

Raymii.org ~~~~

I will lift up my eyes to the mountains, from whence cometh my help.

[Home](#) | [About](#) | [All pages](#) | [Cluster Status](#) | [RSS Feed](#)

Search

Building opkg .ipk packages by hand (for OpenEmbedded/Yocto/OpenWRT)

Published: 05-04-2019 | Author: Remy van Elst | [Text only version of this article](#)

! This post is over six years old. It may no longer be up to date. Opinions may have changed.

Table of Contents

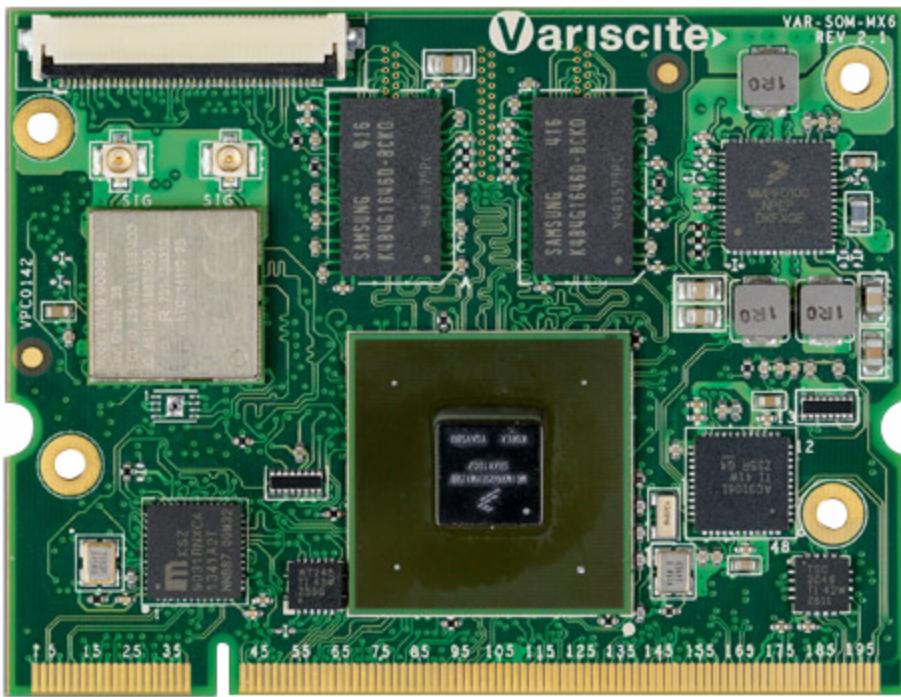
[IPK packages](#)

[Folder structure & Data](#)

[Control & Postint, preinst, postrm and prerm scripts](#)

[Packing it all up](#)

[gzip vs debian binary package \(mime type\)](#)



Variscite VAR-SOM-MX6 - a high-performance ARM System on Module that can run Yocto

.ipk packages are used by a variety of embedded linux systems, like routers running OpenWRT and appliances running on OpenEmbedded (Yocto). The opkg command installs these packages and OpenEmbedded comes with a set of tools to build .ipk packages.

Recently I had to create ipk packages in a scripted fashion for a few hundred systems, all unique per system. The .ipk packages includes a few software changes for debugging, a systemd service and one precompiled binary. The yocto build tools were not available on the machine where these packages would be made so I had to figure out how to make them by hand, which means, automatically. The packages are actually just compressed files containing a few control files and the data to be extracted on the filesystem.

This article will walk you through the steps of creating these packages by hand.

All steps are executed as the root user. The system the packages are built on is running Ubuntu 18.04. If you haven't got ar installed, make sure to

```
apt-get install binutils
```

IPK packages

An IPK package is very simple. It's like a .deb debian package, as in that is has both data and control files packaged up into an archive. The data will be extracted onto the filesystem

where the package is installed, the control files are used for dependency management and to execute pre and post install actions.

In my case, the `postinst` script is used to start the service (the binary we're packaging up). The `prerm` script is used to stop the service and disable it before uninstalling the package. The `postinst` script is used to check if the serial number matches the machine.

An ipk is an archive (either tar or ar or gzip) containing two archives (`control.tar.gz` & `data.tar.gz`) and a `debian-binary` file with the contents 2.0:

```
tar -tzf example_package_1.3.3.7.varam335x.ipk
```

Output:

```
./debian-binary  
./data.tar.gz  
./control.tar.gz
```

Folder structure & Data

The following folder structure is used for the package build. There is a main folder named `packages`, which has a subfolder for each machine based on the machines serial number. Under the machine folder there is a folder named after the package we're building (`examplepackage`), which has a `control` and `data` folder. The `data` folder contains the files that will be extracted on the filesystem and the `control` folder contains the pre and post scripts and some package information.

```
packages/serialnumber/  
|-- ipkbuild  
|   '-- example_package  
|       |-- control  
|           |-- control  
|           |-- postinst  
|           |-- preinst  
|           '-- prerm  
|       '-- data  
|           |-- usr  
|               '-- bin  
|                   '-- my_binary  
|           '-- lib
```

```

|       |   '-- systemd
|       |       '-- system
|       |           '-- example_package.service
|       '-- debian-binary
`-- example_package_1.3.3.7_varam335x.ipk

```

To create the folder structure listed above, use this command:

```
mkdir -p packages/serial/ipkbuild/example_package/{control,data}
```

Then copy all the files you need installed (including folder structure and permissions) into the data folder. As you can see in the above listing, my data folder contains one binary and a systemd script (to start that binary).

Make sure that your binaries have executable permissions and are for the correct architecture. A binary for a mipsel machine will not run on an armv7l, even if it's in that ipk package.

Control & Postint, preinst, postrm and prerm scripts

The control folder must contain at least a file named control. This has information on the package, like name, version, dependencies, etc. My control file is simple and contains just the bare minimum:

```
cat packages/serial/ipkbuild/example_package/control/control
```

Output:

```

Package: example_package
Version: 1.3.3.7
Architecture: varam335x
Maintainer: user@domain.tld
Description: This is an example IPK package
Priority: optional
Depends: systemd other_package

```

The debian-binary file must contain just 2.0:

```
echo 2.0 > packages/serial/ipkbuild/example_package/debian-binary
```

Some systems use this to check the MIME type of the package.

The postinst, postrm, preinst and prerm are executed in their respective phases during installation or removal. Exit code 0 means all is well and the action will continue. Other exit codes (>1) mean that something went wrong and the action will stop. By default these scripts are executed with sh, but that depends entirely on your embedded system. In my case I know bash is available, but make sure to hold back onto bash specifics.

My preinst file contains a check on the machine serial number. Since I build the packages for a specific machine, I know this beforehand. I want to make sure that packages can only run on the machine they're built for:

```
cat packages/serial/ipkbuild/example_package/control/preinst
```

Output:

```
#!/bin/bash
confserial=123456789
machineserial=`cat /example/serial.txt`
if [ $confserial -ne $machineserial ]; then
    echo "Configured serial does not match machine serial"
    exit 1
fi
```

Make sure this file is executable. It will not run otherwise, opkg will fail with a Permission Denied error.

```
chmod +x packages/serial/ipkbuild/example_package/control/preinst
```

The postinst file is executed after successfull installation. I use it to start the service we just installed:

```
cat packages/serial/ipkbuild/example_package/control/postinst
```

Output:

```
#!/bin/bash
systemctl --system daemon-reload
systemctl enable example_service
systemctl start example_service
```

This file must be executable as well:

```
chmod +x packages/serial/ipkbuild/example_package/control/postinst
```

The `prerm` file is used to stop the service and remove it from systemd:

```
cat packages/serial/ipkbuild/example_package/control/prerm
```

Output:

```
#!/bin/bash
systemctl stop example_service
systemctl disable example_service
systemctl --system daemon-reload
```

This one has to be executable as like all the others:

```
chmod +x packages/serial/ipkbuild/example_package/control/prerm
```

If you have all your data files and your control files in the correct folder you can continue to package it all up.

Packing it all up

The archive files must not contain any paths, therefore we create them in the folder structure we've created. I use `pushd` and `popd` because it's all scripted, but `cd` might work just as well. The paths and archive structure were a bit of a try, fail and retry experiment for me.

```
pushd packages/serial/ipkbuild/example_package/control/
tar --numeric-owner --group=0 --owner=0 -czf ../control.tar.gz ./*
popd
```

```
pushd packages/serial/ipkbuild/example_package/data
tar --numeric-owner --group=0 --owner=0 -czf ../data.tar.gz ./*
popd

pushd packages/serial/ipkbuild/example_package
tar --numeric-owner --group=0 --owner=0 -cf ../../example_package_1.3.3.7.varam335x.ipk ./debian-binary ./data.tar.gz ./control.tar.gz
popd
```

You will now have an ipk package built:

```
packages/serial/example_package_1.3.3.7_varam335x.ipk
```

gzip vs debian binary package (mime type)

If you have a system that does specific MIME type checks, you might want to use ar to create the package. If you use tar to create a package, the mimetype will be that of a tar or gzip file. Using ar, it will be a Debian Binary package format.

tar packaged package:

```
example_package_1.3.3.7.varam335x.ipk: gzip compressed data, last modified: Thu Apr 4 07:51:34 2019, from Unix (application/gzip)
```

arpackaged package:

```
example_package_1.3.3.7.varam335x.ipk: Debian binary package (format 2.0) (application/vnd.debian.binary-package)
```

To create the package with ar, use the following command:

```
pushd packages/serial/ipkbuild/example_package
ar rv ../../example_package_1.3.3.7.varam335x.ipk debian-binary ./data.tar.gz ./control.tar.gz
popd
```

Output:

```
ar: creating example_package_1.3.3.7.varam335x.ipk
a - ./debian-binary
a - ./data.tar.gz
a - ./control.tar.gz
```

Tags: [arm](#) , [deb](#) , [embedded](#) , [ipk](#) , [openembedded](#) , [openwrt](#) , [opkg](#) , [packages](#) , [tutorials](#) , [variscite](#) , [yocto](#)
