

White Rabbit Switch: User's Manual

Information about configuring the White Rabbit switch, for final users
July 2015 (wr-switch-sw-v4.2)

Alessandro Rubini, Adam Wujek, Benoit Rat, Federico Vaga, ...

Table of Contents

Introduction	1
1 WRS Documentation.....	1
1.1 The Official Manuals	1
1.2 Supported Hardware Versions.....	1
2 Upgrading WRS Software	2
2.1 hwinfo	2
2.2 Upgrading from v4.0.....	2
2.2.1 Upgrading from v4.0 with the USB cable.....	2
2.2.2 Upgrading from v4.0 remotely	3
2.3 Upgrading from v3.x	3
3 Configuration of the Device	3
3.1 Dynamic WRS Configuration	4
3.2 The Configuration File	4
3.3 Configuration Items that Apply at Build Time.....	5
3.4 Configuration Items that Apply at Run Time	5
3.5 Timing Configuration.....	9
3.6 VLANs Configuration	10
3.7 Front panel's LEDs.....	11
3.7.1 Status LEDs	11
3.7.2 Ports' LEDs.....	11
4 Booting with Barebox	11
4.1 Description of the menus	11
4.2 Using wrboot.....	12
4.3 Creating an NFS-Root Environment for WRS.....	13
5 WRS Command-Line Tools	13
5.1 sdb-read	14
5.2 wrs_vlans	15
5.3 wrs_auxclk	17
6 SNMP Support.....	17
6.1 The WRS MIB.....	18
6.2 show-pstats	19
Appendix A Bugs and Troubleshooting.....	20

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 programs 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.
- *White Rabbit Switch: Failures and Diagnostics*: describe various failure scenarios of a switch and ways how to recognize them. Additionally describe SNMP exports of a switch (WR-SWITCH-MIB).

The official PDF copy of these manuals at each release is published in the *files* tab of the software project in [ohwr.org](http://www.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 four manuals is maintained in `wr-switch-sw/doc`. Within the repository, three of them the *User's Manual*, the *Developer's Manual* and the *Failures and Diagnostics* 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](http://www.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 sometimes 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 performing 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 proper 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 front 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 lose 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 at least v4.1 into the `/update` partition. This can be the official firmware 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 updated version 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 at least v4.1 into the `/update` partition. This can be the official firmware 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 note of what they are.

Another option is flashing the latest firmware 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 Configuration of the Device

After release 3.3 of this software package, we added Kconfig support to `wr-switch-sw`. If you build your software image (as documented in the WRS *Developer's Manual*), you can make some configuration choices for your customized firmware image. But most users are not expected to rebuild.

After release 4.1, we moved most of the configuration to run-time (rather than build-time): the `.config` file that you create with a `"make menuconfig"` or equivalent command, is now copied to the WRS filesystem and used during boot. Moreover, the switch can download a new configuration at boot time, if so configured. This allows customization of each installed switch through a central server, without modifying the filesystem image in each specimen.

Starting with release 4.2, we added `"make config"` support at run time, to be run in `/wr/etc`; the file is called `dot-config`, and not `.config`. This is meant to be useful for developers,

when testing different configurations in the lab, rather than in production. We also support “`make config`”, “`make defconfig`” and `make oldconfig`”. We are not yet able to run “`make menuconfig`”, but that may be added when we upgrade *buildroot* after release 4.2.

3.1 Dynamic WRS Configuration

The switch can boot using its internal NAND memory or as an NFS-Root host. In the latter case configuration can be changed on the server, and if a unit is replaced, a change in the DHCP database is all that’s needed to recover network operation. But this option implies some network traffic on your management network, as well as an NFS server able to host all of your switches. When a switch is booted from internal storage, we used to rely on internal configuration (either selected at build time or modified using *ssh* or the web interface). This approach doesn’t scale well to large installation, because if a device needs to be replaced, its own configuration is lost. With *dynamic configuration*, each WRS device loads its own configuration file each time it is booted, and applies the choices before starting any service. The name of the configuration file can include the MAC or IP address of the device, to allow running several switches with different configurations in the same network. The URL to the configuration file can also be retrieved from DHCP server.

3.2 The Configuration File

The main configuration file for the WRS is `/wr/etc/dot-config`. You create this file by running “`make menuconfig`” within `wr-switch-sw`, and making your choices. You can also edit the text file, or run other configurators: `make xconfig`, `make gconfig`, `make config`.

The configuration step creates `.config`, that you can copy to your WRS as `/wr/etc/dot-config`. After reboot, you’ll see your choices in effect.

The first configuration choice is about source of the `dot-config` file (items starting with `CONFIG_DOTCONF_SOURCE_`). The following `dot-config` sources are implemented in current version:

`CONFIG_DOTCONF_SOURCE_LOCAL`

Use local `dot-config` file stored in `/wr/etc/dot-config`. In this case no network access is performed.

`CONFIG_DOTCONF_SOURCE_REMOTE`

Get a `dot-config` file from the URL provided in `CONFIG_DOTCONF_URL`.

`CONFIG_DOTCONF_SOURCE_FORCE_DHCP`

Get a URL to a `dot-config` file from a DHCP server. The URL can be configured in the “`filename`” configuration field of the DHCP server. The configured URL has to be in the same form as `CONFIG_DOTCONF_URL`.

`CONFIG_DOTCONF_SOURCE_TRY_DHCP`

The same as `CONFIG_DOTCONF_SOURCE_FORCE_DHCP`, but this option does not cause errors in SNMP’s objects if the switch fails to retrieve the URL to the `dot-config` via DHCP. Note that syntax and download errors of `dot-config` are notified in the same way as for other choices.

If the selected option triggers WRS to download a new `dot-config` file, it will replace the copy in the local storage if it is different.

The URL (stored in `CONFIG_DOTCONF_URL` or retrieved via DHCP) is of the form “*protocol://host/pathname*”. The special upper-case strings `IPADDR` and `MACADDR` are substituted with the current addresses of the management port of the switch.

The three parts of the URL are as follows:

protocol

We support `http`, `ftp` and `tftp`. Any other protocols result in an error, and the `dot-config` file is not replaced.

host

The host can be an IP address, or a name. In order to use a name you must specify a valid `CONFIG_DNS_SERVER` and optionally `CONFIG_DNS_DOMAIN`. The values in the current `dot-config` are used to load the new file.

path

The pathname can include directory components and `IPADDR` or `MACADDR` (or both).

For example this is a valid configuration for run-time update:

```
CONFIG_DOTCONF_SOURCE_REMOTE=y
CONFIG_DOTCONF_URL="tftp://morgana/wrs-config-IPADDR"
CONFIG_DNS_SERVER="192.168.16.1"
CONFIG_DNS_DOMAIN="i.gnudd.com"
```

And it results, in my case, in `wrs-config-192.168.16.9` being served to the WRS.

Please remember that the new `dot-config` should include a valid `CONFIG_DOTCONF_SOURCE_*` setting, or you won't be able to update the configuration at the next boot. In any case, you can always copy a configuration file using `ssh`, or use the web interface to change the configuration. Changes performed using the web interface are immediately active, because the web server takes proper action; the new file copied over with `ssh`, or any hand-edits, are only effective at next boot, unless overwritten by a remote configuration file. In case there are errors or unknown configuration entries in the retrieved file, the old one will be used.

3.3 Configuration Items that Apply at Build Time

The following items in `dot-config` are used at build time; changing them in the installed version has no effect:

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_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.

3.4 Configuration Items that Apply at Run Time

The following items in `dot-config` are used at run time: at every boot the value (the old one or the just-downloaded one) is used in the appropriate way, before the respective service is started. When the value is changed by the web interface, proper action is taken.

CONFIG_DOTCONF_SOURCE_LOCAL**CONFIG_DOTCONF_SOURCE_REMOTE****CONFIG_DOTCONF_SOURCE_FORCE_DHCP****CONFIG_DOTCONF_SOURCE_TRY_DHCP****CONFIG_DOTCONF_URL**

The source and location of a config file to be used at a replacement the next time the system boots. See [Section 3.1 \[Dynamic WRS Configuration\]](#), page 4 and [Section 3.2 \[The Configuration File\]](#), page 4 for details.

CONFIG_ETH0_DHCP

CONFIG_ETH0_DHCP_ONCE

CONFIG_ETH0_STATIC

Configuration of management port's (`eth0`) IP. When `CONFIG_ETH0_DHCP` is used, then switch tries to obtain IP via DHCP forever. For option `CONFIG_ETH0_DHCP_ONCE` switch tries to get IP via DHCP once, if this try is unsuccessful then switch uses static IP. `CONFIG_ETH0_STATIC` forces switch to use provided static IP address.

CONFIG_ETH0_IP

CONFIG_ETH0_MASK

CONFIG_ETH0_NETWORK

CONFIG_ETH0_BROADCAST

CONFIG_ETH0_GATEWAY

Management port's (`eth0`) static IP configuration when `CONFIG_ETH0_DHCP_ONCE` or `CONFIG_ETH0_STATIC` parameter is used.

CONFIG_ROOT_PWD_IS_ENCRYPTED

CONFIG_ROOT_PWD_CLEAR

CONFIG_ROOT_PWD_CYPHER

This set of options allow setting the password for the “root” user (the administrator). The password is used to login to your switch using `ssh` (secure shell). If you choose `CONFIG_ROOT_PWD_IS_ENCRYPTED`, you will be prompted for a text version of a pre-encrypted password (`CONFIG_ROOT_PWD_CYPHER`). To encrypt your *magic* string, you must run “`mkpasswd --method=md5 magic`” on your Linux host (or switch). If you choose to configure an unencrypted password, then you are asked to specify it as `CONFIG_ROOT_PWD_CLEAR`. In this latter case encryption is performed at run-time to use the normal *ssh* authentication, but the clear-text password will remain in dot-config. By default the root password is an empty string, like in the initial *wr-switch-sw* releases.

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_WRS_LOG_HAL

CONFIG_WRS_LOG_RTU

CONFIG_WRS_LOG_PTP

CONFIG_WRS_LOG_WRSWATCHDOG

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 string selects no logging at all. Please note that unknown facility names will generate a runtime error on the switch. All four strings default to `daemon.info`.

CONFIG_WRS_LOG_SNMPD

Value can be a pathname, to select logging to file (and `/dev/kmsg` is a possible “file” target) or a valid snmpd log option (without `-L`). Allowed strings are in format “*s facility*” (e.g. “`s daemon`”) and “*s level facility*” (e.g. “`s 2 daemon`”). For example, “`sd`” or “`s daemon`” will forward messages to syslog with daemon as facility. To set level (i.e. 5) use “`S 5 daemon`”. For details please check “`man snmpcmd`”. An empty string selects no logging at all. Please note that unknown facility names will generate a runtime error on the switch.

CONFIG_WRS_LOG_MONIT

The string can be a pathname (e.g. `/dev/kmsg`) or a `syslog` string. An empty string is used to represent no logging. If it is needed to select facility and level please leave here empty string and change `/etc/monitrc` file directly. Please note that unknown facility names will generate a runtime error on the switch.

CONFIG_PORT00_PARAMS

CONFIG_PORT01_PARAMS

...

CONFIG_PORT17_PARAMS

These configuration items are used to set up port parameters; this includes the delays, the PTP role, PTP protocol type and the fiber type as an integer index. Most likely the default values work for you. See [Section 3.5 \[Timing Configuration\]](#), [page 9](#) for details.

CONFIG_SFP00_PARAMS

...

CONFIG_SFP09_PARAMS

Configuration for SFP models. You should fill values for all SFP models you are using in your WRS and all wavelengths you are using. See [Section 3.5 \[Timing Configuration\]](#), [page 9](#) for details.

CONFIG_FIBER00_PARAMS

...

CONFIG_FIBER03_PARAMS

Configuration for fiber types. You are expected to have no more than 4 fiber types installed in your deployment (if more, you need to add items to the `dot-config` file). See [Section 3.5 \[Timing Configuration\]](#), [page 9](#) for details.

CONFIG_TIME_GM

CONFIG_TIME_FM

CONFIG_TIME_BC

The type of PTP clock this switch is. Only one of the three items should be set (running “`make menuconfig`” offers them as an exclusive choice). The options select a grand-master with external reference, from GPS or Cesium or both; a free-running master, used for isolated acquisition networks, without an external reference; or a normal “boundary-clock” device that is slave on some ports and master on other ports.

CONFIG_PTP_PORT_PARAMS**CONFIG_PTP_CUSTOM****CONFIG_PTP_REMOTE_CONF**

By default, PPSi configuration file is assembled based on role and protocol parameters stored in `PORTxx_PARAMS`. `TIME_BC` selected by *Kconfig* defaults role of port `wr0` to slave, other ports to master. For `TIME_GM` and `TIME_FM` all ports are mandated to master. Parameters `clock-class` and `clock-accuracy` can be changed or new global PPSi settings can be added by editing file `/wr/etc/ppsi-pre.conf`, which is used as begging of final ppsi configuration file.

Alternatively PPSi can use custom user file for configuration (`CONFIG_PTP_CUSTOM`).

Finally, you can choose `PTP_REMOTE_CONF` and be able to specify an URL (`http://`, `ftp://` or `tftp://`) whence the switch will download the `ppsi.conf` at boot time. The filename in the URL can include `IPADDR` and/or `MACADDR`, so the same configuration string can be used to set up a batch of switches with different configurations.

Please see the help provided within *Kconfig* for more details about the various options we support.

CONFIG_PTP_CUSTOM_FILENAME

If you chose `CONFIG_PTP_CUSTOM` in the choice above, you can provide your own filename for the PPSi configuration file; the chosen name is expected to be installed in the WRS filesystem.

CONFIG_PTP_CONF_URL

If you choose `CONFIG_PTP_REMOTE_CONF` this is the URL used to download `ppsi.conf` at each system boot.

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_SNMP_TEMP_THOLD_FPGA**CONFIG_SNMP_TEMP_THOLD_PLL****CONFIG_SNMP_TEMP_THOLD_PSL****CONFIG_SNMP_TEMP_THOLD_PSR**

Threshold levels for FPGA, PLL, Power Supply Left (PSL) and Power Supply Right (PSR) temperature sensors. When any temperature exceeds threshold level SNMP object `WR-SWITCH-MIB::tempWarning` will change accordingly.

CONFIG_SNMP_SWCORESTATUS_HP_FRAME_RATE

Error via SNMP if rate of HP frames on any port exceed given value.

CONFIG_SNMP_SWCORESTATUS_RX_FRAME_RATE

Error via SNMP if rate of RX frames on any port exceed given value.

CONFIG_SNMP_SWCORESTATUS_RX_PRIO_FRAME_RATE

Error if frame rate of any RX priority exceed given value.

```
CONFIG_WRSAUXCLK_FREQ
CONFIG_WRSAUXCLK_DUTY
CONFIG_WRSAUXCLK_CSHIFT
CONFIG_WRSAUXCLK_SIGDEL
CONFIG_WRSAUXCLK_PPSHIFT
```

Set of parameters passed to `wrs_auxclk` at boot time. For more information please check [Section 5.3 \[wrs-auxclk\]](#), page 17.

```
CONFIG_MONIT_DISABLE
```

Disable monitoring of running processes by `monit`. `Monit` by default re-spawns processes that have died. This option is useful mostly during development.

3.5 Timing Configuration

This section describes how timing configuration works in the switch. Please note that up to version 4.1 (included) of `wr-switch-sw` things were different and not really documented.

Timing configuration now depends on three sets of *dot-config* variables: per-port information, per-sfp information and fiber description.

This is, for explanation's sake, an example of such items:

```
CONFIG_PORT09_PARAMS="name=wr9,proto=raw,tx=226214,rx=226758,role=slave,fiber=2"
CONFIG_SFP00_PARAMS="pn=AXGE-1254-0531,tx=0,rx=0,wl_txx=1310+1490"
CONFIG_FIBER02_PARAMS="alpha_1310_1490=2.6787e-04"
```

When making timing calculation for port `wr9`, assuming the auto-detected SFP model is “AXGE-1254-0531”, the software uses configuration in the following way:

- The port has known tx and rx delays (around 226ns); the values depend on trace length and other hardware-specific details and are determined by a calibration procedure. These values are used as constant delays in the *tx* and *rx* directions.
- The port is also configured as slave (*role*) using raw whiterabbit protocol (*proto*) and is deployed using fiber type 2 – this number is just a local enumeration of fiber types; most likely you'll be using type “0” in every port.
- The transceiver installed uses 1310nm light for tx and 1490nm light for rx (this is part of the specification of the transceiver, but cannot be auto-detected). Moreover it has 0 constant delay in both tx and rx, determined by calibration.
- The fiber type being used (type 2 here), when driven with 1310nm and 1490nm wavelengths, features an *alpha* parameter of 0.00026787 (i.e. a ratio of 1.00026787) between the speed of light in the two directions. This value depends on both the fiber type and wavelength, and is determined, again, by calibration.

Please note that only one alpha value is provided, because the opposite one (`alpha_1490_1310`) is calculated by software.

SFP name matching

SFP matching is based on vendor name (*vn*), part number (*pn*) and vendor serial (*vs*). While matching SFP's values are compared with values stored in `CONFIG_SFPxx_PARAMS`. First try is to match all SFP identifiers (*vn*, *pn* and *vs*) with stored in config. If match is not successful, *vn* and *pn* of SFP are compared only with config entries without vendor serial. If match is still not found SFP's values are compared with config entries, which has defined only part number. Such approach prevents matching SFPs to config entries with defined serial.

Below are shown matching examples:

```
CONFIG_SFP00_PARAMS="vn=Axcen Photonics,pn=AXGE-3454-0531,vs=AX12390009629,tx=0,rx=0,..."
```

Above config may be matched only to one SFP.

```
CONFIG_SFP01_PARAMS="vn=Axcen Photonics,pn=AXGE-3454-0531,tx=0,rx=0,wl_txrx=1310+1490"
```

Above config may be matched only to SFP with vendor name "Axcen Photonics" and part number "AXGE-3454-0531", with exception to these SFPs that were matched to example like above (with vendor serial defined).

```
CONFIG_SFP02_PARAMS="pn=AXGE-3454-0531,tx=0,rx=0,wl_txrx=1310+1490"
```

Config above will be matched to all SFPs with part number "AXGE-3454-0531", that were not matched by configs above.

Other Deployments

The example above matches the choices we make at CERN, where our White Rabbit networks are all run with a single mono-modal fiber and 1310/1490 light.

If you are using dual-fiber transceivers, which is acceptable for short links, you use the same wavelength in both direction, over two fibers of the same length. In this case you may choose to avoid writing the `wl_txrx` parameter in SFP configuration and the `alpha_XX_XX` parameter in fiber configuration. The missing parameters will cause warning messages to log destination, but are not fatal, and a default alpha of 0 is used.

If you are using a pair of transceivers with different wavelengths, and long fibers, you should provide an appropriate value of alpha, according to laboratory measures on your fiber type. The `CONFIG_FIBERxx_PARAMS` items are parsed as a list of comma-separated assignments, so you can specify and number of wavelength pairs. The accuracy of your value depends on the length of the fiber link. For a 10km fiber (100us round-trip) you need to know alpha up to 1e-7 if you want the related uncertainty to be less than 10ps.

Calibration

Calibration of per-port and per-SFP delays is described in the White Rabbit wiki: <http://www.ohwr.org/projects/white-rabbit/wiki/Calibration>. The procedure can measure the total constant delays, but splitting between the what depends on the port and what depends on the transceiver is arbitrary (also, the split between the tx and rx paths is arbitrary).

The delays used in this example come from values listed in the above web page, and you should not be surprised by the fact that the transceiver specifies the delays as zero. By performing the calibration procedure using this very transceiver type, the hardware engineers decided to assign the whole of the delay to the port (any split of the measured value is equally right). Other transceiver types can be calibrated to either positive or negative values, to cope with the *difference* between them and the AXGE devices.

3.6 VLANs Configuration

Unfortunately, the current firmware release does not support VLAN configuration in the main *dot-config* file. Therefore, if you want to use VLANs you have to manually configure them using the *wrs_vlans* tool described in [Section 5.2 \[wrs_vlans\]](#), [page 15](#).

In addition to that, to have synchronization working with VLANs, you have to prepare a custom *PPSi* configuration file with VLANs specified per-port. You can simply copy the file generated in the WRS filesystem (*/wr/etc/ppsi.conf*) to a central `tftp/http/ftp` server where *dot-config* files for your switches are stored and fetched on boot time. For every VLAN-enabled port you should add the following line:

```
vlan <VID>
```

where *VID* is a VLAN ID configured on the port. To let your switch use the modified *ppsi.conf*, you should add it as `CONFIG_PTP_REMOTE_CONF` option in the *dot-config* ([Section 3.4 \[Configuration Items that Apply at Run Time\]](#), [page 5](#)). This way it will be fetched and applied every time your switch boots.

3.7 Front panel's LEDs

There are two LEDs on front panel describing switch's status and two LEDs for each switch's port. Each LED can be off, light with green or orange color, or combination of both giving yellow. For more details please refer to following subsections.

3.7.1 Status LEDs

The status LED is placed together with power indicator LED on the left side of switch's front panel. The status LED is the right one.

During barebox/kernel boot the status LED is off. When `startup-mb.sh` starts the LED is set to yellow. If the HAL starts successfully then the LED is set to green. If the HAL caught a SIGNAL sent by other process, HAL sets the status LED to orange. When reboot is performed status LED is turned off.

3.7.2 Ports' LEDs

Under each switch's port there are two LEDs. The left LED is on when particular port is populated with a SFP and the link is up. It's color is dependent on the configured function. For ports configured as a slave the LED is green, for non-wr ports the LED is orange. For ports configured as a master and other cases (including wrong configuration) the left LED is yellow. The right LED blinks when packet is transmitted or received on a port.

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 DHCP 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 12](#) 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. Users are not expected to change any configuration, though, as further updates may fail.

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 startup time is a few seconds long 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 *boot from TFTP script* menu entry looks for the script using three different names, from most specific to most generic; the first found is be used. When using the boot script, the WRS first performs a DHCP query, and then uses that IP address to retrieve the script using the following names (the `eth0.ethaddr` is stored by the manufacturer in static storage and retrieved by the boot loader; the `eth0.ipaddr` comes from DHCP):

```
wrboot-$eth0.ethaddr
$eth0.ipaddr/wrboot
wrboot
```

As an example, the following excerpt shows what I see in my logs when only providing *wrboot*. The last message uses a different IP address because my script forces a static address into the kernel, whereas the initial one was assigned to the boot loader using DHCP.

```
dhcpcd: 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 a number of known-working *wrboot* scripts as examples;

wrboot-static-ip

The script forces a static IP address, server and gateway, and a custom mount point for an NFS-root system.

wrboot-dhcp

The script preserves the DHCP-assigned address, and runs a custom NFS-root system.

wrboot-install

This performs an installation, by loading everything to RAM and forcing install mode. Please check comments in the script.

wrboot-nand

This script is a copy of the default boot script executed by standalone switches. Booting from a script allows changing the kernel command line or anything else it may be useful to developers.

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 WRS Command-Line Tools

Tools are build from source files in `userspace/tools` while the scripts are copied directly from `userspace/rootfs_override/wr/bin`.

The following tools and scripts are provided:

load-virtex**load-lm32**

They load into the FPGA the gateway and the LM32 application. They are used by the init scripts of the Linux system. The LM32 loader can also change variables in the loaded binary, and read or write variables without stopping the running CPU. This is limited to 32-bit integer variables, though. See the commit message for details.

mapper**wmapper**

The former reads from a file using *mmap* (usually you run it on */dev/mem*) and writes to *stdout*. The latter read from *stdin* and writes using *mmap*. They are classic tools distributed in the *Linux Device Drivers* examples since 1998.

com

The program is a simple program for talking with serial ports.

wr_phytool

A tool to read and write PHY registers in the switch.

- wr_mon** A simple monitor of White Rabbit status. It prints to *stdout* using the standard escape sequences for color output. The **-b** command line options removes color change (b/w).
- wr_date** The program can read or set the White Rabbit date. When setting, using “**wr_date set value**” assigns an arbitrary date, and “**wr_date set host**” passes the host time to White Rabbit. If the file `/etc/leap-seconds.list` exists, it is used to pass the TAI offset to the kernel, and to consider it in setting White Rabbit time to the current TAI value. The program is meant to prime the White Rabbit counter at boot time, and is run by `/etc/init.d/wr_date` – this script uses NTP to set host time as a first step, if `/wr/etc/wr_date.conf` exists and includes a line of the form `ntpserver 192.168.16.1`.
With “**wr_date get**” you can read White Rabbit time, and by using **wr_date get tohost** you can set host time from White Rabbit time. This can be useful in slave switches that are not synced to NTP at boot.
- wrs_version** Print information about the SW & HW version of the WRS. Please check the help message.
- shw_ver** A symbolic link to **wrs_version**, to be compatible with older versions that used this tool name. The name is inconsistent with anything else in the switch, so it is being replaced.
- wrs_vlans** The tool allows to configure and unconfigure the VLAN settings for each port and for the RTU daemon. The **--help** option lists all configuration items of the tool.
- apply_dot-config** The script is used to apply **dot-config** settings to the current configuration files. It is run at boot time before any service is started. The **dot-config** mechanism is documented in [Chapter 3 \[Configuration of the Device\]](#), page 3.
- assembly_ppsi_conf.sh** The script is used to assembly ppsi configuration file based on information stored in `PORTxx_PARAMS`. This script is called by *apply_dot-config*.
- change_dot-config** This script changes the current **dot-config** file. It is designed to be the back-end of the web interface, when changing configuration items. The script does nothing to *apply* the changes, and it only performs editing. It is the responsibility of the caller to ensure the proper service is restarted with the new configuration.
- sdb-read** The tool, copied from the **fpga-config-space** project, is documented in the next section,
- wrs_auxclk** The tool allows to setup the parameters of a clock generated on the *clk2* SMC.

5.1 sdb-read

[Note: this documentation section comes from the **ohwr** project called **fpga-config-space**.]

The *sdb-read* program can be used to access an *sdbfs* image stored in a disk file or an FPGA area in physical memory. It works both as *ls* (to list the files included in the image) and as *cat* (to print to its own *stdout* one of the files that live in the binary image).

The program can be used in three ways:

sdb-read [options] <image-file>

This invocation lists the contents of the image. With **-l** the listing is *long*, including more information than the file name.

sdb-read [options] <image-file> <filename>

When called with two arguments, the program prints to *stdout* the content of the named file, extracted from the image. Please note that if the file has been over-sized at creation time, the whole allocated data area is printed to standard output.

sdb-read [options] <image-file> <hex-vendor>:<hex-device>

If the second argument is built as two hex numbers separated by a colon, then the program uses them as vendor-id and device-id to find the file. If more than one file have the same identifiers, the *first* of them is printed.

The following option flags are supported:

-l

For listing, use *long* format. A *verbose* format will be added later.

-e <entrypoint>

Specify the offset of the magic number in the image file.

-m <size>@<addr>

-m <addr>+<size>

Either form is used to specify a memory range. This is the preferred way to read from a memory-mapped area, like an FPGA memory space. Please note that in general you should not read a “file” in FPGA space, because this would mean read all device registers. The form “<image-file> <filename>” is thus discouraged for in-memory SDB trees (i.e. where <image-file> is /dev/mem).

-r

Register the device with a *read* method instead of the *data* pointer. In this way the tool can be used to test the library with either access method. If *mmap* fails on the file (e.g., it is a non-mappable device), *read* is used automatically, irrespective of **-r**.

5.2 wrs_vlans

The *wrs_vlans* shell tool can be used to setup VLANs in the switch. The configuration is divided into two parts:

wrs_vlans --ep <port number or range> [options]

Per-port Endpoint VLAN configuration. Used to set VID and priority for ingress frames tagging, egress untagging and port mode to:

- ACCESS - tags untagged frames with configured VID and priority, drops tagged frames not belonging to the configured VLAN
- TRUNK - passes only tagged frames, drops all untagged frames
- Disabled - VLANs disabled
- Unqualified port - passes all frames regardless of VLAN configuration

wrs_vlans --rvid <vid> [options]

Per-VLAN configuration of the Routing Table Unit, used to configure port mask describing which port belong to a given VLAN. RTU uses this information to be able to forward incoming frames only to ports inside the VLAN.

Both per-port Endpoint and per-VLAN RTU configuration has to be performed in order to have a full VLAN setup on a WR Switch.

For per-port configuration, multiple ways of specifying ports are supported:

`wrs_vlans --ep 1 [options]`

Selected configuration will be applied only to port 1.

`wrs_vlans --ep 1,3,4 [options]`

Selected configuration will be applied to ports 1,3 and 4.

`wrs_vlans --ep 5-8 [options]`

Selected configuration will be applied to port range 5 to 8.

`wrs_vlans --ep 5-8,15 [options]`

Selected configuration will be applied to port range 5 to 8 and port 15.

To configure each Endpoint the following options may be used:

`--emode <mode No.>`

Sets qmode for a port (0 - ACCESS, 1 - TRUNK, 2 - disabled, 3 - unqualified)

`--evid <vid>`

Sets VLAN id for tagging ingress frames.

`--eprio <priority>`

Sets priority for tagging ingress frames.

`--eumask <hex mask>`

Sets mask for egress untagging. By default, if you configure ingress tagging, all VLANs are untagged on egress.

To configure VLANs in RTU the tool has to be used with parameter specifying VLAN id to be set up and then the list of configuration options:

`wrs_vlans --rvid <vid> [options]`

Possible RTU VLAN configuration options:

`--rmask <hex mask>`

Mask defines which physical ports of the WRS belong to a configured VLAN.

`--rfid <fid>`

Assigns filtering ID *fid* to the configured VLAN. Multiple VLANs can be configured to have the same *fid*. This way they create a group where learning a new MAC address in one VLAN implies learning this MAC in the rest of VLANs in the group as well.

`--rprio <prio>`

Forces 802.1q priority override for VLAN. Setting *prio* to -1, cancels the override.

`--rdrop <1/0>`

Forces (if set to 1) or disables (if set to 0) frames drop for the configured VLAN.

`--del`

Deletes selected VLAN from the RTU configuration.

In addition to that *wrs_vlans* can be also used to display and clear current VLAN configuration of the switch:

`wrs_vlans --elist`

Current Endpoints VLAN configuration

`wrs_vlans --list`

Current RTU VLAN configuration.

`wrs_vlans --clear`

wrs_vlans tool can be called multiple times to make a set of per-port and per-VLAN configurations. However, it is also possible to configure multiple ports/VLANs in one go. For example to configure ports 0,1,2,5 to VLAN 5 and ports 3,4 to VLAN 6 with tagging ingress frames one could call *wrs_vlans* with these parameters:

```
wrs_vlans --ep 0-2,5 --emode 0 --evid 5 --ep 3,4 --emode 0 --evid 6 \
--rvid 5 --rmask 0x27 --rvid 6 --rmask 0x18
```

5.3 wrs_auxclk

The *wrs_auxclk* shell tool can be used to configure parameters of a clock signal generated on the *clk2* SMC connector.

Note: you need to have WRS hardware at least in version 3.4 to have *clk2* output.

By default *wrs_auxclk* is called by init scripts to generate 10MHz clock signal with 50% duty cycle. This configuration can be modified by using various options:

--freq <f>

Desired frequency of the generated clock signal in MHz. Available range from 4kHz to 250MHz.

--duty <frac>

Desired duty cycle given as a fraction (e.g. 0.5, 0.4).

--cshift <csh>

Coarse shift (granularity 2ns) of the generated clock signal. This parameter can be used to get desired delay relation between generated 1-PPS and *clk2*. The delay between 1-PPS and *clk2* is constant for a given bitstream but may be different for various hardware versions and re-synthesized gateware. Therefore it should be measured and adjusted only once for given hardware and gateware version.

--sigdel <steps>

Clock signal generated from the FPGA is cleaned by a discrete flip-flop. It may happen that generated aux clock is in phase with the flip-flop clock. In that case it is visible on the oscilloscope that *clk2* clock is jittering by 4ns. The **--sigdel** parameter allows to add a precise delay to the FPGA-generated clock to avoid such jitter. This delay is specified in steps, where each step is around 150ps. This value, same as the **--cshift** parameter, is constant for a given bitstream so should be verified only once.

--ppshift <steps>

If one needs to precisely align 1-PPS output with the *clk2* aux clock using **--cshift** parameter is not enough as it has 4ns granularity. In that case **--ppshift** lets you shift 1-PPS output by a configured number of 150ps steps. However, please have in mind that 1-PPS output is used as a reference for WR calibration procedure. Therefore, once this parameter is modified, the device should be re-calibrated. Otherwise, 1-PPS output will be shifted from the WR timescale by <steps>*150ps.

6 SNMP Support

The White Rabbit Switch supports SNMP. 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. 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.

There is currently no support for traps.

6.1 The WRS MIB

This section contain a summary of the WR-SWITCH-MIB, for details please refer to the document *White Rabbit Switch: Failures and Diagnostics*. Objects from 96.100.2 to 96.100.5 are obsolete, they were used during early implementation of switch's snmp.

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.

96.100.6

wrsStatus – MIB's branch with collective statuses of entire switch.

96.100.7

wrsExpertStatus – Branch with detailed statuses of switch subsystems.

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: 1
WR-SWITCH-MIB::wrsMainSystemStatus.0 = INTEGER: ok(1)
WR-SWITCH-MIB::wrsOSStatus.0 = INTEGER: ok(1)
WR-SWITCH-MIB::wrsTimingStatus.0 = INTEGER: ok(1)
[...]
WR-SWITCH-MIB::wrsConfigSource.0 = INTEGER: remote(4)
WR-SWITCH-MIB::wrsConfigSourceUrl.0 = STRING: tftp://192.168.1.1/config-192.168.1.10
WR-SWITCH-MIB::wrsBootConfigStatus.0 = INTEGER: ok(1)
WR-SWITCH-MIB::wrsBootHwinfoReadout.0 = INTEGER: ok(1)
WR-SWITCH-MIB::wrsBootLoadFPGA.0 = INTEGER: ok(1)
WR-SWITCH-MIB::wrsBootLoadLM32.0 = INTEGER: ok(1)
[...]
WR-SWITCH-MIB::wrsPstatsPortName.1 = STRING: wr0
WR-SWITCH-MIB::wrsPstatsPortName.2 = STRING: wr1
[...]
WR-SWITCH-MIB::wrsPstatsTXFrames.1 = Counter32: 232
WR-SWITCH-MIB::wrsPstatsTXFrames.2 = Counter32: 543
[...]
WR-SWITCH-MIB::wrsPstatsRXFrames.1 = Counter32: 255
WR-SWITCH-MIB::wrsPstatsRXFrames.2 = Counter32: 544
[...]
WR-SWITCH-MIB::wrsPtpServoState.1 = STRING: TRACK_PHASE
WR-SWITCH-MIB::wrsPtpServoStateN.1 = INTEGER: trackPhase(4)
WR-SWITCH-MIB::wrsPtpPhaseTracking.1 = INTEGER: tracking(2)
WR-SWITCH-MIB::wrsPtpSyncSource.1 = STRING:
WR-SWITCH-MIB::wrsPtpClockOffsetPs.1 = Counter64: 0
WR-SWITCH-MIB::wrsPtpClockOffsetPsHR.1 = INTEGER: 0
WR-SWITCH-MIB::wrsPtpSkew.1 = INTEGER: -1
WR-SWITCH-MIB::wrsPtpRTT.1 = Counter64: 943893
WR-SWITCH-MIB::wrsPtpLinkLength.1 = Gauge32: 91
WR-SWITCH-MIB::wrsPtpServoUpdates.1 = Counter32: 33
[...]
WR-SWITCH-MIB::wrsPortStatusPortName.1 = STRING: wr0
WR-SWITCH-MIB::wrsPortStatusPortName.2 = STRING: wr1
[...]
```

```

WR-SWITCH-MIB::wrsPortStatusLink.1 = INTEGER: up(2)
WR-SWITCH-MIB::wrsPortStatusLink.2 = INTEGER: up(2)
[...]
WR-SWITCH-MIB::wrsPortStatusConfiguredMode.1 = INTEGER: slave(2)
WR-SWITCH-MIB::wrsPortStatusConfiguredMode.2 = INTEGER: auto(4)
[...]
WR-SWITCH-MIB::wrsPortStatusSfpVN.1 = STRING: Axcen Photonics
WR-SWITCH-MIB::wrsPortStatusSfpVN.2 = STRING: Axcen Photonics
[...]
WR-SWITCH-MIB::wrsPortStatusSfpPN.1 = STRING: AXGE-3454-0531
WR-SWITCH-MIB::wrsPortStatusSfpPN.2 = STRING: AXGE-3454-0531
[...]

```

Another example is to print all objects exported by switch.

```

snmpwalk -c public -v 2c wrs -m all \
-M ${WRS_OUTPUT_DIR}/build/buildroot-2011.11/output/build/netsnmp-5.6.1.1/mibs/\
:${WRS_SWITCH_SW}/userspace/snmpd/ \
1

```

6.2 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:

CTR NAME	WR0	WR1	WR2	WR3	WR4	WR5	WR6	WR7	WR8	WR9	WR10	WR11	WR12	WR13	WR14	WR15	WR16	WR17
TX Underrun	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Overrun	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Invalid Code	137	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Sync Lost	5	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Frame Errors	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Filter Dropped	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PCS Errors	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Guard Frames	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Runt Frames	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX CRC Errors	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Pclass 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Pclass 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Pclass 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Pclass 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Pclass 4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Pclass 5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Pclass 6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Pclass 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TX Frames	1271	0	1981	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Frames	4544	0	576	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX Drop RTU Full	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PRIO 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PRIO 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PRIO 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PRIO 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PRIO 4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PRIO 5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PRIO 6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RX PRIO 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RTU Valid	4544	0	576	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RTU Responses	4544	0	576	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RTU Dropped	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FastMatch: Priority	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FastMatch: NonForward	4544	0	576	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FastMatch: Resp Valid	4544	0	576	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FullMatch: Resp Valid	4544	0	576	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Forwarded	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TRU Resp Valid	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Command `snmptable` can also be used to get similar results:

```

snmptable -Cw 80 -c public -v 2c 192.168.1.10 -m all \
-M ${WRS_OUTPUT_DIR}/build/buildroot-2011.11/output/build/netsnmp-5.6.1.1/mibs/\
:userspace/snmpd/ WR-SWITCH-MIB::wrsPstatsTable

```

Output is in text form and looks like:

```

SNMP table: WR-SWITCH-MIB::wrsPstatsTable

wrsPstatsPortName wrsPstatsTXUnderrun wrsPstatsRXOverrun wrsPstatsRXInvalidCode
wr0 0 0
wr1 0 0
wr2 0 0
wr3 0 0
wr4 0 0

```

wr5	0	0	0
wr6	0	0	0
wr7	0	0	0
wr8	0	0	0
wr9	0	0	0
wr10	0	0	0
wr11	0	0	0
wr12	0	0	0
wr13	0	0	0
wr14	0	0	0
wr15	0	0	0
wr16	0	0	0
wr17	0	0	0

SNMP table WR-SWITCH-MIB::wrsPstatsTable, part 2

wrsPstatsRXSyncLost	wrsPstatsRXPauseFrames	wrsPstatsRXFilterDropped
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

(...)

Unfortunately output due to the number of counters is very wide. Number of characters per line can be limited by switch `Cw`, in example was set to 80.

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.