L-846e I.MX6 Android BSP Manual Head

__NOCACHE__

L-846e I.MX6 Android BSP Manual Head	
Document Title	Has Document Title::L-846e_I.MX6_Android_BSP_Manual
Document Type	Has Document Type::BSP Manual
Yocto Page	Has Yocto Page::
Article Number	Has Article Number::L-846e
Status	Has Status::Published
Release Date	Has Release Date::2017/09/26
Is Branch of	Is Branch of::L-846e_I.MX6_Android_BSP_Manual_Head

Introduction to Android

Android's growth is phenomenal. In a very short time span, it has succeeded in becoming one of the top mobile platforms in the market. Clearly, the unique combination of open source licensing, aggressive go to market and trendy interface is bearing fruit form Google's Android team. Needless to say, the massive user uptake generated by Android has not gone unnoticed by handset manufacturers, mobile network operators, silicon manufacturers, and app developers. Products, apps, and devices "for," "compatible with, "or "based on" Android seem to be coming out ever so fast. Beyond its mobile success, however, Android is also attracting the attention of yet another, unintended crowd embedded systems developers. While a large number of embedded devices have little to no human interface, a substantial number of devices that would traditionally be considered "embedded" do have user interfaces. For a goodly number of modern machines, in addition to pure technical functionality, developers creating user-facing devices must also contend with human-computer interaction (HCI) factors. The Android mobile platform is being adopted by a broad range of embedded devices that span multiple industries and segments.

See, Embedded android - **Karim Yaghmour**.

- Official Introduction to Android: https://developer.android.com/guide/index.html

Introduction to the Phytec Android BSP

Besides Yocto based Board support Packages, Phytec also provides Android based BSPs for a few chosen Products. Such as the phyBOARD-Mira and the phyCORE-RK3288 Rapid Development Kit. Vendors like NXP or Rockchip are fully supporting the Android operating system through Board support Packages. Phytec adds support for it's own Boards on top of the work done by the semiconductor companies. In general we can say, that these are the same BSPs.

Building the Android BSP

Please note, that Android will require quite a lot of RAM on the host system especially while building the browser environment. So at least 8GB are recommended for crashfree building. For disc space at least 120 GB should be allocated.

Preparations

We have decided to use **Docker** to create a reproducible building environment. Install Docker on your host machine using the following command:

```
sudo apt-get install docker.io
```

All Android SDKs can be build with the following Dockerfile:

```
FROM ubuntu:14.04

RUN apt-get update && apt-get -y install \
 git-core \
 gnupg \
 flex \
 bison \
 gperf \
 build-essential \
 zip \
 curl \
 zlib1g-dev \
 gcc-multilib \
 g++-multilib \
 libc6-dev-i386 \
 lib32ncurses5-dev \
 x11proto-core-dev \
 libx11-dev \
 lib32z-dev \
 ccache \
 libgl1-mesa-dev \
 libxml2-utils \
 xsltproc \
 unzip \
 openjdk-7-jdk \
 lzop \
 bc
```

Manage Docker as a non-root user:

The docker daemon binds to a Unix socket instead of a TCP port. By default that Unix socket is owned by the user root and other users can only access it using sudo. The docker daemon always runs as the root user. If you don't want to use sudo when you use the docker command, create a Unix group called docker and add users to it. When the docker daemon starts, it makes the ownership of the Unix socket read/writable by the docker group.

- Add the docker group if it doesn't already exist:

```
sudo groupadd docker
```

- Add the connected user "$USER" to the docker group. Change the user name to match your preferred user if you do not want to use your current user:

```
sudo gpasswd -a $USER docker
```

- Either do a

```
newgrp docker
```

or log out/in to activate the changes to groups.

- You can use

```
docker run hello-world
```

to check if you can run docker without sudo.

Create a docker image (Paste the above content into a Dockerfile) and run:

```
docker build .
```

Check if your image was created with:

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
<none>              <none>              6a00cd10794c        About a minute ago   656.6 MB
ubuntu              14.04               c69811d4e993        10 days ago          188 MB
```

Now you can enter to the new created docker image in interactive mode:

```
docker run -it 6a00cd10794c /bin/bash
```

Get the Android BSP

After the build system preparations are completed, you can now acquire the Android sources with these commands. This will download the official sources only (from Android Open Srouce Project).

NXP does not provide own git repositories for their own android related changes. So the additional patches have to be applied on top to the AOSP repositories.

The final source base we are using is compiled in the following way:

```
AOSP repositories
+ NXP patch set (Manual download required, nxp.com)
+ AOSP repositories touched by PHYTEC (hosted on the Phytec git server)
```

Additional patches will be applied in the following steps.

The Android source tree is located in a Git repository hosted by Google. The Git repository includes metadata for the Android source, including those related to changes to the source and the date they were made. This document describes how to download the source tree for a specific Android code-line.

Download AOSP (using repo)

Note: You don't have to be inside a docker container for the below steps. Chapter Get the Android BSP is only describing the android source fetching process.

- Download the Repo tool and ensure that it is executable:

```
curl https://storage.googleapis.com/git-repo-downloads/repo > /usr/local/bin
chmod a+x /usr/local/bin/repo
```

Repo is a tool that makes it easier to work with Git in the context of Android. See also Downloading the Source [1].

- Download the Android source; Create a directory and checkout the source:

```
mkdir android_imx; cd android_imx
repo init -u ssh://git@git.phytec.de/phydroid -b imx6 -m PD-ALPHA1.xml
repo sync
```

Please be aware that *repo sync* may take several hours depending on your system/internet connection.

Download and apply the NXP i.MX6 patch set

This patch set contains code which is not covered by any AOSP (Apache) or GPL license. It is covered by the Freescale EULA (LA_OPT_FSL_OPEN_3RD_PARTY_IP v6 February 2015). Due to the above mentioned EULA, PHYTEC can not host git repositories containing Freescale's NDA licensed code.

- Download both packages *IMX6-L500-100-ANDROID-SOURCE-BSP* and *IMX6_L500_101_ANDROID_SOURCE_BSP* from the following URL. You will need to register an NXP account and accept the Freescale EULA.

```
http://www.nxp.com/products/software-and-tools/
software-development-tools/i.mx-software-and-tools/
android-os-for-i.
mx-applications-processors:IMXANDROID?tab=Design_Tools_Tab
```

- Extract the patches from the archives to */tmp/nxp*.

```
mkdir /tmp/nxp; cd /tmp/nxp
tar xzf ~/Downloads/android_L5.0.0_1.0.0-ga_core_source.gz
tar xzf ~/Downloads/android_L5.0.0_1.0.1_source.tar.gz
tar xzf ./android_L5.0.0_1.0.0-ga_core_source/code/l5.0.0_1.0.0-ga.tar.gz
tar xzf ./android_L5.0.0_1.0.1_source/code/L5.0.0_1.0.1.tar.gz
```

Please note that you don't need to use the shell scripts which can be found inside the archive files.

- Create the following git repositories inside your android checkout directory (This documentation used *android_imx*)

fsl_vpu_omx:

```
mkdir -p ~/android_imx/external/fsl_vpu_omx
cd  ~/android_imx/external/fsl_vpu_omx
git init
git am /tmp/nxp/l5.0.0_1.0.0-ga/platform/external/fsl_vpu_omx.git/*.patch
```

linux-firmware-imx:

```
mkdir -p ~/android_imx/external/linux-firmware-imx
cd ~/android_imx/external/linux-firmware-imx
git init
git am /tmp/nxp/l5.0.0_1.0.0-ga/platform/external/linux-firmware-imx.git/*.patch
```

linux-lib:

```
mkdir -p ~/android_imx/external/linux-lib
cd ~/android_imx/external/linux-lib
git init
git am /tmp/nxp/l5.0.0_1.0.0-ga/platform/external/linux-lib.git/*.patch
git am /tmp/nxp/L5.0.0_1.0.1/platform/external/linux-lib.git/*.patch
```

linux-test:

```
mkdir -p ~/android_imx/external/linux-test
cd ~/android_imx/external/linux-test
git init
git am /tmp/nxp/l5.0.0_1.0.0-ga/platform/external/linux-test.git/*.patch
```

powerdebug:

```
mkdir -p ~/android_imx/external/powerdebug
cd ~/android_imx/external/powerdebug
git init
git am /tmp/nxp/l5.0.0_1.0.0-ga/platform/external/powerdebug.git/*.patch
```

fsl-proprietary:

```
mkdir -p ~/android_imx/device/fsl-proprietary
cd ~/android_imx/device/fsl-proprietary
git init
git am /tmp/nxp/l5.0.0_1.0.0-ga/platform/external/fsl-proprietary.git/*.patch
```

Starting the Build Process

- Enter into a docker image

```
docker run -v ~/android_imx:/android_imx -it 6a00cd10794c /bin/bash
```

You can use the *-v* option to mount your android source directory to the docker image. After that you should see the docker shell and your mounted directory *android_imx*:

```
root@4ad5c988bed2:/# ls
android_imx  boot  etc   lib    lib64   media  opt   root  sbin  sys  usr
bin          dev   home  lib32  libx32  mnt    proc  run   srv   tmp  var
```

- Configure the build process:

```
cd /android_imx
```

Optionally you can set an android output directory. The building process will create quite a lot of files and I/O workload, so this might help to improve the time spent compiling.

```
export OUT_DIR_COMMON_BASE=/android_imx/bdir
```

- Select the output image type (Is it a SD card image or a ubifs image for a NAND flash?)

To create Images for booting from a SD card you have to change the board configuration accordingly. Open /android_imx/vendor/SIGMA/miraq/BoardConfig.mk and enable the ext4 images.

```
BUILD_TARGET_FS ?= ext4
include device/fsl/imx6/imx6_target_fs.mk
```

and disable the ubifs images:

```
# uncomment below line if you want to use a NAND image
#TARGET_USERIMAGES_USE_UBIFS := true
```

and select the correct fstab file:

```
$(_thisdir)/files/fstab.sd:root/fstab.${_hw}
```

The default configuration is for ubifs images. So if you want to build your image again for NAND flashes (ubifs), you have to revert the changes noted above, otherwise the settings are already correct.

- Start the build process:

```
. build/envsetup.sh
lunch miraq-user
make update-api
make -j32
```

The *lunch miraq-user* command will toggle an user based build. It means you will have no access to the android shell. Use *lunch miraq-eng* for to enable the shell and other options.

Build Variants

eng: This is the default flavor. A plain make is the same as make eng.

```
- Installs modules tagged with: eng, debug, user, and/or development.
- Installs non-APK modules that have no tags specified.
- Installs APKs according to the product definition files, in addition to tagged APKs.
- ro.secure=0
- ro.debuggable=1
- ro.kernel.android.checkjni=1
- adb is enabled by default.
```

user: This is the flavor intended to be the final release bits.

```
- Installs modules tagged with user.
- Installs non-APK modules that have no tags specified.
- Installs APKs according to the product definition files; tags are ignored for APK modules.
- ro.secure=1
- ro.debuggable=0
- adb is disabled by default.
```

Android Documentation: Build Variants [2]

Compiling barebox

Use the prebuild cross toolchain provided by android for your barebox build:

```
cd /android_imx/barebox
export PATH=/android_imx/prebuilts/gcc/linux-x86/arm/arm-eabi-4.8/bin/:$PATH
make ARCH=arm V=0 CROSS_COMPILE=arm-eabi- imx_v7_defconfig
make ARCH=arm V=0 CROSS_COMPILE=arm-eabi- oldconfig
make ARCH=arm V=0 CROSS_COMPILE=arm-eabi- all
```

Barebox generates a lot of images. Please select the proper barebox image according to your hardware. Currently our Android SDK does only support the

```
/android_imx/barebox/images/barebox-phytec-phycore-imx6q-som-nand-1gib.img
```

image.

Compiling the linux kernel

Note: This step have to be done after you have compiled the android base code.

Since the former created ramdisk needs to be embeeded into the kernel, we have to tell the kernel where to find it. Therefore we place a symlink in the source directory. (Depending if you used OUT_DIR_COMMON_BASE your ramdisk location might differ)

```
cd /android_imx/kernel
ln -s /android_imx/bdir/target/product/miraq/ramdisk.img ramdisk.cpio.gz
```

After that, we build the kernel config and check if the kernel will search for the ramdisk in the right place.

```
make ARCH=arm android_miraq_defconfig
grep CONFIG_INITRAMFS_SOURCE .config
CONFIG_INITRAMFS_SOURCE="ramdisk.cpio.gz"
```

Now we can build the linux kernel.

```
make ARCH=arm CROSS_COMPILE=/android_imx/prebuilts/gcc/linux-x86/arm/arm-eabi-4.8/bin/arm-eabi- -j32
```

Currently our Android SDK does only support the **imx6q-phytec-mira-rdk-nand.dtb** device tree.

```
/android_imx/kernel/arch/arm/boot/dts/imx6q-phytec-mira-rdk-nand.dtb
/android_imx/kernel/arch/arm/boot/zImage
```

BSP Images

- **Barebox**: barebox-phytec-phycore-imx6q-som-nand-1gib.img
- **Kernel**: zImage
- **Kernel device tree file**: imx6q-phytec-mira-rdk-nand.dtb
- **Android system image**: system.img

Creating bootable images

Currently we support two different boot devices, NAND flash and SD cards. This chapter will guide you through the process of creating a bootable system. We explain how to use the barebox bootloader to update the images in NAND Flashs.

Development Host Preparations (Updating from Network)

On the development host a TFTP server must be installed and configured. The following tools will be needed to boot the Kernel from Ethernet:

1. a TFTP server and
2. a tool for starting/stopping a service.

- For *Ubuntu* install:

```
host$ sudo apt-get install tftpd-hpa xinetd
```

After the installation of the packages you have to configure the TFTP server.

Set up for the TFTP server:

- Edit */etc/xinetd.d/tftp*.

```
service tftp
{
      protocol = udp
      port = 69
      socket_type = dgram
      wait = yes
      user = root
        server = /usr/sbin/in.tftpd
      server_args = -s /tftpboot
      disable = no
}
```

- Create a directory to store the TFTP files:

```
host$ sudo mkdir /tftpboot
host$ sudo chmod -R 777 /tftpboot
host$ sudo chown -R nobody /tftpboot
```

- Configure a static IP address for the appropriate interface:

```
host$ ifconfig eth1
```

You will receive:

```
eth1      Link encap:Ethernet  HWaddr 00:11:6b:98:e3:47
          inet addr:192.168.3.10  Bcast:192.168.3.255  Mask:255.255.255.0
```

- Restart the services to pick-up the configuration changes:

```
host$ sudo service tftpd-hpa restart
```

- Now connect the first port of the board to your host system, configure the board to network boot and start it.

Usually TFTP servers are using the */tftpboot* directory to fetch files from. If you built your own images, please copy them from the BSP's build directory to there.

We also need a network connection between the embedded board and the TFTP server. The server should be set to IP 192.168.3.10 and netmask 255.255.255.0.

After the installation of the TFTP server, an NFS server needs to be installed, too. The NFS server is not restricted to a certain file system location, so all we have to do on most distributions is to modify the file */etc/exports* and export our root filesystem to the embedded network. In this example file the whole work directory is exported, and the "lab network" address of the development host is 192.168.3.10, so the IP addresses have to be adapted to the local needs:

```
/home/<user>/<rootfspath> 192.168.3.10/255.255.255.0(rw,no_root_squash,sync,no_subtree_check)
```

Where <*user*> must be replaced with your home directory name. The <*rootfspath*> can be set to a folder which contains a rootfs *tar.gz* image extracted with *sudo*.

Preparations on the Embedded Board (Updating from Network)

- To find out the Ethernet settings in the bootloader of the target type:

```
bootloader$ ifup eth0
bootloader$ devinfo eth0
```

With your development host set to IP 192.168.3.10 and netmask 255.255.255.0, the target should return:

```
ipaddr=192.168.3.11
netmask=255.255.255.0
gateway=192.168.3.10
serverip=192.168.3.10
```

- If you need to change something, type

```
bootloader$ edit /env/network/eth0
```

Here you can also change the IP address to DHCP instead of using a static one.

- Just configure:

```
ip=dhcp
```

- Edit the settings if necessary and save them by leaving the editor with ***CTRL+D***.
- Type *saveenv* if you made any changes.
- Set up paths for TFTP and NFS in the file */env/boot/net*.

Updating from Network

i.MX 6 boards that have an Ethernet connector can be updated over network. Be sure to set up the development host correctly. The IP needs to be set to 192.168.3.10, the netmask to 255.255.255.0, and a TFTP server needs to be available.

- Boot the system using any boot device available.
- Press any key to stop autoboot, then type:

```
bootloader$ ifup eth0
bootloader$ devinfo eth0
```

The Ethernet interfaces should be configured like this:

```
ipaddr=192.168.3.11
netmask=255.255.255.0
gateway=192.168.3.10
serverip=192.168.3.10
```

If a DHCP server is available, it is also possible to set:

```
ip=dhcp
```

If you need to change something:

- Type:

```
bootloader$ edit /env/network/eth0
```

- Edit the settings, save them by leaving the editor with ***CTRL+D*** and type:

```
bootloader$ saveenv
```

- Reboot the board.

Updating NAND Flash from Network

To update the bootloader you may use the *barebox_update* command. This provides a handler which automatically erases and flashes two copies of the *barebox* image into the NAND Flash. This makes the system more robust against ECC issues. If one block is corrupted the ROM loader does use the next block. This handler also creates an FCB table in the NAND. The FCB table is needed from the ROM loader to boot from NAND.

- Type:

```
bootloader$ barebox_update -t nand /mnt/tftp/barebox.bin
```

On startup the TFTP server is automatically mounted to */mnt/tftp*. So copying an image from TFTP to flash can be done in one step. Do not get confused when doing an *ls* on the */mnt/tftp* folder. The TFTP protocol does not support anything like *ls* so the folder will appear to be empty.

We recommend to also erase the environment of the old *barebox*. Otherwise the new *barebox* would use the old environment.

- First erase the old environment with:

```
bootloader$ erase /dev/nand0.barebox-environment.bb
```

After erasing the environment, you have to reset your board. Otherwise the *barebox* still uses the old environment.

- To reset your board in order to get the new *barebox* running type:

```
bootloader$ reset
```

- Create UBI volumes for Linux kernel, oftree and android root filesystem, cache, device, data in NAND:

```
bootloader$ ubiformat /dev/nand0.root
bootloader$ ubiattach /dev/nand0.root
```

```
bootloader$ ubimkvol -t static /dev/nand0.root.ubi kernel 8M
bootloader$ ubimkvol -t static /dev/nand0.root.ubi oftree 1M
bootloader$ ubimkvol /dev/nand0.root.ubi system 400M
bootloader$ ubimkvol /dev/nand0.root.ubi cache 100M
bootloader$ ubimkvol /dev/nand0.root.ubi device 10M
bootloader$ ubimkvol -t dynamic /dev/nand0.root.ubi data 0
```

- Now get the Linux kernel and oftree from your TFTP server and store it also into the NAND Flash with:

```
bootloader$ ubiupdatevol /dev/nand0.root.ubi.kernel /mnt/tftp/linuximage
bootloader$ ubiupdatevol /dev/nand0.root.ubi.oftree /mnt/tftp/oftree
```

Note: If you get the following message */mnt/tftp/linuximage has unknown filesize, this is not supported* you have to copy the *linux* kernel and the *oftree* before with *cp*.

```
bootloader$ cp /mnt/tftp/linuximage .
bootloader$ cp /mnt/tftp/oftree .
```

- For flashing *Android*'s root filesystem into NAND, please use:

```
bootloader$ cp -v /mnt/tftp/system.img /dev/nand0.root.ubi.system
```

Updating from SD Card

Updating NAND Flash from SD Card

Note: The SD Card which can be generated using the tools described in Create a bootable SD Card can not be used for this process.

To update an i.MX 6 board from SD card the SD card used for updating must be mounted after the board is powered and the boot sequence is stopped on the bootloader prompt.

- Create a SD Card with one FAT32 formated partition
- Copy the following files to the partition

```
/android_imx/kernel/arch/arm/boot/zImage
/android_imx/kernel/arch/arm/boot/dts/imx6q-phytec-mira-rdk-nand.dtb
/android_imx/barebox/images/barebox-phytec-phycore-imx6q-som-nand-1gib.img
/android_imx/bdir/target/product/miraq/system.img
```

- Boot the system using any boot device available.
- Press any key to stop autoboot
- Insert your SD Card
- Create UBI volumes for Linux kernel, oftree and android root filesystem, cache, device, data in NAND:

```
bootloader$ ubiformat /dev/nand0.root
bootloader$ ubiattach /dev/nand0.root
```

```
bootloader$ ubimkvol -t static /dev/nand0.root.ubi kernel 8M
bootloader$ ubimkvol -t static /dev/nand0.root.ubi oftree 1M
bootloader$ ubimkvol /dev/nand0.root.ubi system 400M
bootloader$ ubimkvol /dev/nand0.root.ubi cache 100M
bootloader$ ubimkvol /dev/nand0.root.ubi device 10M
bootloader$ ubimkvol -t dynamic /dev/nand0.root.ubi data 0
```

- Copy the above mentioned files from your SD Card to the NAND flash

```
bootloader$ barebox_update -t nand /mnt/mmc/barebox.bin
bootloader$ erase /dev/nand0.barebox-environment.bb
```

```
ubiupdatevol /dev/nand0.root.ubi.kernel /mnt/mmc/zImage
ubiupdatevol /dev/nand0.root.ubi.oftree /mnt/mmc/imx6q-phytec-mira-rdk-nand.dtb
ubiupdatevol /dev/nand0.root.ubi.system /mnt/mmc/system.img
```

- Reboot your device

Create a bootable SD Card

Now you can create a bootable SD-Card with the flash script. Please have a look at it first and ensure that all variables are correct and your SD is located under the locations specified. (If you don't you might end up destroying your system/data.)

```
/android_imx/vendor/SIGMA/miraq/tools/flash.sh
```

After the flashing is completed you just have to insert the microSD into the device and toggle the boot switch for SD-Boot. Now Android should boot up.

Customizing the BSP

Hardware

The hardware is described in the Linux Kernel. The devicetree used is kernel/arch/arm/boot/dts/imx6q-phytec-mira-rdk-nand.dts

Software

The Android Image is defined in /android_imx/vendor/SIGMA/miraq.

There you can find the BoardConfig.mk, which describes the Image created (GPU used, build properties, model description, ...) and the miraq.mk describing which applications will be included.

Adding Software

To simply add an apk to the image I would recommend you create a separate recipe for it. (This example will create one for the DM.apk)

Switch into the recipes directory and create a directory for the APK.

```
cd /android_imx/vendor/SIGMA/miraq/recipes
mkdir DM
cd DM
```

Place you APK in this directory and create a build file (Android.mk). Here is an example:

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE_TAGS := optional
LOCAL_MODULE := DM
LOCAL_CERTIFICATE := PRESIGNED
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk
LOCAL_MODULE_CLASS := APPS
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)

include $(BUILD_PREBUILT)
```

Now you can add the new receipt to you miraq.mk and after a recompile it will be included into the system.img.

Install the Google Apps (Playstore)

There are multiple ways to accomplish that, so this is just one way.

First download a prepacked gapps binary for Android 5.0. We've tested this approach with the gapps-5.0.x-20150404-minimal-edition-signed.zip bundle from

```
http://www.cyanogenmod-forum.de/Thread-DOWNLOAD-Gapps-f%C3%BCr-Android-4-4-4-5-0-5-1
```

Put the archive on a SD card and insert it into the device. Connect to the device via the debug console and issue these commands to install the binarys into the system.

```
mount -o remount,rw /system

umask 022

cd /

busybox unzip /mnt/media_rw/extsd/gapps-5.0.x-20150404-minimal-edition-signed.zip -x META-INF/* -x optional/* -x install-optional.sh
```

```
cd /system

busybox unzip /mnt/media_rw/extsd/gapps-5.0.x-20150404-minimal-edition-signed.zip-x META-INF/* -x system/* -x install-optional.sh

cp -a optional/gms/430/priv-app/PrebuiltGmsCore optional/setup/tablet/priv-app/SetupWizard priv-app/

rm -r optional
```

Now reboot the device and you should have installed working Google Apps.

Android Rooting

To install SuperSU into the image you have to enable the build_flag BOARD_IAMROOTED in BoardConfig.mk.

This build flag will install all required additions and configure them.

References

[1] https://source.android.com/source/downloading

[2] https://source.android.com/source/add-device#build-variants

Article Sources and Contributors

L-846e I.MX6 Android BSP Manual Head *Source*: http://devwiki.phytec.de/mediawiki/index.php?oldid=33406 *Contributors*: Wegorov, Yastein