vodafone

Vodafone MachineLink SDK

Software Developer Guide

Trademark Notice

This product may reference NetComm. On December 26, 2024, Lantronix, Inc. acquired the Industrial Internet of Things (IIoT) product portfolio from NetComm Wireless Pty Ltd and is authorized to use the NetComm trademark in association with this product.

Copyright

Note – This document is subject to change without notice.

Revision History

Date	Rev.	Comments
November 2025	A	Several branding updates from NetComm to Lantronix. Updated the 'Signing IPK Packages' section to comply with EN18031 requirements.

NetComm Revision History

DOCUMENT VERSION	AUTHOR	DATE
1.0 – Initial document release	Iwo Mergler	13/01/2011
2.0 – New layout implementation	Adrian Macarthur-King	16/05/2012
3.0 – Revised Edition	Mace Ledingham	12/06/2012
3.2 – Added Custom Kernel warning	Iwo Mergler	29/11/2012
4.0 – Updated for Vodafone MachineLink 3G	John Manson	22/02/2013
4.1 – Correction to Depends field description on p.8	John Manson	16/07/2013
4.2 – Updated for MachineLink 3G Plus, added new sections on Utilities, Inputs and outputs, Serial port, USB port, SMS and TR-069.	John Manson	22/08/2014
4.3 – Updated librdb description.	John Manson	12/09/2014
4.4 – Updated Custom Kernel section.	John Manson	19/01/2015
4.5 – Minor corrections	John Manson	23/03/2015
4.6 – Updated Introduction to add references to the various Vodafone MachineLink product models.	John Manson	12/06/2015

4.7 – Various updates and corrections. Added Example Library, Package installation guidelines, Installable image files (*.cdi) and Display Daemon sections. Updated Custom Kernel, Bad block handling, U-Boot, FIlesystems, Watchdog Timer (WDT), System log, Runtime Database (RDB), USB Port and GPS sections. Corrected Sensory I/O configuration table.	John Manson	01/07/2015
4.8 – Added I/O for MachineLink 4G	John Manson	08/12/2015
4.9 – Added Debugging section	John Manson	17/11/2016
4.10 – Added information about Pre-built Virtualbox virtual machine and Ipkg-cl Utility. Corrections to Custom Kernel section.	John Manson	13/01/2017
4.11 – Added MachineLink 4G Lite model and Best Practice for Flash Memory Usage section.	John Manson	24/04/2018
4.12 – Added source code supply offer	John Manson	23/10/2018
4.13 – Added Signing IPK packages section	John Manson	20/08/2019
4.14 – Updated Package installation guidelines	John Manson	18/12/2019
4.15 – Added the serial port device path for the MachineLink 4G Lite	John Manson	6/7/2020

Table of contents

Using the SDK

Introduction

This document serves as a guide to those wishing to develop or customise software to run on a NetComm NTC product now supported by Lantronix. The SDK itself is a relatively simple wrapper around the NetComm Wireless toolchain. It is intended to simplify the learning curve when compared to the bare crosstoolchain.

As such, it includes the cross-toolchain (binutils, gcc, g++, glibc, etc.), selected sections of the source code, the pre-compiled components of a full system and a set of hierarchical make files.

The SDK includes the kernel source to facilitate the compiling of additional driver modules. Other source code can be supplied on request - most of the system is licensed under BSD, Apache, GPL, LGPL and similar licenses.

With respect to the SDK archive filename and the folders that it creates, the Vodafone MachineLink family of devices are referred to by their product model number. These product model numbers are as follows:

Vodafone MachineLink 3G:	vdf_nwl10
Vodafone MachineLink 3G Plus:	vdf_nwl12
Vodafone MachineLink 4G:	vdf_nwl22
Vodafone MachineLink 4G Lite	vdf_nwl220

The examples in this document use the Vodafone MachineLink 3G package for demonstration. Please take note to modify the commands as necessary for your Vodafone MachineLink product model.

Offer to supply source code

The software included in this product contains copyrighted software that is licensed under the GPL. A copy of that license is included in this document on pages 64 and 68. You may obtain the complete corresponding source code from us by contacting Lantronix via: https://tickets.lantronix.com/.

This offer is valid to anyone in receipt of this information.

Installation

The SDK can be installed in any directory. Enter the following command to unpack the archive:

```
$ tar -xvjf SDK_Bovine_vdf_nwl10.tar.bz2
```

This will create a directory called SDK_Bovine_vdf_nwl10. The full path to this directory is determined automatically in the top-level Makefile and is available as a variable (BASE) to all project Makefiles.

Pre-built VirtualBox virtual machine

To make things easier, the Vodafone MachineLink SDK comes with a pre-built Oracle Virtualbox virtual machine which may be run freely from Windows, Mac OS, Linux or Solaris hosts. Go to www.virtualbox.org to download the latest version of VirtualBox then load the .ova file in the SDK package.

Log in to the virtual machine with the following credentials:

```
Username: SDK
Password: lantronix
```

Dependencies

The SDK should run on any Linux distribution (the SDK was verified on Ubuntu 20), but sometimes additional packages may have to be installed. To aid with this, a dependencies.sh script is provided, which can check for missing host system features. Here is an example:

```
$ ./dependencies.sh
Checking make... OK
Checking bash... OK
Checking sed... OK
Checking grep... OK
Checking patch... OK
Checking diff... OK
Checking fakeroot... OK
Checking Bovine compiler... OK
Compiling test program...OK
```

Note: On 64-bit Linux systems, Bovine compiler requires 32-bit libraries installed.

The script only checks for features that were found missing on some systems in the past, but is by no means complete. Should you encounter a missing feature on your host system that is not detected by the dependencies script, please contact Lantronix so that the script can be updated. You may contact Lantronix via Technical Services Portal https://tickets.lantronix.com/.

Layout

Initial directories

DIRECTORY	DESCRIPTION
binaries/	Pre-compiled binaries, including kernel image, main system, rootfs tree.
compiler/	Cross-compiler Toolchain (arm-cdcs-linux-gnueabi).
dependencies.sh	The dependencies script which checks for missing host system features.
libstage/	Contains the include files and libraries required to build applications for the target.
linux/	Kernel source tree.
Makefile	Top-level make file for examples.
projects/	Example projects, you may add your own here.
tools/	Tools used during the build process.
SDK.pdf	This document.

Generated directories

DIRECTORY	DESCRIPTION
images/	The results of the build process end up here, together with a copy of the main image.
staging_packages/	Staging directory for packages. All projects install their files here. Each project subdirectory is then later transformed into an *.ipk package.

Building

To build all projects, just run 'make' in the BASE directory (in this example it is SDK_Bovine_vdf_nwl10/). It is not necessary to run this as 'root' and also not recommended.

```
$ make
SDK: Building in dir /home/john/SDK_Bovine_vdf_nwl10
mkdir -p /home/john/SDK_Bovine_vdf_nwl10/images
cp /home/john/SDK_Bovine_vdf_nwl10/linux/config_main_falcon
/home/john/SDK_Bovine_vdf_nwl10/linux/.config
cp /home/john/SDK_Bovine_vdf_nwl10/binaries/Module.symvers
/home/john/SDK_Bovine_vdf_nwl10/linux/
make -C /home/john/SDK_Bovine_vdf_nwl10/linux -j8 oldconfig
make[1]: Entering directory
`/home/john/SDK_Bovine_vdf_nwl10/linux'
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
HOSTLD          scripts/kconfig/conf
scripts/kconfig/conf      --oldconfig
Kconfig
...
Packaging /home/john/SDK_Bovine_vdf_nwl10/staging_packages/example_app into IPK (target:
/home/john/SDK_Bovine_vdf_nwl10/images)
Packaged contents of /home/john/SDK_Bovine_vdf_nwl10/staging_packages/example_app into
/home/john/SDK_Bovine_vdf_nwl10/images/signal-strength_1.0_arm.ipk
Packaging /home/john/SDK_Bovine_vdf_nwl10/staging_packages/example_driver into IPK (target:
/home/john/SDK_Bovine_vdf_nwl10/images)
Packaged contents of /home/john/SDK_Bovine_vdf_nwl10/staging_packages/example_driver into
/home/john/SDK_Bovine_vdf_nwl10/images/example-driver_1.0_arm.ipk
```

When this is done, you'll find a *.ipk file for every project in images/. As shipped, these would be images/example-driver_1.0_arm.ipk and images/signal-strength_1.0_arm.ipk.

Examples

The SDK includes two examples, demonstrating each type of project - kernel drivers and userspace applications.

Example Driver

This example project demonstrates how to compile a driver module. It is unlikely that this will be necessary in practice, as the kernel contains drivers for all common hardware. In most situations, it is only a matter of enabling the driver in the kernel configuration.

As part of the build preparations, the kernel is partially configured and built, to the extent needed to compile external driver modules. The kernel configuration used here lives in kernel/config_main_falcon.

In general, the kernel configuration of the kernel running on the board must match the configuration of the one used to compile the drivers.
Should you decide to change this configuration, rebuild the kernel and make sure all the driver modules are recompiled against the kernel tree.

The Makefile in the example_driver project contains the necessary commands to build the driver and install it into the package staging directory:

```
#
# This is a Makefile to compile external kernel drivers for
# the 2.6-series kernels
#
# Expects the ARCH, CROSS_COMPILE and KDIR variables to
be # set correctly.
```

```
#
# The finished driver gets packaged into an ipk

DRIVER := example_driver

obj-m := $(DRIVER).o
```

The obj-m variable sets the name of the driver - NAME.o becomes NAME.ko. In case of a driver with multiple source files (driver1.c and driver2.c), you should set ''obj-m := DRIVER.o'' and then set ''DRIVER-y := driver1.o driver2.o'', where DRIVER is the name of your driver.

```
# Current directory
BDIR := $(shell /bin/pwd)

$(info Building against the 2.6 kernel in $(KDIR) )

.PHONY: all
all:
$(DRIVER).ko
$(DRIVER).ko:
        make -C $(KDIR) SUBDIRS=$(BDIR) modules
```

This rule invokes the kernel build system to build the driver:

```
.PHONY: clean distclean clean distclean:
make -C $(KDIR) SUBDIRS=$(BDIR) clean
rm -f modules.order
```

For cleaning, too, we invoke the kernel build system:

```
# Generic rule for all other targets
%:
        make -C $(KDIR) SUBDIRS=$(BDIR) $@
```

This rule passes any other targets to the kernel build system. It is not strictly necessary, but may help with debugging.

```
.PHONY: install
install: $(DRIVER).ko
        mkdir -p
$(INSTALLDIR)/lib/modules/         cp $
$(INSTALLDIR)/lib/modules/         mkdir
-p $(INSTALLDIR)/CONTROL         cp
control $(INSTALLDIR)/CONTROL/
```

The install rule gets invoked by the projects Makefile. It has the driver module as a prerequisite, so the module is built first.

The INSTALLDIR variable is set to the absolute path of the package staging subdirectory for this project, so we use it to create an IPK structure (more about IPKs). In this case, we simply copy the driver into /lib/modules and the IPK control file into /CONTROL.

Please note that to make a driver package like this useful, there should also be a script in /etc/init.d/rc3.d that loads the module at boot time.

An example driver is shown below:

```
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/module.h>

static int __init example_init(void)
{
        printk(KERN_INFO "Example driver: Hello world!\n");
return 0;
}
static void __exit example_exit(void)
```

```
{
        printk(KERN_INFO "Example driver: Goodbye!\n");
}
module_init(example_ini
t);
module_exit(example_exi
t);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("<Joe Blogs> engineering@wireless.com");
```

This is an example of a very simple driver. On loading, it emits the ''Hello World'' string to the system log, on removal it emits the ''Goodbye'' string. You may test it on the system (after uploading and installing the IPK), by typing ''insmod /lib/modules/example_driver.ko''. To remove the driver, type ''rmmod example_driver''.

The system log output can be seen using ''logread'' on the command line or via the Web interface.

Example Application

The example application (example_app) is a daemon that displays the 3G signal strength on the LEDs, similar to the ''bars'' display on a mobile phone. It uses a less accurate, but fast responding signal parameter to make it feasible for antenna adjustment.

This is meant as an example of how to write a daemon and how to interact with the LEDs and the rest of the system. See the RDB Templates section for a much better way (11 lines of shell script) of achieving the same functionality using RDB templates.

A daemon is an application that, upon launch, forks a child process. The parent process exits while the child process remains running in the background. The required sequence to achieve this is somewhat complex, so we provide a small library (daemon.h / daemon_lib.a) to make this easier.

When it is running, the daemon watches an RDB (see Runtime Databases section) variable 'called service.signal_leds' to determine if its services are required. Only then does it disable the normal LED functions and override them with the current signal strength reading.

Further, the example application includes a web interface extension that allows the user to enable or disable the function. This demonstrates how to add a ''user menu'' and how to use it to control aspects of the system. Please see the Custom Menus section for details on this.

This application also demonstrates the use of a RDB Template to control the daemon via the RDB. The web interface page only sets a RDB variable, which then triggers the template which starts or stops the daemon.

Example Library

An example project to build a library is provided (example_sqlite3). This example compiles and packages the SQLite database.

The purpose of this project is two-fold: First, it demonstrates how to cross-compile Autotools-based open source projects with the SDK and, second, it shows how to handle library dependencies. See the project Makefile on how to do this in detail.

The majority of open-source projects use Autotools (i.e. configure - make - make install). To cross-compile such projects with the SDK, the Autotools build system must know the system types of the target and the build machines.

There are three configure options that are relevant to this:

--host=arm-ntc-linux-gnueabi

--target=arm-ntc-linux-gnueabi

--build=i386

--host specifies the run target of the build. That is, binaries compiled by the SDK will run on the router.

--target specifies the target architecture. This option is usually ignored, but becomes relevant when the compiled program includes a code generator. A classic (but somewhat pointless) example would be a “Canadian Cross” compiler - if configured with --host=arm-ntc-linux-gnueabi -target=x86_64-linux --host=i386, you would build a compiler that runs on the router and produces binaries for an Intel x86-64 architecture.

--build specifies the build machine containing the SDK. This is usually autodetected, but if --target and/or --host options are provided, the autodetection doesn’t always work. i386 is a safe architecture to use for the SDK.

On some rare occasions, one can encounter an autotools-based project that doesn’t have cross-compiling support (see ./configure --help). This can often be fixed quickly by adding the AC_CANONICAL_TARGET macro into the configure.ac file and running autoreconf. This method requires a full installation of Autotools on the build machine.

Library dependencies can be a complex topic. Most non-trivial programs tend to use libraries for part of their functionality. Most non-trivial libraries, in turn, also require external libraries, etc. The list of such libraries are said to comprise the dependencies of a specific project.

Such prerequisite libraries and their include files are needed during build, but only shared library files (*.so) have to be installed into the target.

The SDK build system has support for libraries via the CDCS_INCLUDE and CDCS_LIBS predefined directories. When building a library, the include files and the shared (*.so) and static (*.a) library files should be copied into these directories.

In the same vein, CFLAGS should be set up to contain -I$(CDCS_INCLUDE) and LDFLAGS should contain -L$(CDCS_LIBS) - this instructs the compiler and linker where to find prerequisite libraries. Autotools configure will also use these options to determine if a library is available.

The SDK does not currently support project dependencies - if you are using several separate projects, those projects are compiled in alphabetical order. To ensure that prerequisite libraries are compiled before projects that need them, it is recommended to name the projects so that alphabetical build order is correct.

Example C++ program

This example is a simple “Hello World” program written in C++.

Adding a new project

The projects Makefile (BASE/projects/Makefile) is written such that it automatically treats all subdirectories of BASE/projects as individual projects. In other words, simply creating a new projects subdirectory with a Makefile is sufficient to hook the project into the build system.

A project Makefile must have, at a minimum, rules for 'install' and 'clean' targets.

A number of variables are pre-set by the build system for the use in project Makefiles. Here is a partial list, the full list can be found at the start of the toplevel Makefile:

```
# ARCH            = [arm] Architecture
# CROSS_COMPILE   = [arm-linux-] Cross-toolchain prefix
# PLATFORM        = [Bovine]
# RELEASE         = Base firmware release for this SDK
# BASE            = Base directory of the SDK tree
(This_directory/../) # PBASE           = Projects base
(This_directory)
# IDIR            = Image directory, this is where finished packages go
# PSTAGING        = Packages staging directory
# CDCS_INCLUDE    = CDCS include file directory
# CDCS_LIBS       = CDCS library directory
# KDIR            = Kernel source tree
```

Additionally, project Makefiles are provided with the INSTALLDIR variable, which is set to the package staging directory of the project (PSTAGING/PROJECTNAME).

The absolute minimum valid package staging directory must contain a CONTROL subdirectory which includes the control file. The control file contains things like package name, package version, target architecture and package description. Optionally, the CONTROL subdirectory may contain install scripts. See the Package Control section for details.

The remainder of the package staging directory is a representation of the target's root file system. That is, a file installed into ''INSTALLDIR/etc/settings'' would appear in ''/etc/settings'' on the target, after the package is installed.

Packages IPKs

IPK is a package format originally derived from Debian's DPKG. It is still in wide use on embedded systems, although it is now slowly being replaced by OPKG. At the level we are using IPK in the NetComm Wireless development platform, there are no practical differences between the two.

IPK allows dependency tracking (package requires other packages) and thus recursive installs. Most ipkg tool implementations can synchronise package lists with Internet repositories (package feeds) and fetch prerequisite packages from the same repositories. Due to the support of package version numbers, packages may be upgraded automatically.

On the NetComm Wireless platform, packages may be installed via the web interface 'upload' feature via TR-069, SMS or the command line (ipkgcl).

The system maintains a list of files installed by all packages, such that it can delete them at uninstall time. This can have repercussions if a package replaces existing files. In such a case, the package should rename the original files first, and restore them after the uninstallation.

Ipkg-cl Utility

On the NetComm Wireless platform, Ipkg-cl is used to modify the system IPK installation. This utility accepts sub-commands. Running Ipkg-cl without sub-commands displays the help screen.

There are three frequently used sub-commands.

Ipkg-cl list – Lists available packages and descriptions, for example:

```
root:~# ipkg-cl list
console-logger - 1.0 -
gdb - 7.7.1 -
Successfully terminated.
```

ipkg-cl remove – Removes one or more IPK packages, for example:

```
root:~# ipkg-cl remove gdb
Removing package gdb from
root...
Successfully terminated.
```

ipkg-cl install – Installs an IPK package, for example:

```
root:~# ipkg-cl install /opt/cdcs/upload/gdb_7.7.1_vdf_nwl22w.ipk
Installing gdb (7.7.1) to root...
Configuring console-logger
Using CortexA8 binary
Configuring gdb
Successfully terminated.
```

Package Control

Beyond the ''control'' file we have seen in the examples, the CONTROL directory used during the package staging process can contain optional scripts, which control the installation or uninstallation of the package.

The ''control file'' contains a number of fields. Here is an example:

```
Package: signal-strength
Priority: optional
Section: Misc
Version: 1.0
Architecture: arm
Maintainer: engineering@wireless.com
Depends:
Description: NWL-10 SDK demo application. Displays 3G signal strength via LEDs.
```

FIELD	DESCRIPTION
Package:	Package name, can only contain alphanumeric characters or '-' ([A-Za-z0-9-]). In particular, the underscore character ('_') is used to separate the package name from the version number and is not allowed in the name.
Priority:	This field can be used by the package manager to group packages by importance. Typical entries are ''optional'', ''required'', ''recommended'', etc. Please use ''optional'' for the NetComm Wireless Development Platform.

Section:	Class of application in package, e.g. Base, Net, Utils, etc. For the NetComm Wireless Development Platform, please use Misc.
Version:	Package version. Typically, two ore more numbers separated by '.'. Other formats are admissible, but may confuse the upgrade decision.
Architecture:	For Vodafone MachineLink routers, this is ''arm''.
Maintainer:	Maintainer of the package, ideally an e-mail address. It is best if this is the person who created the package, rather than the application's author.
Depends:	Comma-separated list of package names this one depends on. If any of these dependencies are missing, ipkg will either refuse installing the package or recursively fetch and install the dependencies first. Leave empty, if package can be installed into a clean SDK target.
Description:	Package description, used by package managers. The description may contain more than one line, but all additional lines must start with a space (' ') character.

Beyond the control file, the CONTROL directory may optionally contain a number of executable programs or scripts - ''preinst'', ''postinst'', ''prerm'' and ''postrm''.

preinst If it exists, this file is executed before the installation. It can be used to rename files that would otherwise be overwritten, or perform additional system checks. If a specific minimum firmware release is required on the target, this is the place to check it. A non-0 return value will abort the installation.

postinst If it exists, this file is executed after installation. It is typically used to insert commands into system startup scripts and launch background tasks. The latter is important when creating IPKs that don't require a reboot after installation. It is also possible to request a system reboot here, but if at all possible, this should be avoided.

prerm If it exists, this file is executed before uninstalling the package. It can also refuse uninstallation (return non-0), in case the package is not meant to be uninstalled. Typically used to stop background tasks related to the package, in case they would interfere with the uninstallation.

postrm If it exists, this file is executed after uninstalling the package. It is typically used to restore original files which were replaced during installation, delete files created by programs in the package, remove commands from startup scripts, etc.

Package structure

The IPK package is, despite the *.ipk name, a tar.gz archive. It contains 3 files:

FILE	DESCRIPTION
control.tar.gz	Contains the contents of the CONTROL directory from the staging directory.
data.tar.gz	Contains the remainder of the staging directory (minus CONTROL).
debian-binary	Text file, contains the string ''2.0'' and is usually ignored.

When a package is installed, the control.tar.gz file is extracted into ''/usr/lib/ipkg/info/packagename.*'' and the data.tar.gz file is extracted into the root directory (''/'') on the target.

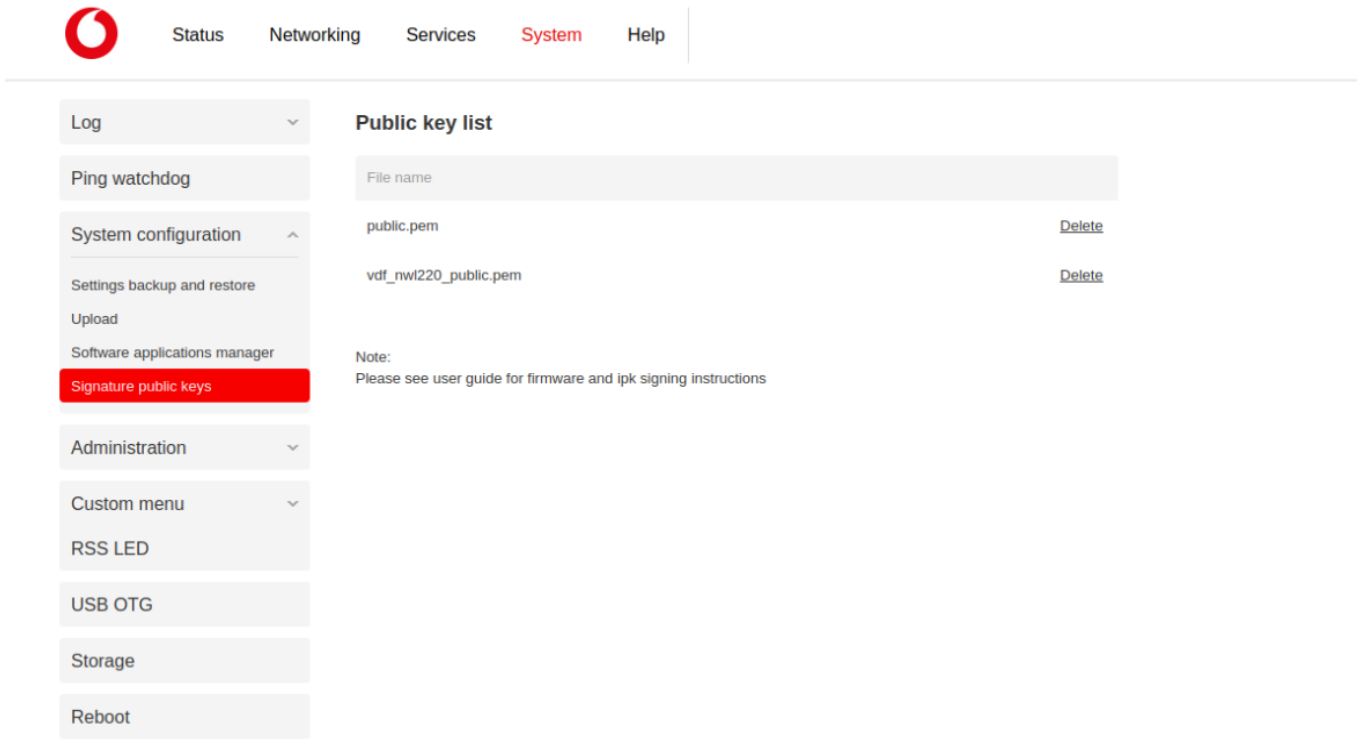
Signing IPK packages

The “example_install_public_key” is a new project added under the “projects” folder, just like “example_app”.

1. If you already have a public/private key pair provided by Lantronix, you may copy the public/private (public.pem and private.pem) key pair into the example_install_public_key package mentioned above. Ensure that the exact name is used.

2. Running “make” from the top level will generate all the IPKs.
3. If you have “example-install-public-key_1.0_arm.ipk” signed by Lantronix previously and have the key pair used to generate the IPK, then you can ignore steps 4-6.
4. The public and private keys (private.pem and public.pem) will be generated by SDK inside the “example_install_public_key” folder inside the “projects” folder.
5. The “example-install-public-key_1.0_arm.ipk” will be created inside the “images” folder. The latest EU RED EN18031 compliance requirement does not allow for disabling the firmware signature check from the web interface. The IPK can be shared with Lantronix for signature via Technical Services Portal https://tickets.lantronix.com/. Lantronix will carry out signing and return signed IPK.
6. The “example-install-public-key_1.0_arm.ipk” can now be used once it is signed by Lantronix. Save the signed “example-install-public-key_1.0_arm.ipk” and the private/public key pair generated by SDK at step 4 which can be reused in the future to be copied in step 1 in a future development cycle.

After installing the IPK, the following is displayed:



For signing firmware or IPKs, the script “sign.sh” should be used, which is located in the “tools” folder under the top level.

Below is an example for signing the “signal-strength_1.0_arm.ipk” which is stored in the SDK’s /images folder:

```
$ ./sign.sh <path-to-private.pem>/private.pem signal-strength_1.0_arm.ipk signal-
strength_1.0_arm Signed.ipk
```

The generated “signed” IPK can now be safely installed on the router.

Package installation guidelines

The root file system of the routers is generally writeable, but does have somewhat limited free space.

If at all possible, packages should install the bulk of their files into the /usr/local file hierarchy. This is a separate partition that usually holds config and log files and provides a fair amount of free space.

To this end, the system includes /usr/local/bin into the binary execution path, and shared libraries are found when installed in /usr/local/lib.

For the future, there are plans to also include /usr/local/init.d/rc3.d/ into the boot script sequence as well as providing for automatic discovery of templates within /usr/local/etc/cdcs/conf/mgr_templates/.

To make packages future proof, it is recommended to install all files into the /usr/local hierarchy and then have the postinst script place symbolic links into the root file system where (still) required.

If a package needs to override an existing binary, the recommended method is to install the new binary, with the same name, into /usr/local/bin. The /usr/local/ hierarchy is first in the binary search path as well as the library search path. That is, as long as it is installed, the new binary in /usr/local/bin/ will override to the one in /bin/ or /usr/bin/, there is no need to make a backup copy.

Start-up scripts can be inserted into the /etc/init.d/rcN.d directories and will get executed in alphabetical order. RDB templates can be dropped into /etc/cdcs/conf/mgr_templates/ and will get picked up automatically on the next boot.

If it is absolutely necessary to backup and replace an existing file, the safe pattern is to have the file with a new name in the IPK, then do the backup and rename in the postinst script. For instance, the IPK would contain and install VPN_ipsec.html.new. Then, the postinst script would "mv VPN_ipsec.html VPN_ipsec.html.nocomplexpasswd.bak; mv VPN_ipsec.html.new VPN_ipsec.html". Note that mv is an atomic operation and a sudden power cut will either leave the old filename or the new one on the filesystem, there is no intermediate state. The postinst script will run again after the reboot and should be written in a way that can handle a situation where the backup mv has run, but then new one has not yet.

Inserting packages into *.cdi images

The installable image files (*.cdi) are, in fact, ISO9660 file system images, containing a number of scripts and image files. In other words, you could rename them to *.iso and write them to a CD, though it wouldn't make much sense.

It is possible to extract the files from the image via a loopback mount, manipulate the file set and re-package them into a new custom image. In particular, IPK packages may be inserted into the images.

During the main system (see System Overview section) image install, any IPKs encountered within the image will be installed as well. That is, you can create a custom image which includes and automatically installs a number of IPKs.

IPKs inside the install image are auto-installed in alphabetical order. If the installation order is important, please prepend letters or numbers ([A-Zaz0-9-]) in front of the package name before inserting it.

A script is provided in the BASE/images directory after a build, which allows manipulation of the image contents. In particular, it allows the insertion or removal of any image or *.ipk file.

For example, here is how you would insert the signal strength demo into the installable image of the NWL-10 (MachineLink 3G):

```
$ sudo ./imagepack.sh vdf_nwl10_X.X.X.cdi -a signal-strength_1.0_arm.ipk
Revision: X.X.X
Bovine, main, trunk, Mon Nov 22 17:42:17 EST 2010
-r--r--r-- 2 root root 11796480 2010-11-01 10:52 root.jffs2
-r-xr-xr-x 2 root root   201828 2010-11-01 10:22 u-
boot.bin -r--r--r-- 2 root root  1613347 2010-11-01
10:32 uImage cdcs_install.sh cdcs_install_m.sh
cdcs_install_r.sh checksum.md5 root.jffs2 scripts/
scripts/utils.sh u-boot.bin uImage vars.sh version.txt
Using CDCS_000.SH;1 for  /cdcs_install_r.sh (cdcs_install_m.sh)
Using CDCS_001.SH;1 for  /cdcs_install_m.sh (cdcs_install.sh)
73.23% done, estimate finish Tue Nov 23 17:36:17 2010
Total translation table size: 0
Total rockridge attributes bytes: 1466
Total directory bytes: 2048
Path table size(bytes): 26
Max brk space used 21000
6835 extents written (13 MB)
```

The script must be run as root (using sudo), because normal users are not usually authorised to create loopback mounts. You may specify more than one [-a filename] options, to insert several files simultaneously.

The full documentation for imagepack.sh is available at the command line, with ''sudo ./imagepack.sh -h''.

After the imagepack.sh run, a new image is created (in this case vdf_nwl10_X.X.X_C.cdi), which, when installed will also install the IPK.

Custom Kernel

You may find that your custom application requires changes to the kernel configuration. This may be because you require a kernel feature which is not currently enabled, or maybe an additional driver.

It is highly recommended that, if possible, changes to the kernel should be configured as modules. That way, you only have to extract the relevant *.ko files from the kernel tree, wrap them into an IPK and install them on the system.

Some configuration options have global scope (cpu, optimisation, debug, etc.). In this case, a config change affects the whole of the kernel, including the driver modules. This is not supported in the SDK, since the binary-only RDB driver won't be automatically rebuilt and may cease to function correctly.

WARNING:

Changing some kernel configuration options may yield a kernel that is incompatible with the driver modules on the running system. Some of these drivers are proprietary, binary only, and will fail with an incompatible kernel.

The current kernel has about 1500 options, with more getting added with every release. Enabling everything would result in a kernel that would vastly exceed the available memory on the NetComm Wireless platform.

The default kernel configuration for each product variant is kept in the kernel tree (BASE/linux), in files like config_main_falcon. After performing an initial SDK build, this file will have been copied to .config and the kernel tree will be configured with those settings.

To change them, under BASE/linux, you need to run ''make menuconfig'' (requires the ncurses library on the host) or ''make xconfig'' (requires the QT3 libraries on the host).

However, since this happens outside the scope of the SDK build system, you must set a few environment variables for this. Thus, the complete command line is:

```
ARCH=arm make menuconfig
```

or

```
ARCH=arm make xconfig
```

Please note that the menuconfig/xconfig changes only affect the .config file, which gets overwritten by the original config_main* file whenever the SDK is rebuilt.

To make kernel configuration changes in the SDK permanent, the .config file should be copied over the original config_main_* file.

```
cp config_main_falcon config_main_falcon_original
cp .config config_main_falcon
```

When you have made the necessary changes, you may compile the kernel and drivers from the top-level directory:

```
make kernel
make
```

The finished kernel uImage will end up in binaries/uImage. This uImage can be inserted into the installable image “<image_name>.cdi”.

In brief, creating a custom kernel involves the following steps:

1. Modify kernel configuration, save and exit menuconfig
2. Copy .config to config_main_xxx (where xxx is the platform name) to ensure the config changes are preserved.
3. Run “make kernel” followed by “make” in the root directory of the SDK. Note that the uImage is only built when “make kernel” is run.
4. In the images/ folder, run “imagepack.sh –a uImage <image_name>.cdi to insert the new kernel into the cdi image.
5. Package any additional driver modules (*.ko) into an IPK, as required. Add the IPK to the image as well.
6. Install the resulting <image_name>_C.cdi image on the device.

System overview

The NetComm Wireless platform is designed for ease of development. As such, it has relatively large reserves of FLASH, SDRAM and computing power. It is also quite forgiving of developer errors and can recover from most failures.

Recovery vs. Main

The NetComm Wireless platform contains two independent systems, each with its own kernel and root file system. These two systems are referred to as ''Main'' and ''Recovery''. It is always possible to use one in order to restore the other.

Both systems have Web interfaces that can be used to manipulate the other, inactive system.

Some FLASH partitions are shared between the two systems, most notably the /opt partition which is provided for user uploaded files and the /usr/local partition, which holds configuration and most of the installed packages.

Installable image files (*.cdi)

The SDK is shipped with the released *.cdi image, which contains a full set of system components, including boot loader, recovery and main systems. However, the installation script within the image file may chose to skip or refuse the installation of certain components.

The installation script compares the releases of the installed recovery and main systems as well as a so-called ‘compat’ version number to decide if and what to install.

The compat version is a revision number maintained in the bootloader (U-Boot). The purpose of the compat version is to control the upgrading of the bootloader itself and the recovery system - the aim here is to only apply necessary upgrades and refuse unsafe ones. Unlike the system release itself, the compat version may only change every few years.

Upgrade rules are as follows:

- The main system is always upgraded or downgraded, unless a downgrade would decrement the major release number (first digit).
- The recovery system and bootloader(s) are upgraded if the new compat version is higher than the old one. They are never downgraded automatically.

For development, individual images (uImage, uImage_r, root.*, root_r.*, etc.) can be flashed without any restrictions.

Boot sequence summary

The bootloader (U-Boot) attempts to load the main system by default. If that fails (e.g. missing or corrupt kernel), it loads the recovery system. If that fails too, the bootloader attempts to fetch a recovery system kernel and rootfs from an external TFTP server - on success, those images are automatically flashed.

By pushing and holding the reset button during the start of U-Boot, it is possible to swap the load order of the main and recovery systems. In other words, push and hold the button for 10 seconds and the system will boot into recovery.

Please see the U-Boot section for details on this.

Database introduction

Most applications running on a NetComm Wireless system are using the Runtime Database (RDB), either directly or indirectly.

In a nutshell, the RDB is an inter-process communication and storage system, using variables. Processes can create, read or write variables. There is support for user-based access control. Processes can also subscribe to a set of variables and get notified if any of these variables has changed. Most of this functionality is also available to scripts and the command line.

Database variables are identified by name strings, usually in field.field.field notation. Names tend to be self explanatory and can be used to group data into records, e.g. ''link.profile.1.auth_type'' and ''link.profile.1.status'' are variables related to link profile 1.

Database variables can contain arbitrary binary data, but most contain human readable strings. In order to communicate variable values with config files and scripts, URL encoding is sometimes used on non human-readable data. This can waste more storage space than a compact binary representation saves. Thus, it is recommended to use readable strings.

Database variables can also contain flags that describe properties. One important property is ''persistent''. If set (at variable creation time), the system automatically maintains a copy in the config file. That is, persistent variables survive a reboot.

More detailed information on the RDB can be found in Runtime Database section.

Launching your application

Usually, when adding a new application to the system, it needs to be launched somehow, possibly triggered by a system event. The best way usually depends on the application, but here are a few typical options.

To get your new application launched at system boot time, you can either insert a line into /etc/inittab, add a system startup script, or create an RDB template script.

/etc/inittab

The first task started by the Kernel is Init. Init's job is to launch other tasks as specified in its configuration file, /etc/inittab.

The Init used on the NetComm Wireless platform supports a number of ways to launch applications - by example:

```
::sysinit:/bin/mount
```

This is a system init line, calling the mount command. All lines with the :sysinit: tag will get run in order, each waiting for its predecessor to complete. While there are still :sysinit: lines to run everything else waits.

```
::respawn:/usr/sbin/telnetd
::once:/etc/init.d/rc.S rc3.d
```

After the :sysinit: lines are done, :respawn: and :once: lines are launched in parallel. The difference is that :respawn: keeps the process running, restarting it if it terminates. :once: will launch the process once and if it terminates, it won't be relaunched.

Another possible tag is :shutdown:, which gets executed, in order, before the system is rebooted or shut down. Please note, however, that it is unwise to rely on a clean shutdown on an embedded system.

Here is a basic example of how to hook your application into inittab. You need to append your new line. However, you must make sure that the procedure can survive a power cut at any time. Thus, this would go into your IPK postinst script:

```
cp /etc/inittab /etc/inittab.new
sync
echo 'null::respawn:/usr/bin/myapp' >>/etc/inittab.new
sync
mv /etc/inittab.new /etc/inittab
```

You may note that we don't create a backup copy. If we did, we could inadvertently kill other packages when we restore it later. To remove the line we added (this goes into the IPK prerm script):

```
grep -v 'null::respawn:/usr/bin/myapp' /etc/inittab >/etc/inittab.new
sync
mv /etc/inittab.new /etc/inittab
```

System start-up scripts

An application may be launched by a system start-up script. In order to launch your custom application, you may place a script into /etc/init.d/rc3.d which will be executed during boot time.

The system startup scripts are organised within the /etc/init.d/rc.d directory:

DIRECTORY	DESCRIPTION
rc.d	This directory contains the actual system startup scripts. However, they are rarely run here. Instead, the rc2.d and rc3.d directories contain symbolic links to the scripts in rc.d.
rc2.d	The symbolic links in this directory are named such that they can be run in alphabetical order. The scripts linked by rc2.d are run in strict order immediately after boot and are used to set up the critical system infrastructure.
rc3.d	The symbolic links in this directory also have a ordered naming scheme, but must not rely on that ordering. Currently, they are executed in order , but future versions of the system may launch them concurrently.

RDB Templates

Templates are scripts that are triggered based on system events - changes to RDB variables. Please see the RDB Templates section for details on templates, this here is only a quick overview.

The templates live in /etc/cdcs/conf/mgr_templates/. They are mostly shell scripts with a grammar extension that allows them to specify a list of RDB variables they are sensitive to. All templates are run once at boot time and then every time any of the sensitive variables are written - even if the values remain the same.

For example, here is a template that runs every time the first connection profile is connected and the WAN IP address is available:

```
 #!/bin/sh
WANIP="?<link.profile.1.iplocal>;"
echo ''$WANIP'' >/tmp/wanip.txt
ftpput some-remote-server.com 3g_ip.txt /tmp/wanip.txt
```

This particular example places any new 3G IP address into a file and uploads that file via FTP to a remote machine - a very primitive way of creating a DynDNS-like system. Of course, to make this reliable the template needs some error handling too.

FLASH memory and boot sequence

During normal development, it should not be necessary to reflash the board from scratch. This section is provided as a reference only, to enable better diagnosis of any issues that may arise.

Best Practice for Flash Memory Usage

NAND flash in Vodafone MachineLink routers typically has an endurance limit of 100,000 erase cycles per block. NAND must be erased one block at a time (128K-256K). It then can be written one page at a time (64 per block).

The flash file system (UBI/UBIFS) tracks the erase cycles for each block and distributes the cycles evenly across all blocks of the partition. As a rule of thumb, about 9 small (<4K) file updates will cost an average of one erase cycle. In case of simple file content (e.g. text files), a small file can extend beyond 8K, due to compression. Larger than that, and the erase cycles per update increase slowly.

As an example, let's assume a small (<8K) text file is updated once every 10 seconds. That means that, roughly, every 1.5 minutes we use up an erase cycle. Now we further assume that the partition is 64MB, or 256 blocks in size. We thus have a total of about 25,000,000 erase cycles over the life time of the device.

For our example, we end up with 25,000,000 * 1.5 / 60 /24 / 356, or about 70 years of endurance.

This assumes that the partition is otherwise unused. If it is filled with static data, the above endurance calculation should be halved. It takes up to twice as many erase cycles for the load balancing if other data must be preserved. Another important factor when using the NAND file systems is that a power cut can happen at any time. The file system will preserve its own integrity in this case but cannot preserve the integrity of file data that was in the process of being written. To avoid corrupted files, the only safe way to modify an existing file is this sequence:

1) write a completely new file into the same directory

2) sync

3) rename (mv) new file to old file.

The mv operation is guaranteed atomic. That is, in the case of a power cut, we'll see either the entire old file or the entire new file, but never parts thereof.

As mentioned above, files should not be written incrementally. Whenever possible, a temporary ramdisk file should be used if incremental writing is unavoidable. The temporary file may be copied and renamed with a single atomic mv call.

Bad block handling

Due to the nature of NAND FLASH memory, it’s unavoidable that the memory may be shipped with a particular small number of ''bad blocks'', depending on manufacturer and type. The manufacturer marks such bad blocks by writing a specific pattern at a number of alternative locations within the block.

This means that the system must check for such bad blocks and avoid touching them. Since the manufacturer's test procedure is significantly more sophisticated than what can be done in software on our board, there is no guarantee that we could detect that block as defective or even reliably reproduce the defective mark, should the block ever be erased.

A related issue arises if a block goes bad during the lifetime of the product. All the file systems used on the NetComm Wireless platform are capable of detecting and handling bad blocks. Due to the ECC algorithms used, blocks tend to deteriorate slowly over time and data is not lost in the event of a new bad block. Newly detected bad blocks will be marked bad permanently.

Knowing that a NAND FLASH device may be shipped with a large number of existing bad blocks, this must be taken into account when partitioning the device. The worst case bad block specification varies between devices, but typical maximum numbers of bad blocks are 20-40, each up to 256K large.

Given the potentially large number of bad blocks in a FLASH device and the possibility that those blocks may form a cluster, it would be wasteful to use a fixed partitioning scheme, since every partition (e.g. 1MB bootloader) would require a large number of spare blocks (e.g. 20 = 1.25MB), just in case all bad blocks happen to fall into that partition.

To avoid such wastage, the partitions are created automatically during board production, such that the specified partition sizes match the good blocks in each partition only. The so generated partition layout is stored in a U-Boot environment variable and passed to the kernel at boot time.

Should the U-Boot environment become corrupted or erased, U-Boot is able to re-calculate a suitable partition table. If new bad blocks have developed since production, the resulting partition table will not match the original exactly and the content of some partitions may need to be recreated.

FLASH partition layout

The NetComm Wireless platform divides the NAND FLASH space into 7 partitions. You may get a list of these partitions on a running system by typing ''cat /proc/mtd'' on a running system:

```
root:# cat /proc/mtd
dev:    size   erasesize name
mtd0: 00100000 00020000 "S1S2EN"
mtd1: 00400000 00020000 "rkern"
mtd2: 00b40000 00020000 "rfs"
mtd3: 00400000 00020000 "kernel"
mtd4: 02000000 00020000 "root"
mtd5: 04c00000 00020000 "usr"
mtd6: 07fc0000 00020000 "opt"
```

DIRECTORY	DESCRIPTION
S1S2EN	First and second stage boot loaders (bootstrap & U-Boot) and the U-Boot environment.
rkern	Recovery system Linux kernel.
rfs	Recovery system root file system.
kernel	Main system Linux kernel.
root	Main system root file system.
usr	Shared /usr/local partition, used for configuration files, statistics, etc. Has about 70MB spare storage for customer use, packages, etc..
opt	Shared /opt partition, used for bulk storage. Has about 120MB spare storage for customer use, shared with the upload area (firmware upgrade, etc.) in the web interface.

ROM-Boot

The processor used for the Vodafone MachineLink routers includes a minimal bootloader contained in on-chip ROM. This bootloader scans the system for memory (NAND, NOR, SRAM, SPI-FLASH, etc) for a valid ARM reset vector, which marks the beginning of all executable code for ARM processors.

This bootloader is very limited and can only access the start of FLASH devices, relying heavily on the device's reset defaults to do so. Since even NAND FLASH devices guarantee that the first sector is non-defective, this always works.

When a valid program is found, the ROM-Boot loads the first few KB of that device into on-chip SRAM and executes it.

If no such program is found (empty flash), ROM-Boot falls back into a very primitive debug mode, using either the serial console port or the USB device controller. In such a case, a full debug cable (serial console + USB device) and associated software must be used to flash the first and second stage bootloaders into FLASH.

Bootstrap

The first stage bootloader (Bootstrap) lives in the first few KB of the first eraseblock in the FLASH memory. It is loaded by ROM-Boot into on-chip SRAM and executed.

Bootstrap contains code to start up the SDRAM controller, program the clock generators and knows how to skip bad blocks in FLASH.

After starting the SDRAM controller, it reads the second stage bootloader (U-Boot) into SDRAM and starts it.

U-Boot

U-Boot is a full-featured boot loader. It knows how to properly handle bad blocks in FLASH, can manage FLASH partitions, understands (some) file systems, includes a scripting language, etc.

U-Boot finalises the system setup and then executes a script that handles the decision making for launching one of the Linux kernels. See the diagram below for a flow diagram of the U-Boot script.

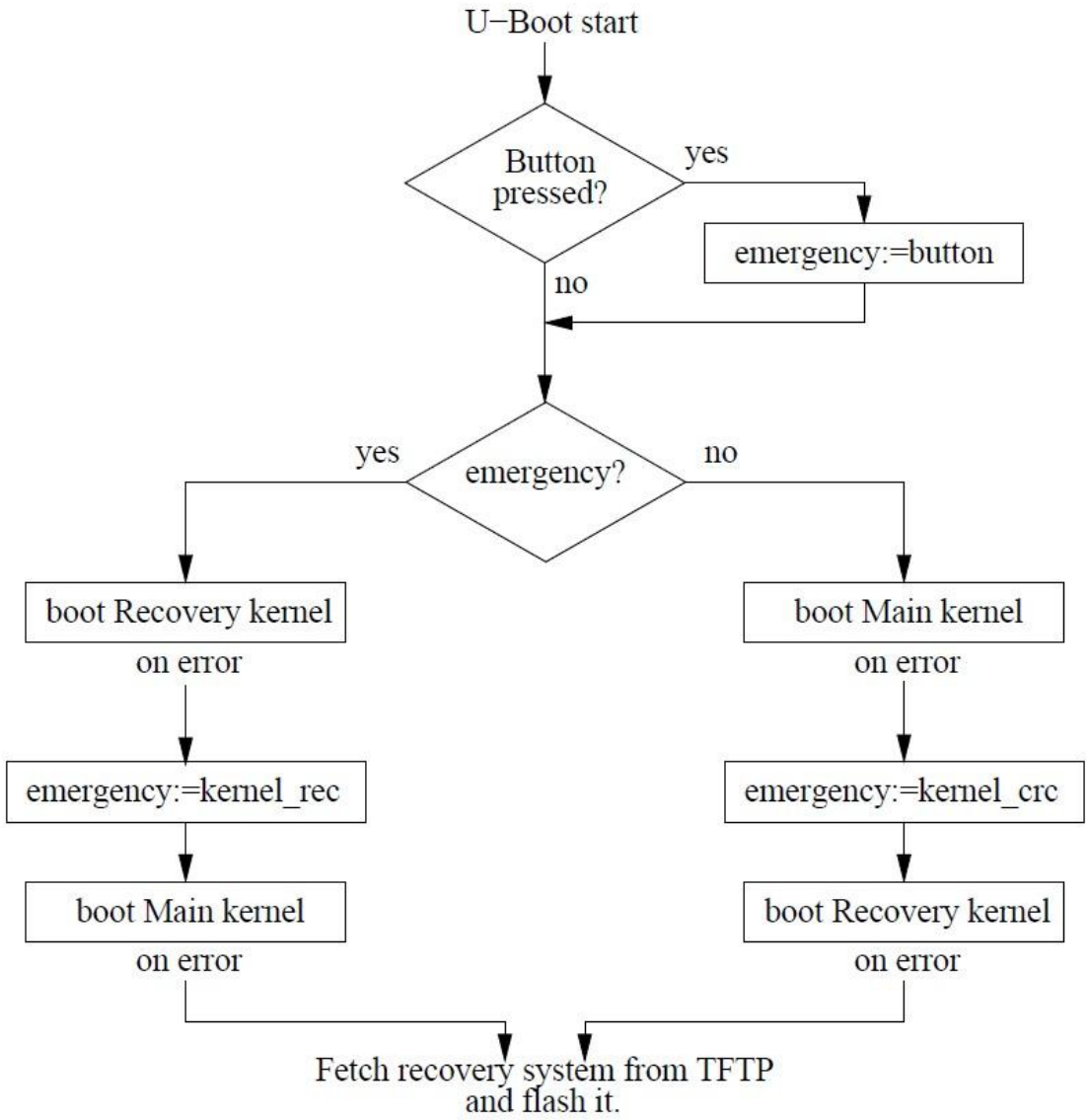


Figure 1: System launch script

Booting a kernel involves determining the kernel command line, which include SDRAM memory size, flash partition layout and root file system to use.

Beyond handling the system startup, U-Boot contains a powerful command line which can be used to reflash parts of the system, manipulate memory locations, talk to the Ethernet PHY, etc.

The U-Boot command line is available on the console serial port, by sending a ‘ ‘ (space) character during the 3-second boot countdown. The console port is separate and different from the serial port on the device’s front panel (where fitted). Typically, the console is a 3.3V TTL serial port and can only be accessed by disassembling the router.

Linux Kernel

The Linux kernel (either Recovery or Main) is launched by U-Boot. It, in turn, uses the information provided by U-Boot to locate and mount the correct root file system.

On the root file system, the kernel starts the first process, called init. A configuration file (/etc/inittab) specifies which startup scripts to run and what other processes to start.

File systems

The file systems used on the NetComm Wireless platform are designed to handle sudden power cuts and bad blocks without loss of data. There are two file systems in use, JFFS2 and UBIFS, depending on the product variant. UBIFS is always used for the two large shared partitions (/opt, /usr/local), while the root file systems are either JFFS2 or UBIFS.

Both file systems are said to use a ''log journal'' - in other words, any modification to the file system is stored as a patch on the previous content. This allows survival in case of sudden power loss - after the reboot, you either get the old content of the file or the new one, but you won't lose it altogether.

Eventually, the partition is filled up with patches upon patches. Both file systems employ background threads that traverse the partitions, consolidating files, erasing obsolete data and so on. Erasing FLASH memory blocks takes most of the writing time, so it is done in the background, proactively.

This behaviour is invisible most of the time, but there are corner cases where it can surprise the user. For instance, should you run a file writing benchmark, the initial results may be too optimistic. The more free space is available on the partition, the more sustained writes it takes before the background task will have an impact.

Also, both file systems employ data compression. A benchmark program that is using predictable data patterns will give surprisingly good results. If you want realistic worst case behaviour use random or already compressed data.

The job of the background task gets harder as the partition approaches its size limit, since it will get more challenging to find blocks consisting entirely of obsolete data. Frequently, the background task has to collect the non-obsolete bits of a number of erase blocks and consolidate them into a single one to free up blocks for erasure. Both the remaining free space and the pattern of past file operations will affect the performance.

It is also recommended to avoid filling a file system with many (thousands) of small files. This skews the data to index ratio - you need more per-file (inode) data structures. On JFFS2, this leads to long mount/boot times and excessive RAM usage. On UBIFS, which keeps the index in the FLASH itself, there is a chance of running out of easily accessible index nodes and the access times increase significantly.

The upshot of all this is that it can be important to select the right partition for the job. We have a selection of file systems and partitions - here are the rules of thumb for use:

PARTITION	DESCRIPTION
rootfs	Separate for Recovery and Main. Use for executables, config files, usually small files that don't change often and won’t get written during normal operation. JFFS2 has the largest difference between best and worst case behaviour. Writing a single, small file is very fast, but hitting the limits can be painful (minutes to write a small file).
/usr/local	Use for files with frequent, small changes. Log files, statistics files, etc. The system uses this partition to maintain the settings and transfer statistics. This is also the default installation location for packages. Try not to exceed half the partition size to maintain fast response times.
/opt	Use for bulk storage. This partition should be used for large files which are written once and not modified often. With this usage pattern, UBIFS tends to show consistent performance over time.

Watchdog Timer (WDT)

The processor includes a hardware watchdog timer which is able to reset the system, in case of software or hardware error. Software has to regularly reset the watchdog timer to keep it running. If it ever reaches the end of the programmed time interval, it issues a hardware reset.

The watchdog timer to times out within 15-100 seconds depending on the product variant.

U-Boot issues watchdog resets within its main loop. That is, while U-Boot is busy running the boot sequence, the WDT is reset periodically. When in command line mode, a watchdog reset is only issued when a key is pressed. A command on the U-Boot command line is provided to disable the watchdog (wdtoff), to avoid a reset after 16 seconds of inactivity.

During a normal boot sequence, the kernel watchdog driver becomes active within a few seconds periodically resets the WDT for about 2 minutes, to allow the userspace to boot.

At boot time, a userspace program is launched (called 'watchdog', part of busybox) which takes over the periodic resetting of the WDT. Typically, at this point the WDT is programmed for a 15 second timeout and the watchdog program attempts to reset every 5 seconds.

Userspace boot sequence

After the kernel (either recovery or main) has mounted its root file system, the system startup scripts are called. These scripts usually live in the /etc/init.d/rc.d/ on the unit (or BASE/binaries/rootfs/etc/init.d/rc.d in the SDK).

However, the scripts in /etc/init.d/rc.d directory are not called directly. Instead, the /etc/init.d/rc2.d and /etc/init.d/rc3.d directories contain symlinks to the files in /etc/init.d/rc.d/. This allows the entries to be ordered - all link names start with Snnn, and scripts can be readily inserted into or removed from the boot process. Here is an example of the /etc/init.d/rc3.d directory:

```
lrwxrwxrwx 1 iwo iwo 15 2010-11-23 10:18 S300urandom -> ../rc.d/urandom*
lrwxrwxrwx 1 iwo iwo 16 2010-11-23 10:18 S301iptables -> ../rc.d/iptables*
lrwxrwxrwx 1 iwo iwo 11 2010-11-23 10:18 S302usb -> ../rc.d/usb*
lrwxrwxrwx 1 iwo iwo 16 2010-11-23 10:18 S306template -> ../rc.d/template*
lrwxrwxrwx 1 iwo iwo 15 2010-11-23 10:18 S999reflash -> ../rc.d/reflash*
```

At boot time, any links to executable files in /etc/init.d/rc2.d are run first, followed by the ones in /etc/init.d/rc3.d. The demarcation line between rc2.d and rc3.d is the local network. At the end of rc2.d, the Ethernet interface, telnet login, web UI are all operational.

Additional application packages may add links to their own initialisation scripts into rc3.d, as required. In general, scripts should allow a service to be brought up (scriptname start) or shut down (scriptname stop). However, the NetComm Wireless system will not shut down services automatically, only start them.

System log

The system log on the NetComm Wireless platform is usually kept in a ring buffer in memory. This buffer can be read with the 'logread' command. Using the -f option to logread will attach to the buffer and continuously dump all new entries.

It may be necessary to temporarily switch to logging to a file, to aid debugging. There is a system setting to log to non-volatile memory, specifically to a file called /opt/messages. On the web interface, the setting is found under System -> Log -> System log settings.

Debugging

Installing gdb and gdbserver on the target router

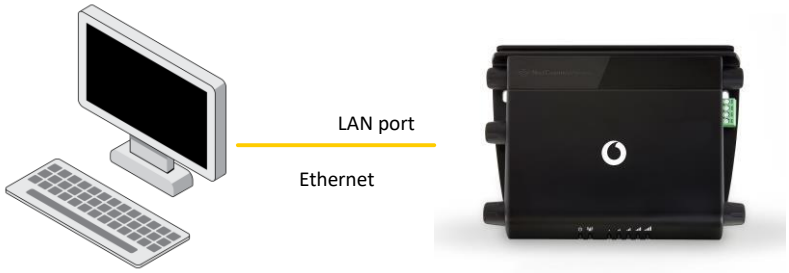
Install the correct package for the model of the MachineLink router you are using:

- gdb_7.7.1_vdf_nwl22w.ipk (for the MachineLink 4G)
- gdb_7.7.1_vdf_nwl12.ipk (for the MachineLink 3G Plus)
- gdb_7.7.1_vdf_nwl10.ipk (for the MachineLink 3G)

gdb and gdbserver will be installed on the target router.

Debugging with gdb and/or gdbserver

Before beginning to debug, ensure that the router is connected directly to the host PC as shown below.



Copy the debugged program to router. Alternatively, on the router, mount NFS to the debugged program's directory on host.

There are several ways to use gdb or gdbserver:

a) On the target router, invoke gdb with the unstripped version of the debugged program and perform debugging.

b) On the target router, invoke gdbserver with the stripped or unstripped version of the debugged program. On the host PC, invoke gdb with the unstripped version of the debugged program and perform debugging.

GDB documentation: https://www.gnu.org/software/gdb/documentation/
Remote debugging: https://sourceware.org/gdb/current/onlinedocs/gdb/Remote-Debugging.html#Remote-Debugging

For example, debugging signal_strength of the example_app:

“example_app” is in <SDK directory>/projects/example_app

The program should be built with debugging information by adding “-g” to CFLAGS.

In <SDK directory>/projects/example_app/Makefile:

```
CFLAGS += -g
```

After building, the unstripped version of signal_strength is in <SDK directory>/projects/example_app while the stripped version of signal_strength is in <SDK directory>/staging_packages/example_app/usr/bin/

On the target router, mount to <SDK directory>:

```
# mount -t nfs -o nolock <host IP address>:<Path-to-SDK-directory-on-host> <mount-point-
ontarget>
```

For example:

```
mount -t nfs -o nolock 192.168.1.166:/projects/sdk /mnt
```

There are several ways to use gdb or gdbserver:

a) On target, invoke gdb with the unstripped version of the debugged program and perform debugging:

```
# cd /mnt/projects/example_app
# gdb ./signal_strength

GNU gdb (GDB) 7.7

Copyright (C) 2014 Free Software Foundation, Inc.
.....
Reading symbols from ./signal_strength...done.

(gdb) break print

Breakpoint 1 at 0x90c0: file signal_strength.c, line 63.

(gdb) list

153    bright="255";

154    } else {

155    bright="0";

156    }

157    set_property(ledlist[i], "brightness", bright);

158    }

159    }

160

161    int main(void)

162    {

(gdb) run

Starting program: /mnt/projects/example_app/signal_strength


Breakpoint 1, print (fmt=0x4001f6d0 "") at signal_strength.c:63

63     {

(gdb) list

58     #endif
```

```
59

60              /* Print function, prints to stdout or system log, as appropriate */
61              static int daemonised=0;

62              int print(const char *fmt, ...)

63              {

64              va_list ap;
65              int rval;

66              va_start(ap, fmt);

67              if (daemonised) {

(gdb) n

67              if (daemonised) {

(gdb) n

63      {

(gdb) n

67              if (daemonised) {

(gdb) n

66              va_start(ap, fmt);

(gdb) f

#0  print (fmt=0xac76 "Starting signal strength daemon (%s, %s)\n") at
signal_strength.c:66
66              va_start(ap, fmt);

(gdb) bt

#0  print (fmt=0xac76 "Starting signal strength daemon (%s, %s)\n") at
signal_strength.c:66
#1  0x00008d90 in main () at signal_strength.c:164

(gdb) print fmt

$1 = 0xac76 "Starting signal strength daemon (%s, %s)\n"

(gdb)
```

b) On the target router, invoke gdbserver with the stripped or unstripped version of the debugged program. On the host PC, invoke gdb with the unstripped version of the debugged program and perform debugging.

On the target router:

```
# cd /mnt/staging_packages/example_app/usr/bin/
```

Invoke gdbserver: gdbserver <Host-IP-address>:<Port> program

parameters For example:

```
# gdbserver 192.168.1.166:6789
./signal_strength
Process ./signal_strength created; pid = 22148

Listening on port 6789
```

On the host PC:

```
$ cd <SDK directory>/projects/example_app
$ ../../compiler/bin/arm-ntc-linux-gnueabi-
gdb
GNU gdb (crosstool-NG crosstool-ng-1.22.0) 7.8

Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

..
..

(gdb) target remote 192.168.1.1:6789

Remote debugging using 192.168.1.1:6789

(gdb) file ./signal_strength

Reading symbols from ./signal_strength...done.

(gdb) list

148            steps=offset/dbled;

149

150            /* Switching LEDS on/off */

151            for (i=0; i<nleds; i++) {

152                    if (i < steps) {

153                            bright="255";

154                    } else {

155                            bright="0";

156                    }

157                    set_property(ledlist[i], "brightness", bright);

(gdb) break print

Breakpoint 1 at 0x90c0: file signal_strength.c, line 63.

(gdb) continue

Continuing.
```

```
Breakpoint 1, print (fmt=0x400206d0 "") at signal_strength.c:63

63     {

(gdb) bt

#0  print (fmt=0x400206d0 "") at signal_strength.c:63

#1  0x00008d90 in main () at signal_strength.c:164

(gdb) n

67             if (daemonised) {

(gdb) n

63     {

(gdb) n

67             if (daemonised) {

(gdb) n
66             va_start(ap, fmt);

(gdb) f

#0  print (fmt=0xac76 "Starting signal strength daemon (%s, %s)\n") at
signal_strength.c:66

66             va_start(ap, fmt);

(gdb) bt

#0  print (fmt=0xac76 "Starting signal strength daemon (%s, %s)\n") at
signal_strength.c:66

#1  0x00008d90 in main () at signal_strength.c:164

(gdb) print fmt

$1 = 0xac76 "Starting signal strength daemon (%s, %s)\n"
```

Runtime Database (RDB)

The Runtime Database (RDB) is the central data repository and inter-process communication system on the NetComm Wireless platform. It allows applications to request or publish information. It also allows applications to synchronise with each other.

Structurally, the RDB is based on a kernel driver, which provides an API via a device node (/dev/cdcs_DD).

On top of that lives rdb_lib, a thin library that abstracts the driver API into functions. This library is linked into all applications that communicate with the RDB.

Database Driver (DD)

Database entries ('variables') are identified by their name. The name is a string containing the characters [0-9a-zA-Z_.)(-] only. By convention, variables are named in a field.field.etc notation. This allows variables to be grouped logically by membership in specific groups or categories. For instance, here are all SNMP related variables:

```
service.snmp.enable 0
service.snmp.name.readwrite private
service.snmp.name.readonly public
```

Variable content can be an arbitrary block of binary data. However, by convention, data is maintained as human readable strings, mostly single words or numbers. Such strings are always stored in RDB including a terminating 0 byte.

Variables are subject to access control permissions, similar to Unix files. There are three permission groups (owner, group & others) and four flags in each group (read, write, erase, perm). By manipulating the permissions, a task can, for instance, create a variable everyone can write to, but only the owner may read (e.g. password check). It is also possible to lock the permissions in place by erasing the perm bit. Since most of the NetComm Wireless system runs as root, this feature is not used much.

Database variables have an additional 'flags' field which denotes special variable properties and behaviour. A few examples:

TRIGGER	DESCRIPTION
PERSIST	Variables with this flag set will be automatically shadowed in the system settings file. That is, they get populated from the settings file at system start and after that, the settings file on disk will mirror the contents.
CRYPT	Variables with this flag set use reversible encryption on their data fields when exported to an external file. This must be used in conjunction with PERSIST, and is set for passwords, PIN numbers, etc.
HASH	Similar to crypt, but uses a one-way cryptographic hash. Rarely used - it can be used to store a password in a nonreversible way. Useful for checking credentials - same passwords result in the same hash.
READ_ONCE	Variable self destructs after one read. Used for temporary keys or other credentials. Avoids inadverted leaking of critical information.
PERSIST	Variables with this flag set will be automatically shadowed in the system settings file. That is, they get populated from the settings file at system start and after that, the settings file on disk will mirror the contents.

Processes can subscribe to a number of variables. After subscribing, the process may sleep (or do something else) and if any of these variables are written, the process will be notified. This is done either via a signal or via a poll/select call.

The database has a global lock, which can be used to group multiple database operations into a single atomic unit. When a process holds the lock, any other process accessing the database will stop until the lock is released. There is a time limit on the lock (a few seconds of CPU time), after which the offending process will be killed.

Library (librdb)

The RDB library abstracts the driver interface for applications. The relevant include file is rdb_ops.h, included in the SDK. The library does not cover the full driver functionality yet and is subject to future extension.

The following functions are supported:

```
int rdb_open(const char *dev, struct rdb_session **s);
```

Opens the database at the device node dev and returns a session reference s that is used by other RDB functions. Returns a negative value on error.

```
void rdb_close(struct rdb_session **s);
```

Closes the database session. All pending subscriptions are dropped. This is also the only way to delete variable subscriptions.

```
int rdb_getnames(struct rdb_session *s, const char *szName, char *pValue, int *pLen, int nFlags);
```

Search function, returns a list of variable names that include the szName

substring. `int rdb_fd(struct rdb_session *s);`

Returns the current file descriptor for direct driver access. Returns a positive value if the database is open, negative otherwise.

```
int rdb_subscribe(struct rdb_session *s, const char *szName);
```

Adds a variable to the task's subscription list.

```
int rdb_getinfo(struct rdb_session *s, const char *szName, int *pLen, int *pFlags, int *pPerm);
```

Returns additional variable information - data length, flags, permissions, owner PID and GID.

```
int rdb_set(struct rdb_session *s, const char *szName, const char *szValue, int len);
```

Write value szValue of length len to database variable szName. The variable must exist.

```
int rdb_set_string(struct rdb_session *s, const char *szName, int *szValue);
```

Writes a value szValue to the RDB variable szName, which should already exist. The string should be 0-terminated.

```
int rdb_get(struct rdb_session *s, const char *szName, char *pValue, int nLen);
```

Read content of a variable. nLen is size of pValue buffer, returns real variable length.

```
int rdb_get_int(struct rdb_session *s, const char *szName, int *pValue);
```

Reads a variable value and converts to integer.

```
int rdb_setflags(struct rdb_session *s, const char *szName, int nFlags);
```

Writes the flags for a single variable in the database.

```
int rdb_getflags(struct rdb_session *s, const char *szName, int *nFlags);
```

Reads the flags for a single variable in the database.

```
int rdb_create(struct rdb_session *s, const char *szName, const char *szValue, int nLen, int
nFlags,
                    int nPerm);
```

Create a new database variable szName and write an initial value szValue to it.

```
int rdb_create_string(struct rdb_session *s, const char *szName, const char *szValue, int nLen,
int nFlags, int nPerm);
```

Create a new database variable szName and write an initial value szValue to it. The value should be a 0-terminated string.

```
int rdb_delete(struct rdb_session *s, const char* szName);
```

Remove variable from database. This is different from a 0-length value.

```
int rdb_update(struct rdb_session *s, const char* szName, const char* szValue, int nLen,
int nFlags,int nPerm);
```

Creates variable if it doesn't exist, writes value. Flags, permissions, etc. are only used during variable creation.

```
int rdb_update_string(struct rdb_session *s, const char* szName, const char* szValue, int nLen,
int nFlags,int nPerm);
```

Creates variable if it doesn't exist, writes value. Flags, permissions, etc. are only used during variable creation. szValue should be a 0-terminated string.

```
int rdb_lock(struct rdb_session *s, int nFlag);
```

Takes the database lock. Must be released within one second.

```
int rdb_unlock(struct rdb_session *s);
```

Releases the database lock.

Command line (rdb_get, rdb_set, rdb_wait)

A command line tool is provided to manipulate the RDB, for use within scripts and on the system's command line. This is a single binary file that may be called using one of the three names: rdb_get, rdb_set and rdb_wait.

You may use rdb_get to fetch a list of variables, or read a single variable. Here we get the list of all variables that contain the string ''snmp'':

```
root:# rdb_get -L snmp
service.snmp.enable 0
service.snmp.name.readwrite private
service.snmp.name.readonly public
```

You can get a full list of all database variables using ''rdb_get -L | sort''.

Creating a new variable (use -p to create a persistent variable - it will end up in the config file):

```
root:# rdb_set testvar test
```

Reading it back:

```
root:# rdb_get testvar
test
```

You can also wait for a variable to be updated:

```
root:# rdb_wait testvar && echo "Got ``rdb_get testvar`'" &
```

This launches a background task waiting for the variable. Now set it:

```
root:# rdb_set testvar 'new value'
root:# Got 'new value'
[2] + Done            rdb_wait testvar && echo "Got \'"$(...)"\'"
```

RDB Manager (rdb_manager)

The system employs a daemon that manages system-wide database related operations. It operates the template system (see the RDB Template section), manages the system settings file and maintains the backing store for statistics.

Those three functions are launched separately during boot time, so you will see ''/usr/bin/rdb_manager -c'', ''/usr/bin/rdb_manager -s'' and ''/usr/bin/rdb_manager -t'' in the process table.

Templates

Templates are a way of triggering an action based on a change in system state, as reflected by the RDB.

In a nutshell, a template is a script. This can be any scripting language or configuration file. A simple grammar extension denotes tags containing RDB variable names which are used to trigger the script.

The grammar to mark RDB variables as triggers for the template is ''?<rdb.variable.name>;''. Several such entries may exist in the template - in which case it becomes sensitive to all. Before the template script is run, these variable placeholders are replaced by the variable values. This way, ''?<link.profile.1.iplocal>;'' becomes ''10.247.49.28''.

A template, like any other script, can also use RDB variables without being sensitive to them. The invocation in that case would be ''`rdb_get link.profile.1.iplocal`'' for shell script templates or luardb.get(“link.profile.1.iplocal”) for Lua templates. This is useful if a template controls a service. In such a case, the template is only sensitive to one variable (e.g. 'service.enabled'), but then uses further variables for configuring the service.

At system start time, rdb_manager parses all available templates in /etc/cdcs/conf/mgr_templates and makes note of the database variable mentioned in each one. After that, each template is run once.

Every time one of these variables changes, rdb_manager creates a copy of the template, replacing those variable tags with the variable content. Then the resulting file is executed.

Templates can set RDB variables and therefore trigger themselves or other templates. It is possible to create loops this way, which will cause high CPU loads. Please be mindful of this possibility.

A template script must terminate quickly. If it needs to start a lengthy process, it may launch it into the background. The template script must then contain code to manage this background process, starting and killing it as necessary.

Here, as an example the simplest template in the system (/etc/cdcs/conf/mgr_templates/reboot.template):

```
#!/bin/sh

REBOOT_ENABLE=?<service.system.reset>;

if [ "$REBOOT_ENABLE" == "1" ]; then
        rdb_set system.reset 0
        (sleep 10; /sbin/reboot;)&
Fi

exit 0
```

The string ''?<service.system.reset>;'' makes this template sensitive to a variable called ''service.system.reset''. This variable is used throughout the system to request a controlled system reboot. You can trigger a system reboot from the command line, using ''rdb_set service.system.reset 1''.

When the variable is set, that line in the template becomes ''REBOOT_ENABLE=1'' in the resulting script. Rdb_manager then runs the script. The script checks the variable value, such that only a value of 1 actually triggers a reboot (you can issue ''rdb_set service.system.reset 0'' all day long).

Finally it launches the reboot command as a background process. The reason for this is to allow the system activity to slow down before the reboot. Thus, a delay is used, in this case 10 seconds. The template is not allowed to go to sleep for that long, so the sleep, together with the reboot command is launched as a background task.

It should be clear by now that the example application (display 3G signal strength on LEDs) in the SDK could be much more simply implemented by replacing the led.template with the following:

```
#!/bin/sh

SIGNAL=?<wwan.0.system_network_status.RSCPs0>;

LEDS="`led info LIST`"
dBm=100
for LED in $LEDS; do
        if [ $SIGNAL -lt $dBm ]; then
                led set $LED trigger none brightness 255
else
                led set $LED trigger none brightness 0
fi
        dBm=$(($dBm-7))
done
```

LEDs

The LEDs are available via the standard sysfs interface. When using them for custom applications, care must be taken to avoid conflicts with existing functionality.

In general, normal operation of the LEDs tends to operate via triggers (see below), which can be switched off. There are a few cases where LED state is also controlled or modified within RDB templates, e.g. RSSI and DCD LEDs.

For most simple applications, it is sufficient to update the LED state about once per second, thus overriding the templates. Beyond that, it may be necessary to disable the relevant templates.

Sysfs LED interface

The Vodafone MachineLink routers have 7 LEDs, which are all accessible via the sysfs interface. Each LED is represented by a directory in /sys/class/leds/. Starting at the red LED, the names are 'power', 'wwan', 'dcd', 'service' and 'rssi', respectively.

Each LED subdirectory contains a few mandatory files and a number of optional files which appear and disappear depending on the selected trigger.

In a nutshell, a trigger is a kernel level signal that can be used to control an LED. These triggers tend to be attached to specific functions, like hard drives. Triggers can be assigned to LEDs, usually any LED can be attached to any trigger.

Available triggers can be listed as follows - they are the same for all LEDs. Not all triggers are used.

```
root:/sys/class/leds/power# cat trigger
[none] nand-disk timer heartbeat cdcs-wwan cdcs-wlanap cdcs-wlanst
cdcs-pots default-on
```

Without a trigger (trigger=none), the led can be switched on or off by writing 255 or 0 to 'brightness'.

```
root:/sys/class/leds/power# echo 255 >brightness
root:/sys/class/leds/power# echo 0 >brightness
```

A trigger can be set by writing the name of the trigger to the 'trigger' file.

```
root:/sys/class/leds/power# echo heartbeat >trigger
root:/sys/class/leds/power# cat trigger
none nand-disk timer [heartbeat] cdcs-wwan cdcs-wlanap cdcs-wlanst
cdcs-pots default-on
```

TRIGGER	DESCRIPTION
none	No trigger selected, LED will only answer to brightness level changes
nand-disk	Triggers on NAND FLASH activity. Currently not enabled.
timer	Pseudo trigger, makes the LED flash. It creates 'delay_on' and 'delay_off' files which control the flashing on and off times (in ms).
heartbeat	Trigger linked to the system load. It creates a double flash (like a heartbeat), the period of which is related to the system's load average. When the system works harder, the heartbeat increases.
cdcs-wwan	This is the WWAN / 3G trigger. It flashes the LED during 3G activity. Like all cdcs-* triggers, the LED operation can be inverted (by setting brightness_on & brightness_off), which allows link/activity functionality.
cdcs-wlanap	WLAN link/activity. Only used on devices with WLAN access point.

cdcs-wlanst	WLAN link/activity. Only used on devices with WLAN access point.
cdcs-pots	POTS/SLIC interface activity. Only used on devices with phone interface.
default-on	Pseudo-trigger, when set switches the LED on. The kernel contains default trigger settings for all LEDs, to be used during boot time. This trigger allows a LED to be solid on during boot.

LED Library (libled)

This library is provided to simplify the LED operations. It accesses the LEDs via the sysfs files as well, but packages these accesses into function calls.

The library allows for LED 'properties' to be read or set. A property is a name+value pair which matches the name of a sysfs file and the current or future value in that file. See libled.h for usage details.

Display Daemon

On some system variants, a daemon controls the LEDs. This daemon (dispd) is present on systems that have to coordinate and prioritise LED functions beyond what can be achieved by stateless templates and LED triggers.

A control script (ctrl_dispd.sh) is provided that can control the dispd - specifically, it must be disabled when controlling the LEDs for other purposes. See below for usage.

LED control via command line ('led')

The first thing to do before manipulating the LEDs manually is to set the LED Display Daemon to User Mode:

```
root:~# ctrl_dispd.sh
user_mode
```

This enables you to take full control of the LEDs without the display daemon interfering with the LED status. To control of the LEDs to the router once more use the following command:

```
root:~# ctrl_dispd.sh
router_mode
```

The 'led' command line program is a simple command line wrapper around libled. It allows writing of cross-platform, portable shell scripts. You can use it to determine the number of available LEDs and the physical order:

```
root:# led info NUM
5 root:# led info LIST
power wwan dcd service rssi
```

The program may also be used to save, restore and manipulate LED state. For example, the following command line gets the power led to flash and then restore the old state:

```
root:# STATE=`led save power`
root:# echo $STATE
power trigger none brightness 255
root:# led set power trigger timer delay_on 100 delay_off
100
```

Now, we restore the old state:

```
root:# led set
$STATE
```

Each LED may display red, green or amber with amber being achieved by setting the LED to display both red and green at the same time. For example, you can set the power LED on the device to red by using this command:

```
root:~# led set power_r trigger none brightness
255
```

The LED can then be set to amber by also setting the power LED to display green:

```
root:~# led set power_g trigger none brightness 255
```

The following script is installed on the system (leds.sh) and is a good example of portable LED control. It creates a LED scanning effect, using the timer trigger as a PWM to 'fade' the LEDs. When terminated, the old LED states and functions are restored.

```
root:# cat /bin/leds.sh
#!/bin/sh
# LED demo
#

led save ALL >/tmp/ledsave

N=`led info NUM`
LIST=`led info LIST`

echo "Leds: $N, $LIST"

pick() { shift $1; echo $1; }

# $1 number, $@ val
ledset() {
        L=$1; shift
        if [ $L -lt 0 ] || [ $L -ge $N ]; then return; fi
        LED=`pick $((L+1)) $LIST`
led set $LED $@
}

exiting() {
         echo "Caught signal, exit"
         if [ -e /tmp/ledsave ];
         then
                 led restore /tmp/ledsave
                 rm -f /tmp/ledsave
         fi
         exit 0
}
trap 'exiting' EXIT
trap 'exiting' INT
trap 'exiting' TERM

X=-2 D=1
while true;
do
        ledset $((X-2)) trigger none brightness 0
        ledset $((X-1)) trigger timer delay_off 10 delay_on 1
        ledset $X trigger none brightness 255
        ledset $((X+1)) trigger timer delay_off 10 delay_on 1
        ledset $((X+2)) trigger none brightness 0
         X=$((X+D))
        if [ $X -ge $((N+2)) ]; then
                D=-1;
        fi
        if [ $X -le -2 ]; then
                D=1;
         fi
done
```

User Interface (Appweb)

The Web-based user interface is currently built around the Appweb server. This server supports server side scripting and is linked with the RDB system, such that server side (ESP) scripts have access to RDB variables.

Web interface related files are located in the /www directory, which acts as the web server root.

System interaction

Most pages use ESP to communicate with the system via RDB variables. Those RDB variables then control various aspects of the system via the RDB templates. Complex settings are validated via Java Script inside the respective pages.

The Web UI page of the example application demonstrates how to use ESP to access an RDB variable:

```
<%if( request['SESSION_ID']!=session["sessionid"] ) redirect('/index.html');%>
```

This makes sure the user is redirected to the log-in page if the user is not logged in yet.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Signal Strength LEDs</title>
<link href="/BovineStyle.css" rel="stylesheet" type="text/css" />
```

The look-and-feel of the page is defined here. Using the style sheet takes care of the page layout.

```
</head>
<!-- Server side script to link this page to an RDB variable -->
<%
        if (request['REQUEST_METHOD'] == "POST")
                tempval = form['sigledEnabled'];
                retval=set_single( 'service.signal.strength.leds.enable='+tempval );
        } else /*GET*/ {
                tempval = get_single( 'service.signal.strength.leds.enable' );
                form['sigledEnabled'] = tempval;
        }
        if(tempval=='1') {
                 checked_on="checked"
                 checked_off=""
        } else {
                checked_on=""
                checked_off="checked"
        }
%>
```

This usually reads an RDB variable called 'service.signal.strength.leds.enable', whose content is used to set up the variables that control the initial state of the radio button (checked_on, checked_off). In the case of a POST (user has hit the 'save' button), the new value is written to the same RDB variable.

```
<body>
<div id="contentWrapper">
<!-- Include the dynamically generated page decorations and menu -->
<% include /menu.html %>
```

This ESP snippet is responsible for creating the header and menu system for the page.

```
<div class="content" >
<h3>User Menu > Signal Strength</h3>
<div class="line" style="padding-left:20px; margin: 8px 20% 16px 0;"></div>
<table width="80%">
    <tr><th>Signal Strength LED Settings</th></tr>
</table>
<form name="SIGLED" method="POST">
<table width="80%">
<tr>
        <td class="head" id="sigledSet">Signal Strength LED display</td>
<td>
                <input type=radio name='sigledEnabled' value=1 @@checked_on>
 Enable   
                <input type=radio name='sigledEnabled' value=0 @@checked_off>
 Disable
        </td>
</tr>
</table>
```

The Enable/Disable radio button uses the ESP variables to select an initial state - only one of the variables will contain the string ''selected''. The rest is basic HTML, using a form submit to save the settings.

```
<div align="center" style="margin-right:20% ">
<p></p>
<input type="submit" value="Save" >   
</div>
</form>
</div>
</div>
<div id="footer">User Name:   <%write(session["user"]);%>
</body>
</html>
```

Custom Menus

The NetComm Wireless platform has a mechanism for easily adding menu items, without having to edit existing files.

To this end, part of the menu bar is generated dynamically from the contents of the /www/usermenu directory. This directory contains simple text files, whose name is used as the menu entry and whose content is the link to invoke.

For example, this command line sequence adds a link that returns the name of the current 3G provider:

```
root:# cd /www/usermenu
root:/www/usermenu# echo "cgi-bin/rdb.cgi?wwan.0.service_provider_name" > "Provider"
/etc/init.d/rc.d/usermenu
```

The last line invokes the dynamic menu script - this happens on every boot, invoking it manually saves a reboot.

Now, the web interface should have a ''User Menu'' entry, with ''provider'' listed on the drop-down.

Utilities

One Shot Timer Utility

The NetComm Wireless software includes a One Shot Timer Utility (one_shot_timer) which effectively provides a programmable countdown timer. When the timer expires it effects a change to an "output" RDB variable. The timer can also be cancelled or reloaded before expiry.

It can optionally provide a feedback "remaining timeout" RDB variable that the caller can poll to calculate how much time is left before the countdown expires. This variable is updated every 100ms.

Usage

The One Shot Timer Utility is invoked from the command line with a minimum of 2 command line arguments: the input (timer value) and the output RDB variables. All timeout values are in multiples of 10 milliseconds.

For example:

```
rdb_set test.timeout 1000
one_shot_timer -t "test.timeout" -o
"test.output"
```

In this example, the One Shot Timer:

1. Reads 1000 from the input (timer value) variable "test.timeout". This is in tens of milliseconds, so the timeout is 10 seconds.
2. Counts down
3. Sets RDB variable "test.output" to a desired value after 10 seconds when the timer expires. Since the value is not given (via the –w command line option), it is set to empty string "" (which is sufficient in triggering templates and the Generalized Event Manager).

Common Use

The following is an example of how the One Shot Timer Utility might commonly be used:

```
rdb_set test.input 1000 # set to 10 seconds
one_shot_timer -t test.input -o test.output -w "expired –d
```

In 10 seconds when the timer expires, the string "expired" will be written to the "test.output" RDB variable, then the application exits. The "-d" option specifies that one_shot_timer runs as a daemon (e.g. the control will be returned to the calling process immediately after one_shot_timer was invoked). It is expected that most commonly the output variable will be a trigger for either a template or a Generalized Event Manager trigger variable, e.g. logic_builder.0.direct_r_trigger. Therefore, one_shot_timer provides a programmable delay capability between inputs and outputs.

Feedback variable

You can create a feedback variable which can be read by other processes to find out the current state of the countdown.

```
rdb_set test input 1000 # set to 10 seconds
one_shot_timer -t test.input -o test.output -w "expired" -d -f test.feedback
```

The test.feedback variable counts down from 1000 to 0 and remains at 0 after the application has exited.

Multiple instances

While it is completely normal to run multiple instances of one_shot_timer to watch different input variables, it's generally not a good idea to run multiple instances of it with the same set of input and output variables.

For example:

```
rdb_set test.input "1000"
```

```
one_shot_timer -t test.input -o test.output -w "expired" -d
one_shot_timer -t test.input -o test.output -w "expired" -d
one_shot_timer -t test.input -o test.output -w "expired" -d
```

This will result in three one_shot_timer processes running and doing exactly the same thing. To guarantee that only one “one_shot_timer” process with the same input is run (and not others are already running when you call the one_shot_timer), do the following:

```
rdb_set test.input 0 # will stop any one shot timer processes watching test.input
rdb_set test.input 1000 # set the desired timeout
one_shot_timer -t test.input -o test.output -w "expired" -d # launch one process
```

Absolute time (-a) option

It is possible, instead of specifying -t and the name of an RDB variable, to specify the absolute time value in tens of milliseconds. For example:

```
one_shot_timer -a 100 -o test.output -w "expired" -d -f test.feedback
```

Short of kill, it is not possible to stop a timer started using this method, neither is it possible to change the expiry timeout once the timer application is running. In other words it is not much different to:

```
sleep(1); rdb set test.outpu "expired"
```

Arbitrary executable (-x) option

It is possible, instead of specifying -o and -w options, to provide a command to be executed when the timer expires. For example:

```
one_shot_timer -t test.input -x "shut_down_power 666" -d -f test.feedback
```

shut_down_power 666 will be executed when/if the timer expires. The new process is always forked by the one_shot_timer before the parent one_shot_timer process exits.

Implementing a periodic timer

It is possible to use the -x option to start another instance of one_shot_timer, thus implementing a periodic, self-restarting timer/process. To do this, create the following shell script (called periodic.sh):

```
#!/bin/sh
one_shot_timer -t test.input -x "./periodic.sh" -d -f test.feedback
# your actions, for example, blink LED, etc
```

test.input should NOT be set in the script but before the script is first invoked, e.g.:

```
rdb_set test.input 1000
./periodic.sh
```

Please note:

- There will always be only one “one_shot_timer” process.
- It can be stopped by writing a 0 to test.input.
- You can vary the period at any time by writing a desired timeout to test.input.
- The optional feedback variable will count down to 0, then become equal to the test.input value again, then count down again and so on, ad infinitum.

Other usage considerations

- Option -d will daemonize the application e.g. the control will return to the caller.
- if -w (Write value) is not given at all, an empty value will be written on expiry to the rdb output variable.
- The name of input and feedback RDB variables cannot be the same.

- The maximum timeout is 24 hours. Therefore, the maximum value for the rdb timeout variable (given in 10's of milliseconds) is 8640000 = (24*60*60*100)
- Each one_shot_timer becomes a process in Linux. No measurements are done in regards to footprint of these processes and no recommendations can be made at this point as to how many simultaneous instances can be run without unduly loading the system. It is however, expected to be very light weight.

System control script

This is a script which is generated at build time and contains the knowledge of how to change the system's operational state. This usually involves GPIOs, but other interfaces may be used, depending on the platform. The sys script supports digital/analogue I/O control commands that are platform independent. These commands are internally converted to GPIO setting commands – NAMUR (input), analogue input, digital input and digital output. Reading external digital input, writing external digital output, reading ADC commands are also centralised as hardware independent commands in this script.

```
root:~# sys

System control script - knows how to power cycle hardware,
configure audio modes, etc. Functions unavailable on platform are
no-ops. This variant is configured for Bovine/ntc_nwl12.

Usage: sys [-u 0|1] [-otg id|h|d] [-m 0|1] [-w 0|1] [-s 0|1] [-c MODE] [-R] [-B] [-S] [-z
0|1|b]

-u 0|1          = USB control, disable/enable USB infrastructure
-otg h|d        = Switch USB OTG mode between host and device
-otg id         = Return state of USB OTG id pin - 0 = host
-m 0|1          = WAN module control
-w 0|1          = Wifi control
-s 0|1          = SLIC control
-c MODE         = CPLD audio routing mode (loop, voice, buf)
-r MODE         = RS-232/422/485 mode (off, loop, rs232, rs422, rs485)
-R              = Outputs reset reason (powerup, button, etc.)
-B              = Outputs bootmode button state (normal, alternate)
-S              = Returns sim present status (0=yes, 1=no)
-z 0|1|b        = ZigBee control (b to start ZigBee
bootloader)
```

For specific examples of how to use the system control script, see the Serial Port and USB sections later in this guide.

Inputs/Outputs

MachineLink 3G and MachineLink 3G Plus

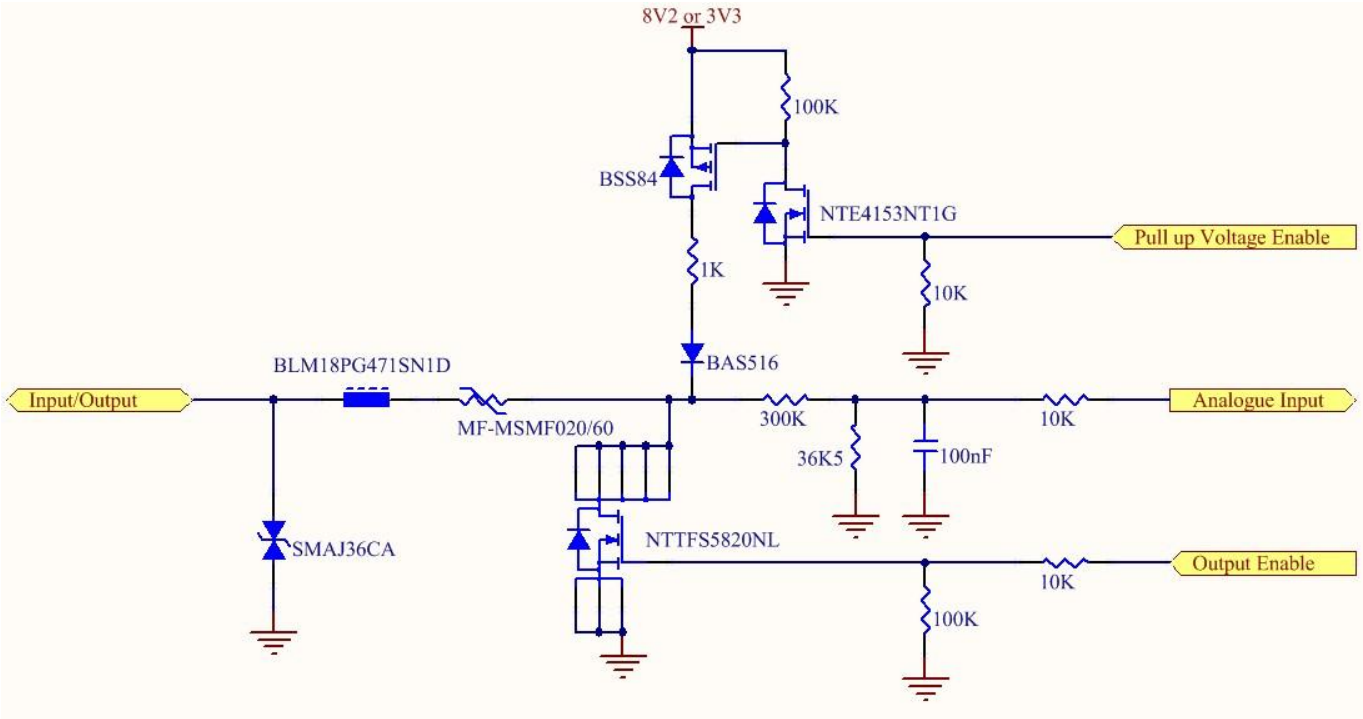
Overview

The Vodafone MachineLink 3G Plus is equipped with a 6-way terminal block connector providing 3 identical multipurpose inputs and outputs as well as a dedicated ignition input. These inputs and outputs may be independently configured for various functions, including:

- NAMUR (EN 60947-5-6 / IEC 60947-5-6) compatible sensor input
- Proximity sensor input for use with contact closure (open/closed) type of sensors (PIR sensors, door/window sensors for security applications) with the input tamper detection possible (four states detected: open, closed, short and break) by the use of external resistors
- Analogue 0V to 30V input
- Digital input (the I/O voltage measured by the Analogue input and the software making a decision about the input state) with the threshold levels configurable in software
- Open collector output.

Hardware Interface

The interface of the 3 multipurpose inputs/outputs are based on the circuit diagram below



The **Input/Output** label is the physical connection to the outside world. There are protection devices and resistor dividers to condition the signal prior to it going into the processor. The three labels to the right are the interface to the processor. **Output Enable** activates the Transistor which provides an open collector (ground) output and can sink 200mA at 23^{o}C. It is protected by a resettable fuse and transient protection diode. If used with the pull up resistor, which can be activated by the **Pull up Voltage Enable** pin, then you can have a High or Low output rather than open drain. The resistor can be pulled up to 3V3 for Cmos compatible output or 8.2V by software. The **Analogue Input** pin can read values from 0V to 30V. It is divided by a resistor network to read appropriate levels in the processor. Depending on the sensor type used, the pull up resistor can be switched on or off. If using the NAMUR sensor configuration the pull up will be activated to 8V2 by default.

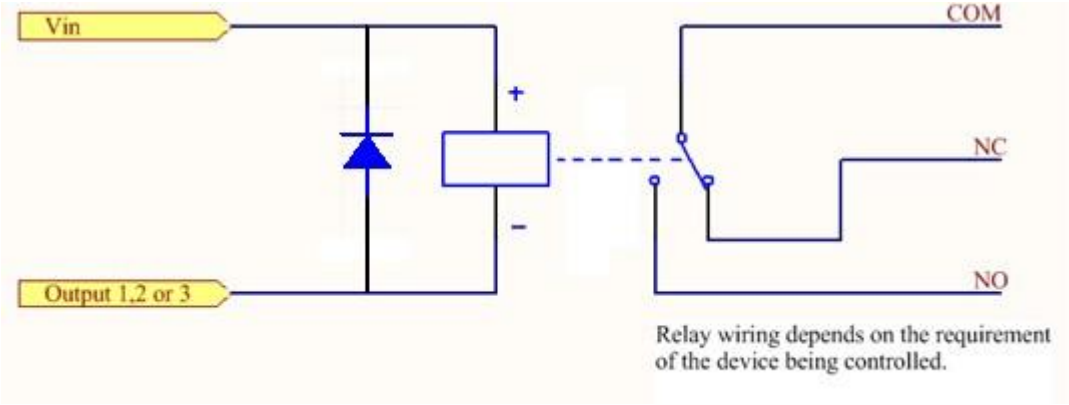
Wiring Examples

The following examples are shown as a guide as to what can be achieved by the I/O features. It is up to the system integrator to have enough knowledge about the interface to be able to achieve the required results.

Note: Lantronix does not offer any further advice on the external wiring requirements or wiring to particular sensors, and will not be responsible for any damage to the unit or any other device used in conjunction with it. Using outputs to control high voltage equipment can be dangerous. The integrator must be a qualified electrician if dealing with mains voltages controlled by this unit.

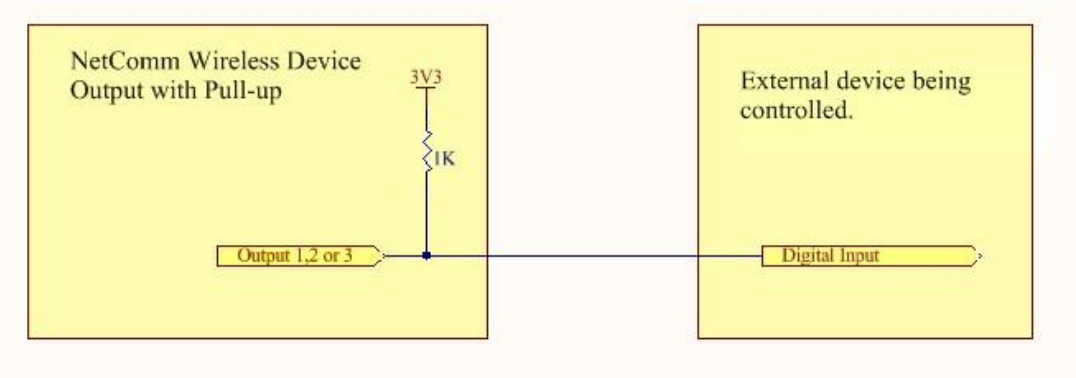
Open Collector Output driving a relay

Any output can be configured to control a relay. This is an example where the transistor will supply the ground terminal of the solenoid. External voltage is supplied to the other side of the solenoid.



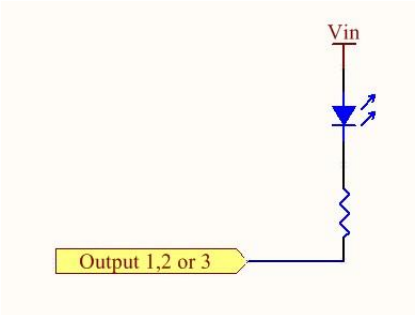
Logic level Output

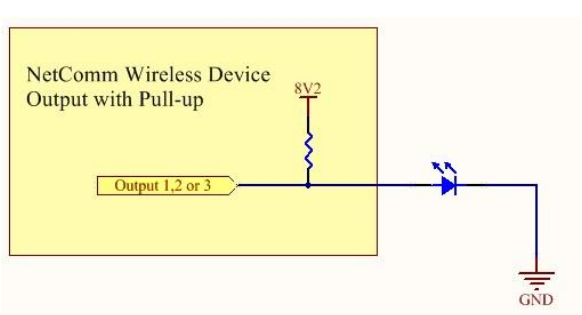
An output can be used with the pull up resistor to provide a logic level output which would be suitable to control an external digital device.



LED Output

An LED can be controlled by simply providing an open collector ground to an externally powered LED Resistor value and Voltage will need to suit the LED type used. Alternatively an LED can be powered using 8V2 via 1K resistor. The suitability of the LED will need to be investigated.

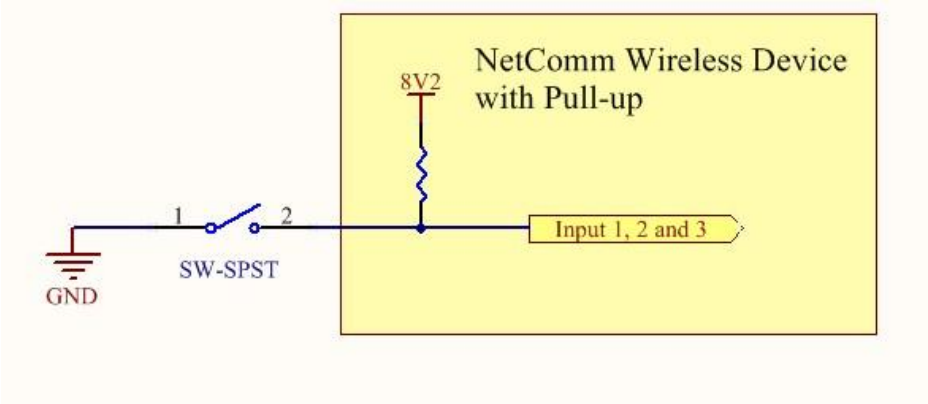




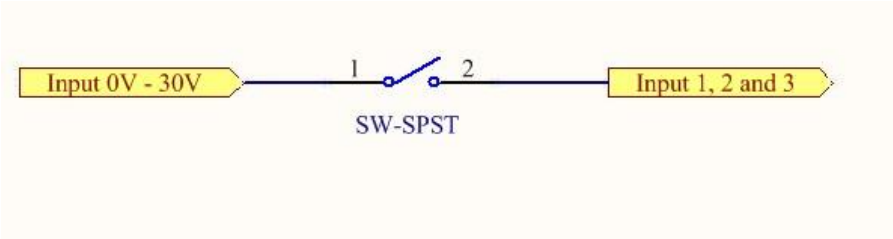
Digital inputs

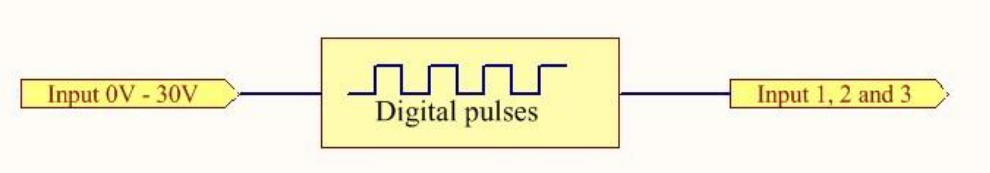
There are several ways to connect a digital input. A digital input can be anything from a simple switch to a digital waveform or pulses. The unit will read the voltage in as an analogue input and the software will decode it in a certain way depending on your configuration.

Below is a contact closure type input, which is detecting an Earth. Pull up is activated for this to work.



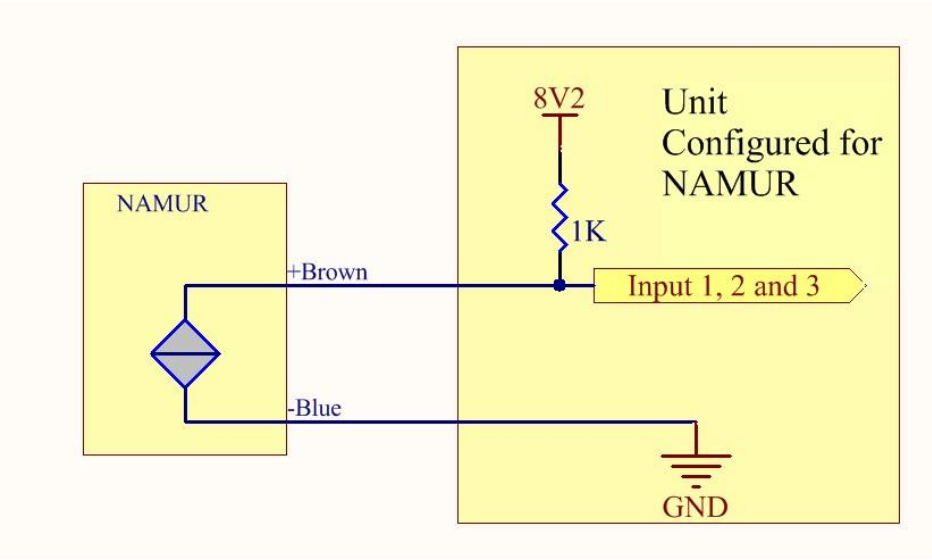
The following input detects an input going high. The turn on/off threshold can be set in the software.





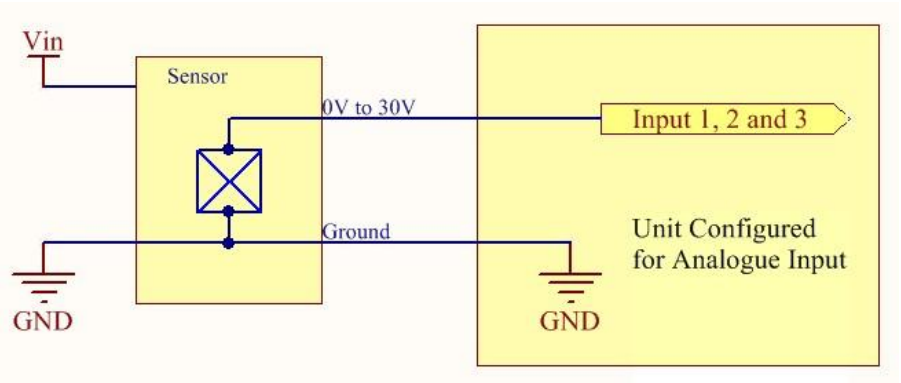
NAMUR Sensor

A NAMUR sensor is a range of sensors which conform to the EN 60947-5-6 / IEC 60947-5-6 standards. They basically have two states which are reflected by the amount of current running through a sense resistor.



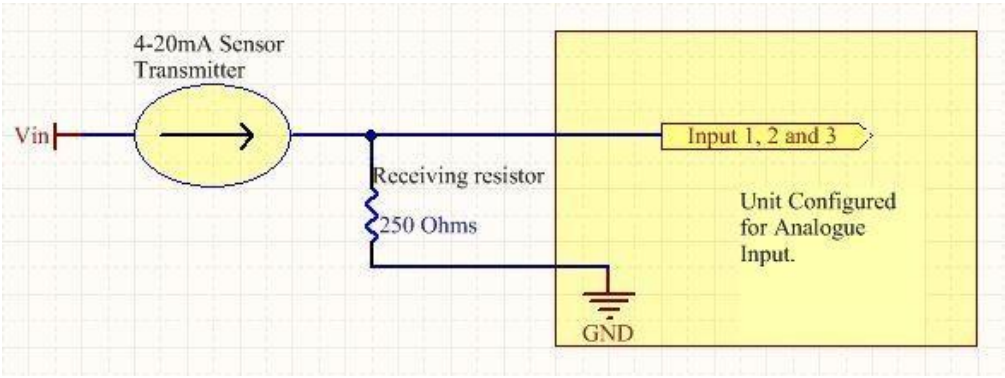
Analogue Sensor with Voltage output

There are various analogue sensors that connect directly to the unit which can provide a voltage output. These would require an external power source which may or may not be the same as the unit itself. The voltage range they provide can be between 0V and 30V. Some common sensor output ranges include 0V to 10V. These would work on the unit, The pull up resistor is not activated in this case.



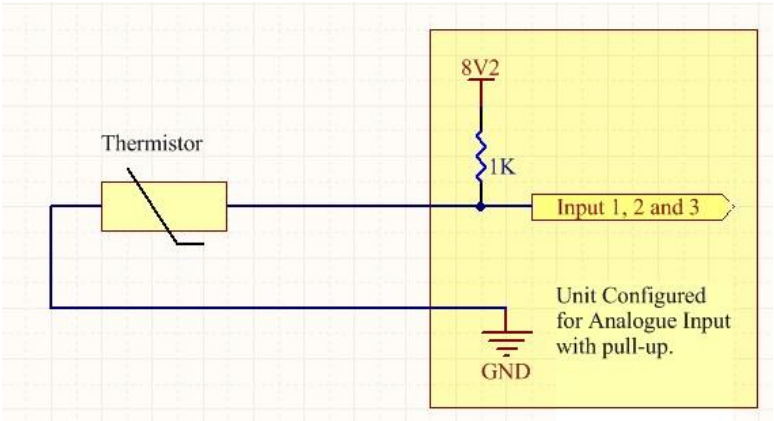
Analogue Sensor with 4 to 20mA output

Another common type of sensor type is the 4-20mA current loop sensor. It provides a known current through a fixed resistor, usually 250 ohms thus producing a voltage of 0v to 5V at the input. The sensor would require an external power source which may or may not be the same as the unit itself. It will also require an external resistor. The internal pull up resistor is not activated.



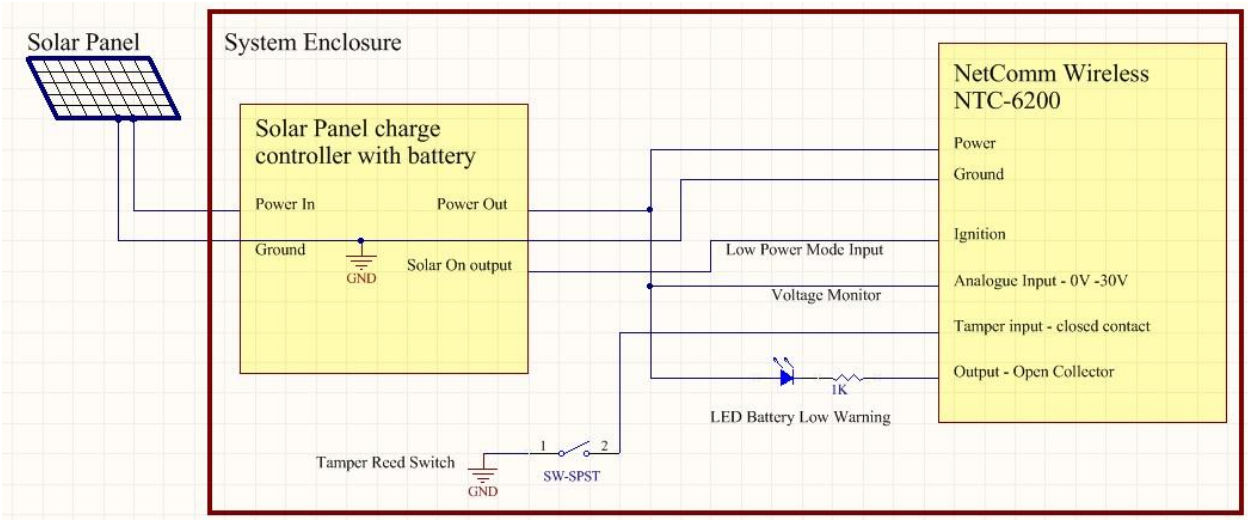
Analogue Sensor with Thermistor

Some sensors work by changing resistance due to a change, such as temperature, light etc. These may be wired up to an external or internal power source and the resistance can be read into the analogue signal. This will require some software calibration like scaling or offset to map the voltage received to the sensor resistor value. An example below shows the internal pull-up voltage and 1K resistor activated. The voltage received depends on the combination of resistors and the value of the resistance of the sensor itself.



System Example –Solar powered Router with battery backup

The previous examples of wiring can be used to come up with a system. The following test case is an example of how the I/O's can be used to enhance a simple router setup.



I/O Design

Kernel Industrial I/O Subsystem

The Lantronix I/O subsystem relies on one of the newest Linux subsystems called the Industrial I/O (IIO) Subsystem because the Linux hwmon driver is more appropriate for fan speed sensors, temperature sensors and voltage sensors. The Linux IIO subsystem also contains a Linux standard application programming interface consisting mostly of I/O control based messages such as Basic device registration and handling, Event chrdevs, Hardware buffer support and Trigger and software buffer support. These standard ways of accessing ADC will allow the I/O daemon to be a generic application that does not depend on a hardware platform. This structure is part of the Linux Kernel and not proprietary to Lantronix.

I/O Daemon

The I/O daemon is a main proprietary structure of the Lantronix IO subsystem. It reads raw ADC data from the standard IIO /sys file system to maintain RDB variables that represent ADC values. Additional configurable pre-processing is performed between raw ADC values and RDB ADC values such as noise filtering, down-sampling, multiplying based on internal/external hardware resistor configuration and error correcting (tuning) per ADC channel. These configurable pre-processing settings exist in RDB, preconfigured in each platform based on V_ADCMAP and changeable in run-time on demand.

Data flow of sensing information

Configuring the I/O pins

There are three methods of accessing the I/O pins; the Sys script, the io_ctl script and the RDB. Each method provides the same set of functionality. The following sections describe how to use each method to configure the I/O pins for your own purposes.

Using the io_ctl script

The io_ctl script performs basic configuration and allows I/O control via the RDB interface. Entering the `io_ctl` command at the command line prompt displays the details of its usage.

```
root:~# io_ctl
```

```
io_ctl controls Aux IO via RDB interface of io_mgr

usage>
        io_ctl <command> [options...]

command syntax>

* Global IO information commands
        stat                            : check to see if daemon is running and reacting
        list                            : list all Ios
         set_pull_up_voltage <3.3|8.2>   : set pull up voltage
         get_pull_up_voltage             : get pull up voltage
         check <IO name>                 : check if the IO exists
         (return code only)
* Per-channel commands

        # Analogue IO commands
        read_analogue <IO name>         : read analogue input
        write_analogue <IO name> <value>        : write analogue output

        # Digital IO commands
        read_digital <IO name>          : read digital Input
        write_digital <IO name> <value> : write digital output

        get_cap <IO name>               : get physical capability of IO
        get_mode <IO name>              : get IO mode
        set_mode <IO name>              : set IO mode

        set_pull_up <IO name> <0|1>     : set pull up status
        get_pull_up <IO name>           : get current status of pull up
        check_pull_up <IO name>         : check to see if pull up control exists

        set_din_threshold <IO name> <value>     : set virtual digital input threshold
        get_din_threshold <IO name>             : get virtual digital input threshold

        set_hardware_gain <IO name> <value>     : set hardware gain
        get_hardware_gain <IO name>             : get hardware gain
* High frequency interface commands

        hfi_enable_output       : enable high frequency output interface (port is 30000)
        hfi_disable_output      : disable high frequency output interface
        hfi_netcat_output       : netcat on the high frequency output interface

        hfi_enable_input                : enable high frequency output interface (port is
30001)
        hfi_disable_input               : disable high frequency output interface
        hfi_netcat_input <input file>   : netcat <input file> to the high frequency input
interface

 * Misc.         help                    : print this help screen

 root:~#
```

Listing the IOs of the device

It's useful to know before beginning any configuration what IOs you can work with. Use the following command to output a list of all configurable IOs of the device:

```
root:~# io_ctl list
3v8poe
ign vin
xaux1
xaux2
```

```
xaux3
root:~#
```

The table below lists each of the I/Os with a description.

I/O	DESCRIPTION
3v8poe	The onboard 3.8V PoE input.
ign	The external ignition input.
vin	The onboard voltage input.
xaux1	The 3 external auxiliary input and output pins.
xaux2	
xaux3	

The I/Os that will be of most interest to many users are the external I/O pins on the power terminal connector of some models of NTC routers. These I/O pins are labelled on the terminal connector as I/O 1, 2 or 3. Through the command line interface, these pins are respectively known as xaux1, xaux2 and xaux3.

Getting the IO capabilities and setting an IO mode

The io_ctl script can be used to return the capabilities of a specified IO, for example:

```
root:~# io_ctl get_cap xaux2
digital_output|analogue_input|virtual_digital_in
put root:~#
```

We now know that auxiliary input 2 can be used for digital output, analogue input and virtual digital input. Let's set auxiliary input 2 to digital output.

```
root:~# io_ctl set_mode xaux2 digital_output
root:~#
```

Setting pull up status

Before setting the pull up status of an IO, check that pull up control can be performed:

```
root:~# io_ctl check_pull_up
xaux2 ok
```

Now turn pull up on for xaux1:

```
root:~# io_ctl set_pull_up xaux2 1
```

Confirm the pull up status:

```
root:~# io_ctl get_pull_up xaux2
1
```

A value of 1 means that pull up is enabled, while a value of 0 means it is disabled.

Setting pull up voltage

The pull up voltage is a global value and may be set at 3.3V or 8.2V. To set the pull up voltage to 8.2V:

```
root:~# io_ctl set_pull_up_voltage 8.2
```

Now confirm that the voltage has been correctly set:

```
root:~# io_ctl get_pull_up_voltage
8.2
```

Writing digital outputs

When an IO is configured as a digital output, you can use the following command to set digital output to 0:

```
root:~# io_ctl write_digital xaux2 0
root:~#
```

Reading digital inputs

When an IO is set to a digital input, you can use the following command to read the digital input:

```
root:~# io_ctl read_digital xaux2
0
```

Setting a digital input threshold

When the device is receiving a digital input, you can set a voltage threshold which is used to determine when the wire is “low” and when it is “high”. Here’s how to set the digital input threshold of auxiliary input 2 to 3V:

```
root:~# io_ctl set_din_threshold xaux2 3.0
```

To confirm the digital input threshold setting:

```
root:~# io_ctl get_din_threshold xaux2
3.0
root:~#
```

Reading analogue inputs

Each of the three IO pins is capable of reading 0V to 10V input. When an IO pin is set to analogue input mode, you must disable pull up on that pin in order to get an accurate analogue input reading. The following command reads the analogue input on auxiliary input 2 and prints a voltage value.

```
root:~# io_ctl read_analogue xaux2
0.16
root:~#
```

Setting hardware gain values

The hardware gain is used to instruct the device to perform division on the voltage which is input to an IO pin. This is because the CPU is expecting 1.8V so some conversion must take place on input voltages.

Hardware gain is turned on by default. To turn it off, set the hardware gain to 1:

```
root:~# io_ctl set_hardware_gain xaux2 1
root:~#
```

High frequency input and output

As the RDB interface is not suitable for extremely high frequency sampling rates (up to maximum hardware specification), the daemon provides two additional TCP ports - one to [extract IO stream] in real-time and the other one to [inject IO stream] back to IO manager. Each TCP stream service allows only a single client to avoid any potential risk of low performance or time sharing issues. Although these services are passive, the extracting feature can easily be redirected to actively send to a certain Internet machine with socat.

Stream extracting service

This extracting service is enabled by default and is easily accessible with netcat or applicable with socat. A single client is allowed at a time.

Stream injecting service

As this stream injecting service overrides all physical inputs, this feature is disabled by default and a single client is allowed at a time. Since this service uses the same format of data that the stream extracting service provides, this service can be useful for many kinds of field test purposes. For instance, for trouble shooting purposes, engineers can apply field data stream to their routers to have a better look to address any application errors.

High frequency interface data format

This information is written in human-readable format. Currently, <IO type> is ain only.

First, enable high frequency output:

```
root:~# io_ctl hfi_enable_output
```

You can then print high frequency output to the device in real-time on the screen:

```
root:~# io_ctl hfi_netcat_output
vin,ain,0:11.90,11.90,11.90,11.90,11.90,11.89,11.88,11.88,11.90,11.88
3v8poe,ain,0:0.01,0.00,0.01,0.01,0.00,0.01,0.00,0.01,0.00,0.01
xaux1,ain,0:8.47,8.47,8.47,8.47,8.48,8.47,8.47,8.47,8.47,8.47
xaux2,ain,0:0.17,0.17,0.17,0.17,0.17,0.17,0.17,0.17,0.17,0.15
xaux3,ain,0:0.15,0.14,0.15,0.14,0.15,0.13,0.14,0.14,0.15,0.13
vin,ain,500:11.87,11.90,11.90,11.89,11.90,11.90,11.90,11.90,11.89,11.
90 3v8poe,ain,500:0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.00,0.01
xaux1,ain,500:8.47,8.47,8.47,8.47,8.47,8.47,8.47,8.47,8.48,8.47
xaux2,ain,500:0.18,0.17,0.17,0.16,0.17,0.17,0.17,0.17,0.16,0.17
xaux3,ain,500:0.14,0.13,0.15,0.14,0.14,0.14,0.15,0.14,0.15,0.14
vin,ain,1000:11.91,11.90,11.90,11.90,11.90,11.90,11.90,11.90,11.88,11
.89 3v8poe,ain,1000:0.00,0.00,0.00,0.00,0.00,0.01,0.01,0.01,0.01,0.00
xaux1,ain,1000:8.47,8.47,8.47,8.47,8.47,8.47,8.47,8.47,8.47,8.46
xaux2,ain,1000:0.17,0.17,0.17,0.17,0.17,0.16,0.17,0.17,0.16,0.17
xaux3,ain,1000:0.15,0.14,0.14,0.14,0.14,0.13,0.14,0.14,0.14,0.13
```

It is also possible to open a TCP socket to access high frequency interface data. This is achieved through the Data stream manager feature available via the web user interface. For further detail on the Data stream manager, please refer to the device's user guide.

RDB Variables

I/O Manager Configuration

RDB VARIABLE	DESCRIPTION	PERMISSIONS	POSSIBLE VALUES
sys.sensors.iocfg.mgr.debug	iomgr debug level	Read and write	[0~7, default:4/LOG_INFO]
sys.sensors.iocfg.mgr.enable	When set to 0, this terminates the IO manager and blocks it from running again. Setting this to 1 removes the block from IO manager running but will not cause IO manager to run. To run IO manager again, you should use the IO manager start up script. WARNING: Disabling the IO manager causes the router to stop monitoring all of the I/O pins including the ignition pin. The router will also cease reporting of the DC input voltage on the Status page. While this RDB variable is used to stop the daemon, it cannot be used to check if IO manager is running. Use the sys.sensors.iocfg.mgr.ready RDB variable to check the status of the IO manager.	Read and write	[1=normal operation | 0=terminate and do not run again]
sys.sensors.iocfg.mgr.ready	When the daemon is ready to accept RDB commands, this flag becomes "1"	Read only	[1=ready, 0=not ready]
sys.sensors.iocfg.mgr.watchdog]	This RDB variable is provided to check the RDB communication with IO manager. As soon as any value is written to this RDB, IO manager resets the RDB back to "1"	Read and write	[only 1]

Sensory I/O Configuration

RDB VARIABLE	DESCRIPTION	PERMISSIONS	POSSIBLE VALUES
sys.sensors.iocfg.pull_up_voltage	Global pull up voltage selection	Read and write	[3.3 | 8.2]
sys.sensors.iocfg.rdb_sampling_freq	Global rdb update frequency	Read and write	[integer number in 1/10th of Hertz, default: 50 (5 Hz)]
sys.sensors.iocfg.sampling_freq	Global ADC sampling frequency	Read and write	[integer number in Hertz, default:10 Hz]

Extra Information

RDB VARIABLE	DESCRIPTION	PERMISSIONS	POSSIBLE VALUES
sys.sensors.info.powersource	Router power source type information - backward compatibility	Read only	[DCJack | PoE | DCJack+PoE]

Example I/O List (this varies depending on the capabilities of the device)

RDB VARIABLE	DESCRIPTION
sys.sensors.io.3v8poe	The onboard 3.8V PoE input.
sys.sensors.io.ign	The external ignition input.
sys.sensors.io.vin	The onboard voltage input.
sys.sensors.io.xaux1 sys.sensors.io.xaux2 sys.sensors.io.xaux3	The 3 external auxiliary input and output pins.

I/O RDB Structure

RDB VARIABLE	DESCRIPTION	PERMISSIONS	POSSIBLEVALUES
OUTPUT			
sys.sensors.io.xaux1.d_out	Digital output	Write only	[0|1]
INPUT			
sys.sensors.io.xaux1.adc	Analogue input	Read only	[real number]
sys.sensors.io.xaux1.adc_hw	ADC register value directly from ADC	Read only	[integer number]
sys.sensors.io.xaux1.adc_raw	Raw analogue voltage. actualy voltage that is asserted to ADC	Read only	[real number]
sys.sensors.io.xaux1.d_in	Digital input. ADC digital input is also updating this	Read only	[0|1]
INFORMATION			

sys.sensors.io.xaux1.cap	Hardware capability of IO pin that the IO pin can be configured for - pipe(|) separated multiple combination of the input or output settings.	Read only	[digital_input | digital_output | analogue_input]
CONFIGURATION – PERSISTENT			
sys.sensors.io.xaux1.mode	Current mode of IO pin	Read and write	[digital_input | virtual_digital_in | digital_output | analogue_input]
sys.sensors.io.xaux1.d_in_threshold	Threshold value for virtual digital input. When [mode] is [virtual_digital_input], [adc] and [d_in] are available. If [adc] is equal and higher than [d_in_threshold], [d_in] becomes 1. Otherwise, [d_in] remains 0. This virtual input mode effectively converts an analogue input only IO pin to a digital input IO pin based on the threshold of [d_in_threshold].	Read and write	[real number]
SCALE CONFIGURATION			
sys.sensors.io.xaux1.scale	Scale value to scale raw value to userfriendly value. pre-configured for each platform or variants	Read and write	[real number, default:preconfigured based on platform]
sys.sensors.io.xaux1.correction	Another scale to correct the value after scaling	Read and write	[real number, default:1]
PER-CHANNEL HARDWARE CONFIGURATION			
sys.sensors.io.xaux1.hardware_gain	Hardware scale configuration	Read and write	[0 = 0~1.8v voltage input, 1 = 0~3.2v voltage input]
sys.sensors.io.xaux1.pull_up_ctl	Hardware pull up control	Read and write	[0=disable pull up, 1=enable pull up]

Starting and stopping the IO manager daemon

While the IO manager automatically starts during the router's boot up sequence, you can manually start or stop the IO manager daemon using the start/stop script. The daemon's process name is "io_mgr".

To start the IO manager daemon:

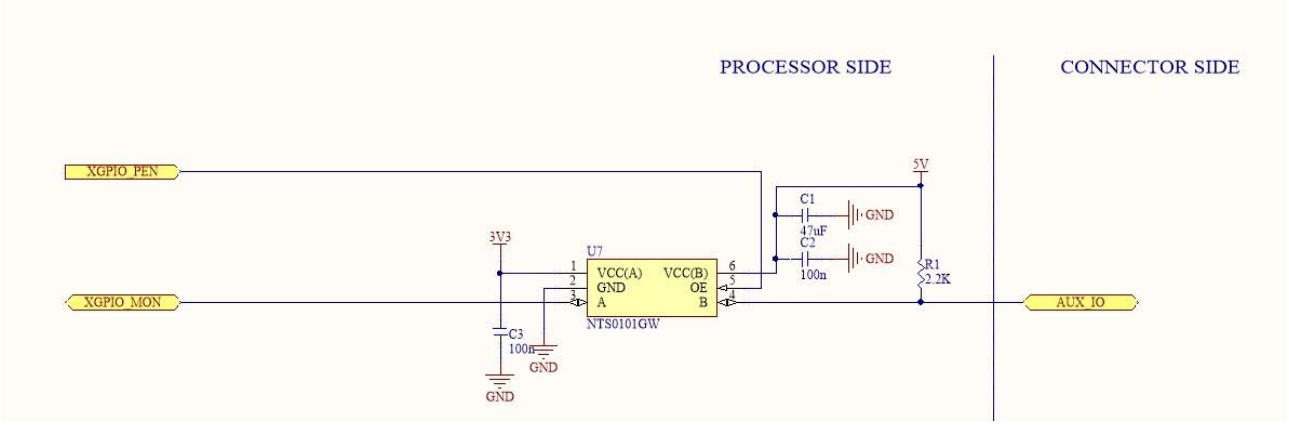
```
/etc/init.d/rc.d/io_mgrd start
```

To stop the IO manager daemon:

```
/etc/init.d/rc.d/io_mgrd stop
```

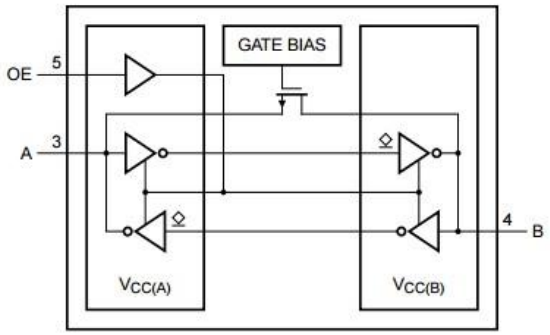
MachineLink 4G

Aux I/O & i-button (1-wire)

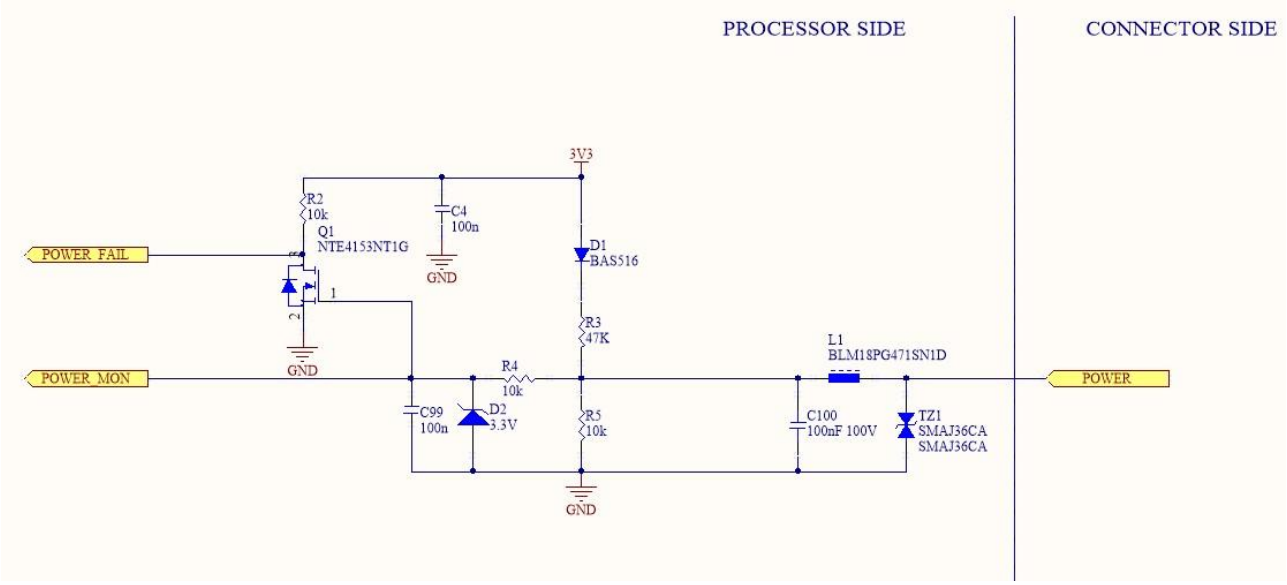


The 1-wire interface allows bi-directional transmission and is activated by XGPIO_PEN within the processor. At the connector side (outside world) it is a 5V tolerant input signal. The device U7 will translate this to processor logic levels of 3.3 volts. The basic operation of the hardware is then interpreted by software (described in the User Guide).

From a hardware point of view the feature revolves around the NTS0101 device which is a dual supply translating transceiver with auto direction sensing. The functional diagram is shown below. More information can be obtained from the datasheet.



Power detection (Ignition) input



The Power detection or Ignition input will detect the presence of three states from the connector side (outside world). It will detect three states, those being High, Low and Unconnected (floating).

The detection uses both analogue and digital inputs to the microprocessor.

- If input voltage at POWER port is less than 0.5V the transistor Q1 is shut, POWER FAIL is asserted high advising micro that no Backup Power is present on POWER port.

  This is the "LOW" state.

- If voltage at POWER port is 0.5-3V POWER FAIL signal de-asserts but POWER MON signal can be measured by ADC of micro and advise the system that there is little or no connection to the unit.

  This is the "Floating" state.

- If voltage at POWER port is higher than 3.3V (which normally is an indication of presence of decent power source), the POWER FAIL is still de-asserted, the POWER MON will measure from 3.3V up to whatever voltage is detected. The maximum detection voltage is 30V and clamps to anything above 36V.

  This is the "HIGH" state.

The states can be adjusted in software to fine tune the transition points between the states.

Serial Port

The SDK allows configuration of the serial port on the Vodafone MachineLink 3G Plus router. Configuration is achieved through the use of the System control script. This section describes the configuration options available.

Disabling modem_emulator

On most products with a serial port, the default configuration has the modem_emulator daemon running which may prevent CLI commands from working. Before you can attempt any manual configuration of the serial port, you should first disable the modem_emulator daemon. The best way of achieving this is to set the "confv250.enable" RDB variable to 0:

```
root:~# rdb_set confv250.enable 0
```

To start it again you can simply set the variable to a value of 1.

```
root:~# rdb_set confv250.enable 1
```

Changing serial port mode (RS232/422/485)

The serial port mode can be configured via the web user interface's Data stream manager. Please refer to the router's user guide for further instructions. Alternatively, the serial port mode can be toggled using the system control script.

The following example shows how to set the serial port to RS-422 mode.

```
root:~# sys -r rs422
sys[-sh ]: (Bovine/ntc_nwl12) serial_mode: rs422
sys[-sh ]: (Bovine/ntc_nwl12) RS-485/422 (full-duplex)
root:~#
```

Accessing and configuring the serial port from your application

On most MachineLink routers, the serial port's device path is /dev/ttyAPP4. On the MachineLink 4G Lite, router, the serial port's device path is /dev/ttyO1. Note that the path uses the letter 'O' and not the number zero.

Example applications

From within your application, you can send data out through the serial port using socat. The following example sends a text file out of the serial port in raw mode with a baud rate of 115000bps.

```
cat /etc/version.txt |socat - /dev/ttyAPP4,raw,b115200,echo=0
```

This example sends GPS data in raw mode through the serial port using socat.

```
gpspipe -r |socat - /dev/ttyAPP4,raw,b115200,echo=0
```

USB Port

Changing USB OTG mode (device/host)

In general, USB OTG mode is fully automated by several kernel components – HCD/UDC controller module, VBUS regulator and USB ID interrupt by the OTG core. The MachineLink 3G Plus router may behave either as a USB device or USB host based on the USB ID pin or OTG negotiation.

As a USB device, routers appear as a multifunction composite device that has a serial port and an Ethernet port. As a USB host, the NTC router can accept major Ethernet devices, USB serial devices and USB storage class devices.

To use the USB functions of the router, USB infrastructure must be enabled. By default it is enabled but you can use the following command to ensure it is enabled:

```
root:~# sys –u 1
root:~#
```

You can then set the USB function to host mode, for example, using the following command. The '1' is the bus number - the external OTG port is bus 1:

```
root:~# sys -otg h 1
root:~#
```

Then check the current state of the USB OTG ID pin:

```
root:~# sys -otg id 1
USB port is in host mode.
root:~#
```

When in host mode, the router supports USB Ethernet, USB Serial and USB storage devices.

USB Ethernet

Multiple USB Ethernet devices can be used and all of the USB Ethernet devices will be bridged to the local interface.

USB Serial

Only a single USB serial external port is supported which is managed through the Data stream manager section of the web user interface.

USB Storage

Multiple USB Storage devices are supported and are automatically mounted and available for SDK users. When connected, USB storage devices are automatically mounted with the following naming convention:

```
/var/mnt/USBDisk[A-Z]
```

The supported file systems are FAT16, FAT32 and NTFS.

SMS

Sending an SMS

Sending an SMS can be done with /usr/bin/sendsms using the following format:

```
sendsms destination_number text store_option
```

For example, to send an SMS containing the message “test sms” to phone number 0412345678:

```
sendsms “0412345678” “test
sms”
```

In the example above, the message to be sent is stored on the SIM card or the memory of the device. To send a message without storing it, use the “DIAG” option:

```
sendsms “0412345678” “test sms” “DIAG”
```

When sending an SMS to an overseas destination, the destination number should begin with the ‘+’ character. The SMS tools application sets the default encoding type to GSM7 and intelligently recognizes when special characters have been entered. If special characters have been entered, the SMS tools application silently changes the encoding type to UCS-2 required to send those special characters. Alternatively, if you wish to enforce an encoding type, change the RDB variable below as required:

```
rdb_set smstools.conf.coding_scheme GSM7
```

OR

```
rdb_set smstools.conf.coding_scheme UCS2
```

If the module supports the IRA character set, the router splits sent messages into multiple messages and sends them with a sequence number which is included in the UDH field. The receiver then concatenates these messages into a single message.

Receiving an SMS

Configuration of the SMS Tools application is available via the web user interface of the router, allowing you to configure forwarding of messages to other mobile numbers, TCP/UDP addresses, email addresses or processing for Diagnostic/Execute commands. Please refer to the SMS Tools section of your router’s user guide for information on configuration via the web user interface.

Accessing received SMS messages

When SMS messages are received by the router, they are stored temporarily in a spool folder (/var/spool/sms/incoming). They are kept there for a short period after which they are processed by the SMS handler script located in /usr/bin/sms_handler.sh. After processing they are stored in /usr/local/cdcs/sms/inbox. If you wish to create extra custom processing of inbound SMS messages, you can perform functions on the messages located in/usr/local/cdcs/sms/inbox. You can change the location that the messages are stored, if so desired, by editing the sms_common.cfg file located in /usr/etc/sms/.

Message format

SMS messages located in the inbox folder use the file naming convention of “rxmsg_XXXXX_unread” if the message has not been read or “rxmsg_XXXXX_read” when it has been read. The ‘X’ characters represent an index number of the message. The first received message will be index number “00000”, the next one will be “00001” and so on. In the case that a message is deleted, the next incoming message will receive the lowest available index number, therefore SMS index numbers do not necessarily indicate a chronological order of when the messages were received.

The following fields are of significance in an incoming message:

FIELD	DESCRIPTION
From	Displays the mobile number of the sender.

From SMSC	The short message service center number.
Sent	The time and date that the message was sent.
UDH	User Data Header. This is an indicator which includes extended information such as totalnumber of messages and sequence number, mostly used for the purpose of concatenating multiple messages.
GSM7 / UCS2	Displays the encoding scheme of the message.

How to disable SMS messaging

To disable the router's SMS service completely, modify the "smstools.enable" RDB variable so that it is 0.

Use the following command:

```
rdb_set smstools.enable 0
```

TR-069

Using TR-069 with your own set of parameters

When using your own custom application, you may wish to use TR-069 to synchronise its settings between a large number of devices. You can achieve this by editing /etc/tr-069.conf. The tr-069.conf file contains the mapping of TR-069 parameters to RDB variables. You may create your own set of parameters and RDB variables for your custom application and list them in the tr-069.conf file so that they can be used over TR-069.

The tr-069.conf file defines the mapping using the following grammar:

```
Definition => Object | Collection | Default |
Param
Object => "object" Name "{" (Definition)* "}" ";"
Collection => "collection" Name Handler "{" (Definition)* "}" ";"
Default => "default" "{" (Definition)* "}" ";"
Param => "param" Name ParamType Notify AccessType Handler ";"

Name => /[_a-zA-Z][-_a-zA-Z0-9]*/
Notify => "notify" "(" Number "," Number "," Number ")"
Handler => HandlerType "(" (ValueList)? ")"

ValueList => Value ("," Value)*

ParamType => "string" | "int" | "uint" | "bool" | "datetime" | "base64"
AccessType => "readonly" | "readwrite" | "writeonly"
HandlerType => "none" | "const" | "transient" | "dynamic" | "rdbobj" | "rdbmem" | "rdb"
Value => String | Number

String => /"[^"]+"/
Number => /-?[0-9]+/
```

For example:

```
param Enable_SNMP bool notify(0,0,2) readwrite rdb("service.snmp.enable", 1, null, null, null,
0);
```

In the example above, the parameter Enable_SNMP is defined as a Boolean. The notify field stipulates how the value is updated, with a default value of 0, a minimum value of 0 and a maximum value of 2. A notify value of “0” means that the value is never updated, “1” means passive notification, i.e. it is updated when the next session is established, while “2” means active notification, where a value update triggers a session to be established. The access type is read and write and the handler name i.e. the designated RDB variable, is “service.snmp.enable”. The following value of 1 indicates that the variable is persistent across reboots. The next four values represents minimum, maximum, validator and default values.

There are several built-in handler types and a generic “dynamic” handler which allows custom logic for special purposes.

HANDLER	DESCRIPTION	ARGUMENTS
none	The default handler, does nothing at all.	none()
const	Constant read-only parameter handler.	const(value)
transient	Memory-only parameters, non-persistent across restarts and reboots.	transient(default)

rdb	RDB parameters, an automatic TR-069 to RDB variable mapping (like persist, but with validation). Persist is a boolean (1 or 0) which turns on the persist flag in RDB. The min and max arguments are context sensitive validation controls, if validator is null the field type is used to determine their meaning. For numeric types min and max are the limits for the value of the field. For string types, min and max validate the length of the field. Passing null for either argument results in no validation of that particular facet. The built-in special purpose validators are current "IPv4", "IPv4Mask" and "URL". For URL the min and max validate the URL length as for strings, for IPv4 and IPv4Mask the min field may be set to zero which enables accepting empty string to indicate no IP specified, if null or otherwise the field must be a valid dotted quad of the specified type.	rdb(rdbKey, persist, min, max, validator, default)
rdbobj	RDB object instance binding (automatic TR-069 to RDB object key convention mapping). Used on collection (multiobject) paths in association with enclosed rdbmem handler parameters. The className is the RDB object class name. Persist is boolean (1 or 0) which controls instance persistence in RDB. The idSelection controls the allocation of object instance IDs, "manual" is not supported. Deletable is also boolean and controls if DeleteObject is allowed to be called on instances.	rdbobj(className, persist, idSelection, deletable)
rdbmem	RDB object member persisted parameters (automatic TR-069 to RDB object key convention mapping). Used on parameters of objects of rdbobj-handled collection (multiobject) paths. The grandparent must be managed by an rdbobj handler. Typically used in the children of the default handler of the rdbobj collection to control the mapping between rdbobject instance members and instance parameters. The memberName is the RDB object instance member name. The min, max, validator and default are as for "rdb" managed parameters - validation and the default value. Unlike rdb the rdbmem parameters inherit their persistence from the configuration of their rdbobj grandparent.	rdbmem(memberName, min, max, validator, default)
dynamic	Manually-coded handler in lua. Implementations in /usr/lib/tr-069/handlers directory as "<handler>.lua". Can have optional extra arguments.	dynamic(handler, default, ...)

GPS

When a GPS unit is first turned on, it needs time to find orbit and clock data for the relevant satellites in order to get a precise fix on its location. The time it takes to retrieve this information is referred to as the Time To First Fix (TTFF). The TTFF value can be further broken down into cold, warm or hot times, the difference between these being the amount of information the GPS begins with when it is turned on. The time it takes to get a fix on its location is affected by factors such as interference and horizon information. Open areas are faster than areas such as canyons or urban areas where tall buildings can interfere with the line-of-sight to the GPS receiver.

To improve the time to a precise fix, some modules include support for Assisted GPS. There are two types of assistance:

- Mobile Station Assisted (MSA) – The router receives some assistance from the A-GPS server, such as rough time. It then samples a snapshot of the satellite signal and uploads it to the server, which calculates the position from the sample and returns the information to the router.
- Mobile Station Based (MSB) – The router acquires satellite signal more quickly and calculates its own position with information (Time, Almanac, Ephemeris, etc.) provided by an A-GPS server.

Sending GPS coordinates to a TCP server

The GPS data can be sent out through the serial port as previously described, but if you would prefer to send GPS coordinates instead of NMEA (raw) data to a TCP server, you could use a simple script to provide GPS details and the IMEI number of the device each time the mobile broadband connection is established. This example requires that the router's module is equipped with a standalone GPS receiver.

The RDB variables containing the values of the coordinates are:

```
sensors.gps.0.standalone.latitude
sensors.gps.0.standalone.latitude_direction
sensors.gps.0.standalone.longitude
sensors.gps.0.standalone.longitude_direction
```

Retrieve the values using the rdb_get command:

```
root:~# rdb_get sensors.gps.0.standalone.latitude
3348.429011
root:~# rdb_get
sensors.gps.0.standalone.latitude_direction
S
root:~# rdb_get sensors.gps.0.standalone.longitude
15108.933068
root:~# rdb_get
sensors.gps.0.standalone.longitude_direction
E
root:~#
```

It is important to note the format of the latitude and longitude values. Latitude is DDMM.MMMM while longitude is DDDMM.MMMM, where D represents degree digits and M represents minute digits. The latitude direction and longitude direction values specify N/S & E/W. If an application requires the latitude and longitude values in a different format than what is presented here, conversion of the coordinates into a more mathfriendly format (e.g. +/-DD.DDDDD) must be done either as a Lua script or written in C.

We can send the coordinates and the IMEI number to a TCP server running on 192.168.1.49:1516 with the following command:

```
root:~# echo "My gps coordinates are `rdb_get sensors.gps.0.standalone.latitude`:`rdb_get
sensors.gps.0.standalone.longitude` Unit IMEI:`rdb_get wwan.0.imei` `date`" | socat
TCP:192.168.1.49:1516 –
```

List of GPS-related RDB variables

```
<< GPS function control
>> sensors.gps.0.enable
<< common data >>
sensors.gps.0.common.altitude
sensors.gps.0.common.date
sensors.gps.0.common.height_of_geoid
sensors.gps.0.common.latitude
sensors.gps.0.common.latitude_directi
on sensors.gps.0.common.longitude
sensors.gps.0.common.longitude_direct
ion sensors.gps.0.common.time
<< standalone GPS data >>
sensors.gps.0.standalone.3d_fix
sensors.gps.0.standalone.GPGGA
sensors.gps.0.standalone.GPGSA
sensors.gps.0.standalone.GPGSV
sensors.gps.0.standalone.GPRMC
sensors.gps.0.standalone.GPVTG
sensors.gps.0.standalone.altitude
sensors.gps.0.standalone.auto_selection
sensors.gps.0.standalone.date
sensors.gps.0.standalone.fix_quality
sensors.gps.0.standalone.ground_speed_knots
sensors.gps.0.standalone.ground_speed_kph
sensors.gps.0.standalone.hdop
sensors.gps.0.standalone.height_of_geoid
sensors.gps.0.standalone.latitude
sensors.gps.0.standalone.latitude_direction
sensors.gps.0.standalone.longitude
sensors.gps.0.standalone.longitude_direction
sensors.gps.0.standalone.magnetic_track_made_good
sensors.gps.0.standalone.magnetic_variation
sensors.gps.0.standalone.magnetic_variation_direction
sensors.gps.0.standalone.mode
sensors.gps.0.standalone.number_of_satellites
sensors.gps.0.standalone.number_of_satellites_tracked
sensors.gps.0.standalone.number_of_sentences
sensors.gps.0.standalone.pdop
sensors.gps.0.standalone.satellitedata
sensors.gps.0.standalone.sentence_number
sensors.gps.0.standalone.status
sensors.gps.0.standalone.time
sensors.gps.0.standalone.track_angle
sensors.gps.0.standalone.true_track_made_good
sensors.gps.0.standalone.valid
sensors.gps.0.standalone.vdop

<< assisted GPS data >>
sensors.gps.0.assisted.3d_fix
sensors.gps.0.assisted.HEPE
sensors.gps.0.assisted.LocUncA
sensors.gps.0.assisted.LocUncAngle
sensors.gps.0.assisted.LocUncP
sensors.gps.0.assisted.date
sensors.gps.0.assisted.height_of_geoid
sensors.gps.0.assisted.lastvalidtime
sensors.gps.0.assisted.latitude
sensors.gps.0.assisted.latitude_direction
sensors.gps.0.assisted.longitude
sensors.gps.0.assisted.longitude_direction
sensors.gps.0.assisted.time
```

```
sensors.gps.0.assisted.updateinterval
sensors.gps.0.assisted.valid

<< GPS One variables >>
sensors.gps.0.gpsone.auto_update.max_retry_count
sensors.gps.0.gpsone.auto_update.retried
sensors.gps.0.gpsone.auto_update.retry_delay
sensors.gps.0.gpsone.auto_update.update_period
sensors.gps.0.gpsone.cap
sensors.gps.0.gpsone.enable
sensors.gps.0.gpsone.update_now
sensors.gps.0.gpsone.updated
sensors.gps.0.gpsone.urls
sensors.gps.0.gpsone.xtra.info.gnss_time
sensors.gps.0.gpsone.xtra.info.valid_time
sensors.gps.0.gpsone.xtra.updated_time

<< odometer variables >>
sensors.gps.0.odometer_control
sensors.gps.0.odometer_enable
sensors.gps.0.odometer_interval
sensors.gps.0.odometer_reset
sensors.gps.0.odometer_threshold
```

GNU General Public License Version 2

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim

copies of this license document, but changing it is not

allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of
Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further

restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and an idea of what it does.
>
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it
>
> and/or modify it under the terms of the GNU General

Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your
option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public
License along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'. This is free software, and you are
welcome to redistribute it under certain conditions;
type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all
copyright interest in the program
`Gnomovision' (which makes passes at
compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

GNU General Public License Version 3

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they

  are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License.
Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's “contributor version”.

A contributor's “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify

or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the nonexercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.

This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <https://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <https://www.gnu.org/licenses/why-not-lgpl.html>.