both the *User's Manual* and the *Developer's Manual* are always tracking the software commits, while the *Startup Guide* may not be authoritative because it is bound to device shipping rather than software development.

1.2 Supported Hardware Versions

This document applies to versions 3.3 and 3.4 of the WRS device.

Very few specimens of `wrs` 3.0 though 3.2 were manufactured; if you are the owner of one of them, please refer to version 3.3 of the *wrs-build* document, that includes appendixes about using older versions. As usual, it is in the *files* tab of `ohwr.org`.

V1 and V2 were development items, never shipped.

2 Upgrading WRS Software

The WRS is shipped with a current version of its software image, which is sometiems called *firmware*. If your devices are running a previous version of the software you may want to upgrade, or you may want to replace the firmware images after rebuilding your own, as explained in the *Developer's Manual*.

If you run version 4.1 or later, you can ignore this chapter, that explains a transition between the initial way we passed MAC addresses and the safer approach we introduced in v4.1

2.1 hwinfo

Version 4.1 (October 2014) and later ones use a new way to pass hardware information to all levels of software, such information includes the MAC addresses for the management Ethernet and the SFP ports. Information is now stored in a Flash partition called *hwinfo*, using the SDB file format. SDB is defined in the `fpga-configuration-space` within `ohwr.org`. Before using SDB we used to edit the boot loader's configuration at flash time, a bad practice developed in a hurry.

The *hwinfo* structure is now written to *dataflash* by the manufacturer, and never changed even when perforiming a complete re-flash of the device, because the flashing scripts preserve the *hwinfo* memory area.

When upgrading from a pre-4.1 switch software, you need to create this *hwinfo* data structure. This procedure is mostly automatic, but you need to be aware of the steps involved, in case something goes wrong.

2.2 Upgrading from v4.0

Version 4.0 and later of `wr-switch-sw` are able to upgrade themselves if you place the profer files in the */update* directory of the WRS. However, in version 4.0 we forgot to provide for an upgrade of the boot loader and didn't note that if the fron USB cable is not plugged, the upgrade procedure blocks.

This latter problem happens because messages are written to the management USB port, to help people flashing from scratch, and the write is a blocking one by default: if nobody collects the USB data, the system waits for a recipient. With version 4.1.1 we fixed the problem, using non-blocking operations; we'd better loose messages than block installation because nobody is reading.

Thus, there are two different ways to upgrade; which one you prefer we can't tell. Both work, each with its own drawbacks. Each of them preserves the current MAC addresses.

2.2.1 Upgrading from v4.0 with the USB cable

This is the procedure if you are able to walk to your WRS and connect to the management USB port, even if the port is not actually used:

- Copy your own `wrs-firmware.tar` for v4.1 into the */update* partition. This can be the official firwmare or one you built yourself. Then reboot and wait for everything to settle (the system will reboot once more by itself).
- Copy `wrs-firmware.tar` again. And reboot again. The system will reboot once more by itself.
- Now you have a running v4.1 with your *hwinfo* in place and the old MAC addresses preserved.

We save you from the long description of what is happening in the various steps. If needed, it is in the *git* history of `wr-switch-sw`, at release point `v4.1`.

2.2.2 Upgrading from v4.0 remotely

If you can't walk to the switch, the procedure is faster but more commands need to be typed on the root shell of the switch. We support a single upgrade provided you change the kernel and initial filesystem before rebooting.

- Copy your own `wrs-firmware.tar` for v4.1 into the */update* partition. This can be the official firwmare or one you built yourself.
- Create and mount */boot* within the switch. This means running the following commands in *ssh*:

  ```
  mkdir /boot
  ```

  ```
  mount -t ubifs ubi0:boot /boot
  ```
- Copy `wrs-initramfs.gz` (which is to be found inside `wrs-firmware.tar`) to the */boot* partition just mounted. This ensures the new upgrade procedure will run, the one that doesn't block if the USB cable is unplugged.
- Copy `zImage` (again, to be found inside `wrs-firmware.tar`) to the */boot* partition. This is need to be able to access the *hwinfo* partition at next boot.
- Reboot and wait for everything to settle (the system will reboot once more by itself after upgrading everything). The MAC addresses will be saved to *hwinfo* during the update procedure, thanks to the new kernel and new boot procedure you manually copied to */boot*.

Note: if you forget to place the new kernel or `wrs-initramfs.gz` in */boot*, no big damage will happen, but you'll have lost your MAC address for the WR ports. You'll find a randomly-chosen value, that will however be persistent over reboot (because it is saved to *hwinfo* after you boot with the new kernel.

2.3 Upgrading from v3.x

Upgrading from versions older than 4.0 (August 2014) requires physical access to the device and, unfortunately, requires some extra steps especially if you want to preserve your MAC addresses.

One possible path is flashing version 4.0 (please refer to v4.0 manuals) and then proceed as described in Section 2.2 [Upgrading from v4.0], page 2. When flashing version 4.0 you'll need to pass your MAC addresses on the command line of the flasher, so please take not of what they are.

Another option is flashing the latest firwmare version and then build your own *hwinfo* structure by specifying your MAC addresses. `wr-switch-sw` includes specific tools for both steps. They are described in the *Developer's Manual*, because they are expected to only be performed by the manufacturer, not the final user.

3 Kconfig Support

After release 3.3 of this software package, we added Kconfig support to wr-switch-sw. The user can ignore this step: building as usual from a fresh checkout of `wr-switch-sw` silently selects the default configuration.

You may exploit this `Kconfig` option to build firmware images preconfigured for your own network.

To change configuration, you are expected to run "`make menuconfig`" (or *gconfig* or *kconfig* or the old text-mode *config*) from the top-level directory of *wr-switch-sw*. To silently enact the default configuration, run "`make defconfig`" (this is done by the normal build if no configuration is present).

The released firmware image uses the default configuration, but we'll soon offer the option to change most of these values at boot-time (by downloading configuration from a network server) or run-time.

The following configuration options are available

`CONFIG_BR2_CONFIGFILE`
: This string option lists a file to be used as Buildroot (BR2) configuration. A simple filename or relative pathname refers to the `configs/buildroot` directory; an absolute pathname is used unchanged.

`CONFIG_NTP_SERVER`
: The NTP server used to prime White Rabbit time, at system boot. The option can be an IP address or a host name, if DNS is properly configured. The configuration value is stored in `/wr/etc/wr_date.conf`. An empty string (default) disables NTP access at boot time.

`CONFIG_DNS_SERVER`
`CONFIG_DNS_DOMAIN`
: The DNS server (as an IP address) and default domain. The values end up in `/etc/resolv.conf` of the generated filesystem. By default the two strings are empty.

`CONFIG_REMOTE_SYSLOG_SERVER`
`CONFIG_REMOTE_SYSLOG_UDP`
: Configuration for system log. The name (or IP address) of the server is stored in `/etc/rsyslog.conf` of the generated filesystem. The UDP option, set by default, chooses UDP transmission; if unset it selects TCP communication.

`CONFIG_SNMP_TRAPSINK_ADDRESS`
`CONFIG_SNMP_TRAP2SINK_ADDRESS`
`CONFIG_SNMP_RO_COMMUNITY`
`CONFIG_SNMP_RW_COMMUNITY`
: Configuration for the SNMP agent. Addresses can be IP addresses or names (if DNS is configured and working), they are unset by default. Community values are strings and they default to `public` and `private`.

`CONFIG_WRS_LOG_HAL`
`CONFIG_WRS_LOG_RTU`
`CONFIG_WRS_LOG_PTP`
: Logging options for the three main WRS processes. Each value can be a pathname, to select logging to file (and `/dev/kmsg` is a possible "file" target) or a *facility.level* string, like `daemon.debug`, for syslog-based logging. An empty strings selects no logging at all. Please note that unknown facility names will generate a runtime error on the switch. All three strings default to "`daemon.info`".

`CONFIG_KEEP_ROOTFS`
: A boolean option for developers: if set the build script does not delete the temporary copy of the generated filesystem and reports its pathname in the build messages.

4 Booting with Barebox

After the initial installation, the boot loader offers an interactive menu, where the first entry is selected by default. The menu is a simple ASCII interface on the serial port, and looks like the following:

```
Welcome on WRSv3 Boot Sequence
    1: boot from nand (default)
    2: boot from TFTP script
    3: edit config
    4: exit to shell
    5: reboot
```

If flashing of the whole system was successful, the first entry will simply work, booting the switch without any network access. Although a DCHP client runs by default after boot, everything will work even if you leave the Ethernet port unconnected or you have no DHCP server when the switch is operational.

If booting from NAND memory fails (for example because you erased the kernel or incurred in other mishaps during development) the menu is re-entered automatically.

The other entries are provided to help developers.

4.1 Description of the menus

The individual menu items perform the following actions:

`1: boot from nand (default)`
: This entry is selected by default after 10 seconds of inactivity on the serial port. It boots the system from its own NAND memory. This "just works".

`2: boot from TFTP script`
: This entry tries to download a *barebox* script from your TFTP server; if successful it then executes it. Developers are expected to customize the script to support any kind of environment, from customized kernel command-line to NFS-Root environments. See Section 4.2 [Using wrboot], page 5 for details.

`3: edit config`
: This fires the editor on the configuration file, and saves it to flash when the user is done. This is useful to change the MAC address of the ARM network port. Please note that saving save the whole '`/env`' file tree, so you can also change the init scripts interactively and have them stored persistently on the flash.

`4: exit to shell`
: By choosing this entry, the user can access the shell-like interface of *barebox*. The entry is aimed at developers who know what they are going to type.

`5: reboot`
: This entry is useful to see and log the exact boot messages. Since the serial-USB converter is *switch-powered* and not *USB-powered*, you won't be able to hook at the serial port soon enough after power-on. Actually, the menu timeout is left to 10 seconds and not less for the very same reason.

4.2 Using wrboot

If you use the *wrboot* script option, you can for example run an NFS-Root system or do whatever customization and testing you want.

The complete filesystem after a successful build is called `images/wrs-image.tar.gz`, and is not included in the release firmware file, because an installed switch runs an *initramfs* system with a separate `/usr` partition in flash memory.

The provided procedure tries to load the script from TFTP under three different names, from most specific to most generic, and the first match will be used. The first name is MAC-address-based, the second is IP-address-based and the third is just '`wrboot`'.

This is for example what I see in my logs when only providing '`wrboot`':

```
dhcpd: DHCPOFFER on 192.168.16.224 to 02:0b:ad:c0:ff:ee via eth0
atftpd[5623]: Serving wrboot-02:0B:AD:C0:FF:EE to 192.168.16.224:1029
atftpd[5623]: Serving 192.168.16.224/wrboot to 192.168.16.224:1030
atftpd[5623]: Serving wrboot to 192.168.16.224:1031
mountd[21014]: NFS mount of /tftpboot/192.168.16.9 attempted from 192.168.16.9
```

We chose to place the IP-address-based name in a subdirectory because this is the default place where the NFS-Root filesystem is mounted from, as shown in the log excerpt above. So you'll have your '`wrboot`' in the same place.

Note: recent *barebox* versions require scripts to include a leading `#!/bin/sh`. Examples in *wr-switch-sw* did not include the line until April 2014 included.

The '`binaries`' subdirectory of `wr-switch-sw` includes two known-working wrboot scripts as examples; one if for use with static IP addresses and the other relies on DHCP. If you want to override the default NFS-Root directory mounted from the server (which is `/tftpboot/<ip-address>`) you can add something like the following line to your '`wrboot`' script:

```
bootargs="$bootargs nfsroot=/opt/root/wrs-3"
```

4.3 Creating an NFS-Root Environment for WRS

In order to create an NFS root directory, you should uncompress `wrs-image.tar.gz` that is created at build time. If you use a released `wrs-firmware.tar`, however, you'll have no overall filesystem for the switch, and you should rebuild it from two parts. This is how to create your NFS filesystem (please adapt for your local pathnames):

```
FW=/tftpboot/wrs-firmware.tar
DIR=/opt/root/wrs-3

mkdir -p $DIR
tar xOf $FW wrs-initramfs.gz | zcat | \
    (cd $DIR && sudo cpio --make-directories --extract)
tar xOf $FW wrs-usr.tar.gz | sudo tar xzf - -C $DIR/usr
```

The above commands extract to *stdout* the two parts of the WRS filesystem, to then uncompress them to the proper directories. The first *tar* pipe is less friendly because the *initramfs* is a compressed *cpio* archive, and *cpio* as a command lacks automatic decompression and the `-C` (change directory) option.

5 SNMP Support

The White Rabbit Switch supports SNMP, although some more work is needed in this respect. The default read-only "community" name is `private`, but you can change it from the `Kconfig` interface before building. The default read-write community is `private`.

The switch supports all the standard information through the *net-snmp* installation. We'll remove some of the items in a later release, because nobody wants to check running processes or disk space usage.

The additional, switch-specific information are in the "enterprise.96.100 subtree, where `96` is CERN and `100` is White Rabbit. The associated MIB is in the directory `userspace/snmpd`, where related source files live as well.

5.1 The WRS MIB

This a summary of the available tables and scalars:

`96.100.1`

This is a simple scalar as a test. It is an integer value that is incremented each time you access it. It can be used to test basic functionality.

`96.100.2`

Port statistics, as an SNMP table. The first column is the name of each counter, and further columns represent interfaces `wr0` through `wr17`. Each counters is shown in a table line, as the number and names of the counters may change in the future.

`96.100.3`

White Rabbit specific information. Subid `.1` is the global items, and subid `.2` is a table with per-port items.

`96.100.4`

Hardware, gateware and software versions, plus serial number and other hardware information.

`96.100.5`

The internal White Rabbit time, as a number and a string.

Note: due to a buglet of mine, there is an extra item at the end of each table (`96.100.2` and `96.100.3.2`. It makes no harm, so its removal is not high priority.

the easiest way to retrieve the values is using *snmpwalk*, telling it to access our MIB file in order to use symbolic names. Assuming `wrs` is the DNS name for your White Rabbit Switch and `WR_SWITCH_SW` is an environment variable pointing to this package:

```
snmpwalk -c public -v 2c wrs \
    -m +${WR_SWITCH_SW}/userspace/snmpd/WR-SWITCH-MIB.txt \
    1.3.6.1.4.1.96.100
```

Using SNMP version 1 instead of 2c is fine as well, but you won't receive the 64-bit values for slave/tracking information.

The output you will get back is something like the following. Clearly the software commit in this example is my own development version while writing this section:

```
WR-SWITCH-MIB::wrsScalar.0 = INTEGER: 2
WR-SWITCH-MIB::pstatsDescr.1 = STRING: TX Underrun
WR-SWITCH-MIB::pstatsDescr.2 = STRING: RX Overrun
WR-SWITCH-MIB::pstatsDescr.3 = STRING: RX Invalid Code
[...]
WR-SWITCH-MIB::pstatsDescr.38 = STRING: Forwarded
WR-SWITCH-MIB::pstatsDescr.39 = STRING: TRU Resp Valid
WR-SWITCH-MIB::pstatsWR0.1 = Counter32: 0
[...]
WR-SWITCH-MIB::pstatsWR17.38 = Counter32: 50819
WR-SWITCH-MIB::pstatsWR17.39 = Counter32: 0
WR-SWITCH-MIB::pstatsEntry.20 = Counter32: 0
WR-SWITCH-MIB::ppsiGrandmaterID.0 = Hex-STRING: 00 00 00 00 00 00 00 00
WR-SWITCH-MIB::ppsiOwnID.0 = Hex-STRING: 00 00 00 00 00 00 00 00
WR-SWITCH-MIB::ppsiMode.0 = INTEGER: unknown(0)
WR-SWITCH-MIB::ppsiServoState.0 = STRING:
WR-SWITCH-MIB::ppsiPhaseTracking.0 = INTEGER: not-tracking(0)
[...]
WR-SWITCH-MIB::portLink.14 = INTEGER: down(0)
WR-SWITCH-MIB::portLink.15 = INTEGER: up(1)
WR-SWITCH-MIB::portLink.16 = INTEGER: down(0)
[...]
WR-SWITCH-MIB::portPeer.18 = Hex-STRING: FF FF FF FF FF FF FF FF
```

```
WR-SWITCH-MIB::ppsiPort.5 = Hex-STRING: FF FF FF FF FF FF FF FF
WR-SWITCH-MIB::wrsVersionSw.0 = STRING: v4.0-rc1-42-gcec7805+
WR-SWITCH-MIB::wrsVersionGw1.0 = STRING: 7cce708
WR-SWITCH-MIB::wrsVersionGw2.0 = STRING: 5118070
WR-SWITCH-MIB::wrsVersionGw3.0 = STRING: 7efeb16
WR-SWITCH-MIB::wrsVersionHw1.0 = STRING: 3.30
WR-SWITCH-MIB::wrsVersionHw2.0 = STRING: LX240T
WR-SWITCH-MIB::wrsManufacturer.0 = STRING: Seven Solutions
WR-SWITCH-MIB::wrsSerialNumber.0 = STRING: 12345
WR-SWITCH-MIB::wrsScbVersion.0 = STRING: 3.3
WR-SWITCH-MIB::wrsDateTAI.0 = Counter64: 1406623390
WR-SWITCH-MIB::wrsDateString.0 = STRING: 2014-07-29-08:43:10
```

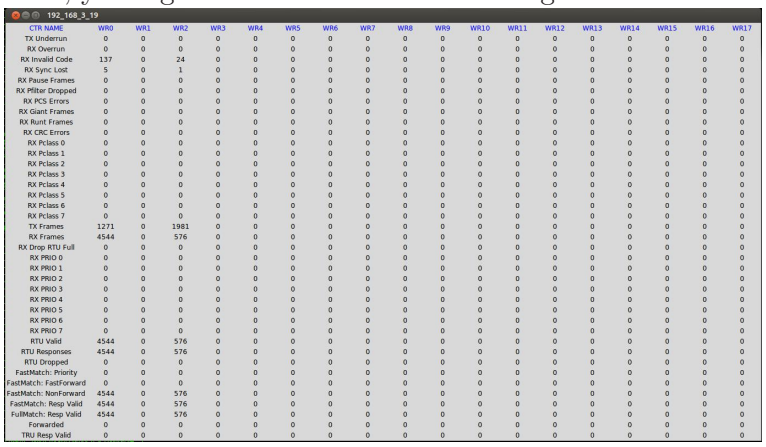
5.2 show-pstats

To visualize all port statistics in a single window, this package includes the simple tool `userspace/snmpd/show-pstats`. It is a Tk script, so you need to install `tk8.5` or any other version.

The script receives one or more host names (or IP addresses) on the command line. They must refer to a switch (or switches) or the program fails like this:

```
laptopo% ./show-pstats morgana
Error in snmpwalk for host morgana
No log handling enabled - using stderr logging
.1.3.6.1.4.1.96.100.2.1.: Unknown Object Identifier (Sub-id not found: enterprises -> )
```

If everything goes well, you'll get a window like the following one:



Appendix A Bugs and Troubleshooting

Even if the package is already released and used in production, some details can be suboptimal, while some procedures may be tricky and need more explanation.

We are collecting all those issues in the *wiki* page of the project, to avoid frequent updates to this manual to just collect those details. So please visit www.ohwr.org/projects/wr-switch-sw/wiki/Bugs and www.ohwr.org/projects/wr-switch-sw/wiki/Troubleshooting if you have any problem with this package, but feel free to reach us on the mailing list if you don't find help there.