Opengear custom development kit (CDK) user guide

01. Introduction

This guide explains how to use the CDK to create custom binaries including applications, kernel modules and custom firmware images for your Opengear unit.

The CDK is a hybrid source/binary distribution and build environment, allowing for relatively easy binary-level customization.

Building a custom image involves:

01. Compiling source (kernel, modules, applications).
02. Installing to the romfs directory (modules->romfs, applications->romfs).
03. Assembling the image (compressing romfs into a squashfs file system, and appending the kernel to the end of the image).

Alternatively, source code is available as board support packages (BSPs) allowing you to build fully open source-based firmware, excluding Opengear proprietary components such as the UI. This is not covered by the *CDK user guide*.

For more information visit:

https://opengear.zendesk.com/entries/23111108-What-does-the-CDK-offer-
https://opengear.zendesk.com/entries/23103953-Source-code

02. Caveats and considerations

Custom binaries built by the CDK may be installed on an Opengear unit running stock firmware with a matching model and version. It is important to test future firmware upgrades before upgrading units in production, in case of binary incompatibility (eg library or kernel version changes).

You may also flash the full firmware image built by this CDK. This may be preferable when incorporating custom libraries or other significant modifications from stock. As new firmware versions become available, you will need to port your modifications to the corresponding new CDK and built a new image.

Either way, once an Opengear unit is running code diverges from stock binary, the maintenance and support onus falls to the CDK user.

You are also welcome to enquire directly to request a feature, or to discuss Opengear's custom engineering services via sales@opengear.com.

03. System requirements

To use the CDK you will need:

- a build host running Ubuntu 16.04 LTS (a 'fresh' VM works well).
- four additional required packages supplied by Ubuntu.
- the appropriate CDK .tar.gz supplied by Opengear.
- the cross-compiler toolchains supplied by Opengear.

These instructions are for a 32- or 64-bit install of Ubuntu 16.04 Desktop or Server, available from:

http://releases.ubuntu.com/16.04/

Once your build host is ready, the additional packages required by the CDK build system

build-essential
liblzma-dev
libncurses-dev
bc

can be downloaded and installed from Ubuntu's package repositories using the following command:

```
# sudo apt-get install build-essential liblzma-dev libncurses-dev bc
```

For 64-bit build hosts, additional 32-bit libraries are required for compatibility:

lib32z1
lib32ncurses5

Use the following command to download and install them:

```
# sudo apt-get install lib32z1 lib32ncurses5
```

Download the CDK version matching your Opengear model and firmware version, from:

http://ftp.opengear.com/download/cdk/

Unpack the CDK into a convenient directory. For example:

```
# mkdir ~/cdk
# cd ~/cdk
# tar zxvf ~/Downloads/OpenGear-*-devkit-*.tar.gz
```

To build binaries that will run on the Opengear unit, you need the appropriate cross-compiling toolchains. There are several required as some components (such as the Linux kernel) may require specific compiler versions.

Download each of the .sh files from:

http://ftp.opengear.com/download/cdk/tools/

These self-contained installer scripts, when run as a superuser, extract and install the toolchains under `/usr/local/`.

Run each of them and hit when prompted to accept defaults.

For example, for a script in `~/Downloads/*tools*.sh` run

```
# sudo sh "$script"; done
```

04. Make Commands

It is important to run `make` commands from the root directory of the CDK as the root Makefile contains configuration settings. If you run `make` commands from any other directory, the build will not use the correct cross-compiler.

```
# cd ~/cdk/
```

Run

```
# make help
```

for available make commands.

```
Supported make commands quick reference
-------------------------------------------------------------------
make config                configure the kernel/modules
make dep                   2.4 and earlier kernels need this step
make                       build the entire tree and final images
make clean                 clean out compiled files, but not config
make linux                 compile the kernel only
make romfs                 install all files to romfs directory
make image                 combine romfs and kernel into final image
make modules               build all modules
make romfs.modules         install modules into romfs
make DIR_only              build just the directory DIR
make DIR_romfs             install files from directory DIR to romfs
make DIR_clean             clean just the directory DIR
```

05. Your First Build

The CDK ships with pre-compiled kernel, libraries and stock apps.

Extra user-defined apps are located in the `user` directory, there are several examples included with the CDK. The build system is configured by default to compile these apps, but not install them into the resulting image.

Therefore your first CDK build will produce a near replica of a stock Opengear image. To build this, run:

```
# make
```

The stock apps, libraries and system files are install into the `romfs` directory. This directory forms the basis for the firmware's root filesystem.

A monolithic firmware image containing the romfs and the kernel is created at `images/image.bin`.

06. Building a Custom Application

This example builds and installs the trivial *hello world* app included with the CDK.

Recompile *hello*, without traversing the entire build tree which has already been built:

```
# make user/hello_clean
# make user/hello_only
```

Install *hello* into the `romfs` directory:

```
# make user/hello_romfs
```

Assemble the firmware image. *NB: this also strips the binaries in `romfs`.*

```
# make image
```

You can now copy *hello* from the build host into your Opengear unit's mass storage directory (run `mount` on your Opengear unit to determine this), and try it out using SSH. For example:

```
# scp romfs/bin/hello root@ip.of.opengear:/var/nvlog/usb/
# ssh root@ip.of.opengear /var/mnt/storage.usb/hello
```

should result in the following being sent to std out:

```
 Hello, World
```

07. Customizing Firmware Image Applications

The Opengear firmware's root filesystem is read-only *squashfs*. As such, adding/removing applications from the firmware image involves generating a new file system.

See the previous sections for details on importing your application into the CDK build system.

To have the build system add the applications to the *romfs* automatically (so that they are included in the image), edit `user/Makefile` and add the names of the application directories to RLIST. For example

```
RLIST=hello smstools3
```

To build the custom applications into the firmware, run:

```
# make user_only
# make user_romfs
# make image
```

In the CDK build system the file system is generated from the `romfs` directory.

Note that there are limitations on the final image size, depending on the Flash memory available on your Opengear model. If you would like to add any applications you might consider removing unrequired features.

You may also remove unneeded stock apps from the `romfs` directory before the final `make image` command, to omit them from the image and free up space.

08. Custom Application Makefiles

There are two methods of importing a custom application:

The first is to copy the source code into the CDK tree, under the `user` directory. This may be preferable for a new or in-house application you want to run on the Opengear unit. See `user/hello/` for an example of this method.

The second is to use `automake.inc` to have the build system download an upstream source tarball, and apply any necessary patches before building. This may be preferable for integrating generic open source applications. See `user/smstools3/` for an example of this method, and `tools/automake.txt` for documentation.

Note: Ensure your application has been added to RLIST in `user/Makefile` before continuing. See the previous section for instructions.

If you added your application using the `automake.inc` method, the following steps are handled automatically and your application is installed to `romfs` when you run `make`.

If you added your application directly to the CDK tree, you need to modify its `make` instructions to install relevant binaries into the `romfs` directory. This can be accomplished by either adding a `romfs` target to the application's top level `Makefile`, or by creating a small-m `makefile` to wrap the main `Makefile`.

In the romfs target use the `$(ROMFSINST)` command to copy files into the romfs. It takes two arguments: the source file and the destination location and filename within the romfs. For example:

```
$(ROMFSINST) build/bin/mybin /bin/mybin
```

installs `build/bin/mybin` (path relative to makefile) to `bin/mybin` in the CDK tree's main `romfs` directory.

example makefile with 4 lib dirs and 3 app dirs

```
dirs += \
    lib/aaa \
    lib/bbb \
    lib/ccc \
    lib/ddd \
    app/eee \
    app/fff \
    app/ggg

appdirs += \
    eee \
    fff \
    ggg

all:
    for i in $(dirs); do \
        if [ -d "$$i" ]; then \
            make -C $$i || exit $$? ; \
        fi; \
    done

romfs:
    for i in $(appdirs); do \
        if [ -f app/"$$i"/"$$i" ]; then \
            $(ROMFSINST) app/"$$i"/"$$i" /bin/"$$i" ; \
            fi; \
        done

clean:
    for i in $(dirs); do \
        if [ -d "$$i" ]; then \
            make -C $$i clean ; \
        fi;  \
    done
```

09. Customizing The Kernel

The CDK allows you to build a custom-configured kernel to run on your Opengear unit, including building new kernel modules.

First, configure the kernel:

```
# make config
```

This process is mostly the same as the standard Linux `make menuconfig`. The base configuration is that of the stock Opengear firmware image, so make any required modifications from stock then `<exit>`.

Build the kernel and kernel modules with:

```
# make linux
# make modules
```

Finally, assemble the firmware image with:

```
# make romfs.modules
# make romfs
# make image
```

10. Installing The Image

A monolithic firmware image containing the romfs and the kernel is created at `images/image.bin`.

To install a new firmware image using the web UI click on ***Firmware*** from the ***System*** menu.

Note: by default, your Opengear unit will not allow you to upgrade to the same or an earlier firmware version. To override this type *-i* into the ***Firmware Options*** field.

For details, see:
https://opengear.zendesk.com/entries/22275692-Upgrading-firmware-from-the-Web-Management-UI

Alternatively, you may upgrade from the CLI using the `netflash` command.

For details, see:
https://opengear.zendesk.com/entries/22271241-Upgrading-firmware-from-the-command-line-CLI

11. Factory Reset

This process resets the Opengear unit back to the factory default settings included in the romfs under `/etc/default/` and clears the configuration stored in `/etc/config/`.

To perform a reset to factory settings push the ***erase*** button twice. A ball point pen or paperclip is a suitable tool for performing this procedure. *Do not use a graphite pencil.* Depress the button gently twice within a couple of seconds while the unit is powered on.

At factory default, the Opengear unit requests an IP address via DHCP and is also available at *192.168.0.1/24*.

SSH and HTTPS services are enabled by default, with the default credentials of:

```
Username:    root
Password:    default
```

You may also reset using the web UI under ***System -> Administration***, or the CLI by running:

```
# flatfsd -i
```

For details, see:
https://opengear.zendesk.com/entries/21683672-How-to-reset-back-to-factory-default

12. System Recovery

Should the firmware become corrupt or unusable you can recover the system by performing a netboot. Network booting is the process of booting from a network location (for example, your development machine). This will load the image file from a specified network location into RAM using the primary Ethernet port (NET1/eth0). The device will then boot from the image in RAM. After you have successfully performed a netboot you will need to install a known working image using the web UI.

In order to netboot from your development machine you will require a TFTP server and a DHCP server configured appropriately.

```
# sudo apt-get install atftpd dhcp3-server
```

These are example packages, you may use others if you wish.

Edit your DHCP server configuration (for example `/etc/dhcp/dhcpd.conf`) to include an entry for the Opengear unit. The entry might look something like:

```
host myopengear
    {
        hardware ethernet 00:13:C6:00:00:01;
        fixed-address 192.168.0.1;
        filename "image.bin";
```

Important to note here is the ***filename*** parameter. By default when you build an image the generated file is called `image.bin` and is placed in the images folder. A copy is placed in `/tftpboot/` if it exists.

To effect a netboot:

01. Turn the device off.
02. Depress the ***erase*** button.
03. Turn the device on while still holding the ***erase*** button.

It is recommended to edit your tftp configuration to also use the `/tftpboot/` directory.

For an alternative guide detailing netboot steps using a PC running Windows, refer to: https://opengear.zendesk.com/entries/22270971-Firmware-recovery.