

TRB1 Software Development Kit instructions

□

Contents

- [1 Introduction](#)
- [2 Prerequisites](#)
- [3 Compiling a standard firmware](#)
- [4 Selecting packages](#)
- [5 Compiling a single package](#)

Introduction

This article contains instructions on how to utilize Teltonika SDK packages for TRB1xx series devices. The resources used in the provided examples are:

- A TRB140 gateway
- [Ubuntu 16.04 OS](#)
- SDK version TRB1_R_GPL_00.02.05.2

However, the examples apply to other routers, SDK versions and operating systems. In most cases the difference will only be that of a visual nature.

Most examples provided in this page are independent from each other. But it is highly recommended to acquaint yourself with the basics by reading the "[Prerequisites](#)" and "[Compiling a standard firmware](#)" sections first as they contain information that will be necessary in order to understand some aspects of the other examples.

It should also be noted early that the first time you compile a firmware it may take **about two hours** to complete. On later attempts the duration is considerably lower.

Prerequisites

You will need:

- A PC, laptop or virtual machine running Linux OS (We recommend using Ubuntu 16.04 LTS)
- A TRB1xx series gateway
- An SDK intended for your router, which can be downloaded here: [Software Development Kit](#)

Compiling a standard firmware

First, you must install the packages required for the SDK to work. Open the Terminal application (**Ctrl + ALT + T**) and execute the following commands:

```
sudo apt update
```

```
sudo apt install -y gcc binutils bzip2 flex python3.6 perl make grep  
diffutils unzip gawk subversion zlib1g-dev build-essential u-boot-tools
```

Create a new folder (for this example I'll be using a directory called *TRB1* found at my **hard drive** directory) and extract the SDK archive inside it. You can achieve this with drag and drop or by executing this command via Terminal:

```
tar -xf ~/Downloads/TRB1_R_GPL_00.XX.YY.Z.tar.gz -C ~/TRB1
```

Note: don't forget to replace the file name and path in accordance with your own circumstances

Open a Terminal inside the SDK directory. You can change the directory in your current terminal (*cd ~/TRB1/openwrt-gpl-mdm9x07.Linux-x86_64*).

Once you open the Terminal you can compile a standard firmware by executing this command in the terminal:

```
make
```

If all is in order, the output should look something like this:



Note: the first time you compile a firmware file it may take up to **two hours** before it is complete. Don't close the Terminal window up until then. Once it is finished, you will find the firmware in the *./bin/targets/mdm9x07/generic/tltFws* directory. It should contain a file *TRB1_R_GPL_00.XX.YY.Z_WEBUI.bin*, it can be used to upgrade your router's firmware via its web interface.

You can speed up this process by passing an extra argument **-j**. This argument then compiles the necessary packages in parallel. To know how much jobs you can pass to the compiling process execute this command in the terminal:

```
nproc
```

This commands output should be a number. This number tells how much processing units are available to the current process. Now execute this command in the terminal:


```
make -j<nproc_output>
```



Selecting packages

To select which packages should be compiled into the firmware is through a special menu called **Kconfig**. To enter the **Kconfig**, standing in your projects root directory execute this command in the terminal:

```
make menuconfig
```

After executing this command you should see a view like in the picture below. 

To move around in the **Kconfig** use the arrows on the keyboard, space bar to select or to unselect the package, and enter key to enter sub directories of packages.

Note: Do not unselect any package if you do not know what you are doing.

1. Packages have three markings:

1. <*> - this symbol says that the package will be compiled and added to the firmware
2. <M> - this symbol says that the package will be compiled but will not be added to the firmware
3. < > - this symbol says that the package will not be compiled and will not be added to the firmware



By default all packages that were marked as to be compiled in the process, can be found separately from the firmware. Packages can be found in directory **/path/to/trb1/project/openwrt-gpl-mdm9x07.Linux-x86_64/bin/packages/arm_cortex-a7_neon-vfpv4**.

This directory also has five sub directories. Packages are categorized in these directories by their type.

Compiling a single package

If you ever need to compile a single package you do not need to compile the whole firmware. As an example we will try to compile a text editor called **nano**.

2. Now we need to do these steps:

1. Open the **Kconfig** by executing this command in the terminal:

```
make menuconfig
```

2. Navigate to the menu section of **nano** text editor. It can be found under "Utilities→Editors"
3. Select **nano** package
4. Three times press the "**Exit**" entry



5. Save the configuration by selecting "**Yes**" in the prompted window



3. When the package is selected we need to compile and install it:

1. While standing in root directory of the project execute this command in the terminal:

```
make package/nano/compile
```



2. The compiled package can be found in the directory **/path/to/trb1/project/openwrt-gpl-mdm9x07.Linux-x86_64/bin/packages/arm_cortex-a7_neon-vfpv4/packages**. Upload the **nano** install file into the router by executing this command in the terminal:

```
scp /path/to/trb1/project/openwrt-gpl-mdm9x07.Linux-  
x86_64/bin/packages/arm_cortex-a7_neon-  
vfpv4/packages/nano_4.3-1_arm_cortex-a7_neon-vfpv4.ipk  
root@<device_ip_addr>:/tmp
```

3. Connect to your routers command line interface by executing this command in the terminal:

```
ssh root@<device_ip_addr>
```

4. Change your directory to **/tmp** directory

5. Install the **nano** text editor by executing this command in the terminal:

```
opkg install nano_4.3-1_arm_cortex-a7_neon-vfpv4.ipk
```

6. Try out the nano text editor by executing this command:

```
nano example
```



Note: If the package that you want to compile isn't selected in the **Kconfig**, the install file of the package will not be created