Maemo SDK+: User Guide

Release Information

Project: maemo SDK+

Version: 1.0.19

Baseline: baseline neutral

Date: 2009-11-02

1. Introduction

1.1 About maemo SDK+

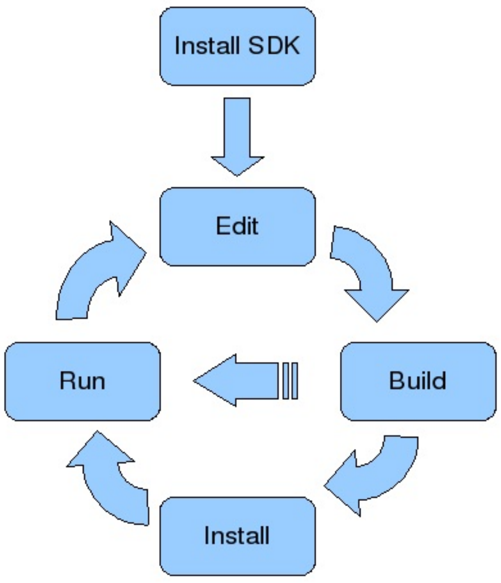
Maemo SDK+ supports cross-compilation and source package building and contains ARMEL and X86 runtime support for running maemo ARM and X86 binaries on the Linux PC side.

This document presupposes an existing installation of maemo SDK+. For installation instructions, see the Installation document.

This user guide is for maemo SDK+ core package. It shows how to use the maemo command line environment to build software for maemo devices.

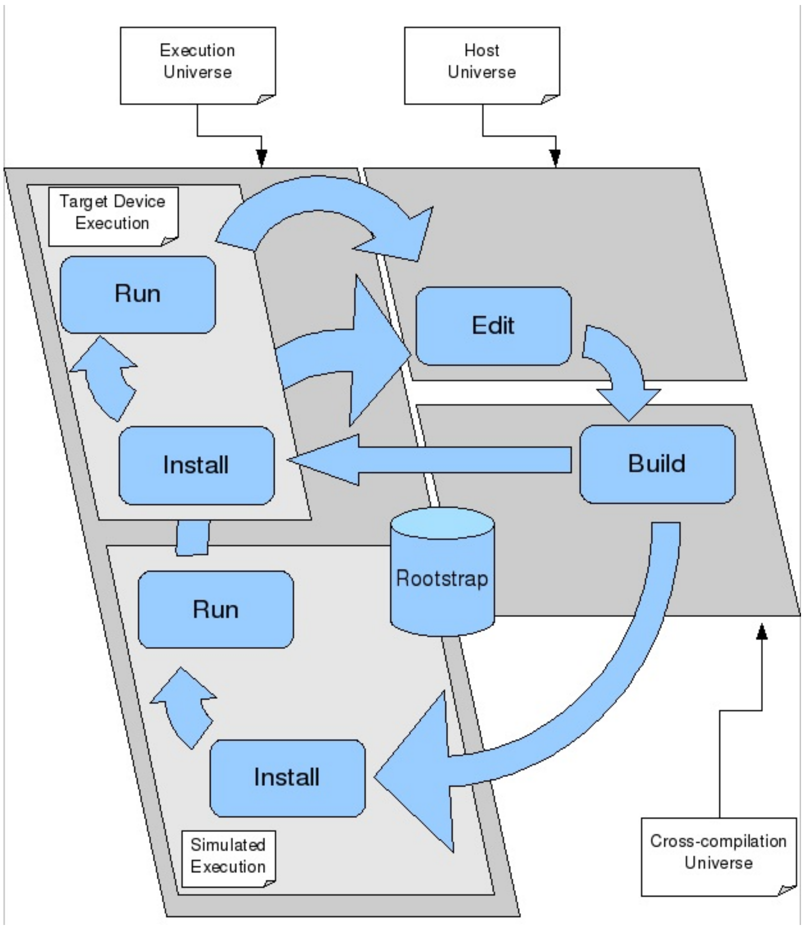
1.2. Cross-Development and SDK

Maemo SDK+ provides a cross-development environment for Nokia's maemo based devices. Cross-development differs from ordinary application development in that applications are built and run on different hardware. In ordinary application development (see figure below) the development process starts with the installation of the development environment. Then an initial application is developed with the editing tools. When application development has reached the point of being buildable, it is built using build tools. If the build was successful, the application is then either installed first, or, run directly to see what happens. Then the application is further developed and the process continues with a new build.



Cross-development, however, is more complicated. It involves a development workstation (host computer) where the software is built and a specific target device like Nokia N810 on which the final application is eventually run. The target device runs a specific operating system version which needs to match the build environment. In addition, a simulated runtime environment may be running on the host to speed up debugging by allowing the application to be run without installing it into an actual device.

Thus cross-development involves multiple universes. Maemo SDK+ itself is installed into a **host universe** consisting of whatever exists on the developer's workstation. The **cross-compilation universe** consists of target device libraries and header files (target device rootstrap) and tools to specificly capable of building software for that target device. The third universe is the **execution universe** which provides means to install and run target software. In fact we have two different execution universes: the actual **target device runtime** and **simulated runtime** on the developer workstation.



With maemo SDK+, the application is edited in the host universe. You can use whatever editor your host provides to write software. Unlike in the old Scratchbox 1 based maemo SDK, you can place the file you edit where you want in our file system; the files do not have to be placed "inside a box".

To build software, you need to enter into the build universe of Scratchbox 2. In the build universe, special tools and a cross-compiler are used to build software using a specific target rootstrap. To enter into the build universe, prefix each command with sb2 (if the build is to be done for the default rootstrap). For instance, sb2 make could be used to run make, or sb2 dpkg-buildpackage -rfakeroot -d to build a Debian package.

To execute the application, either copy and install it into a target device or run it under simulation. If you choose to run it under simulator and the application has a graphical user interface, start the simulated maemo framework first by using the maemo-runtime start command. To enter commands for the simulated runtime universe, prefix each command with sb2 -e (you can also use maemo-sdk enter runtime to enter the sb2 runtime mode). If you need sudo privileges for an operation (like installing an application), use sb2 -eR (or maemo-sdk enter sudo-runtime).

1.3 Cross-compilation Universe

In the Scratchbox 2 cross-compilation universe, system calls related to file system access are remapped. Build tools are taken from a specific build environment; libraries, header files, aclocal files etc. are taken from a target rootstrap; and source code to be compiled is taken as is from the host.

The remapping file refers to modifying the actual file path of the original call. For example, the file execution access /usr/bin/bison is converted to /opt/maemo/dists/etch/usr/bin/bison before launch (all build tools are taken from a build tool distribution). Another example could be an access to a library /usr/lib/libc.a which is converted to ~/.maemo-sdk/rootstraps/armel/diablo4.1.2_armel/usr/lib/libc.a (taken from the Diablo rootstrap). The actual remapping rules of the cross-compilation universe can be read from /usr/share/scratchbox2/lua_scripts/pathmaps/devel/00_default.lua if you have Scratchbox 2 installed.

Because the cross-compilation universe does not map one to one to the target environment, the target packages should not be installed or run there. A separate execution universe is provided which is similar to the target device environment.

1.4 Execution Universe

The execution universe is either an actual target device or a simulated runtime environment. In the simulated runtime environment (sb2 -e operation mode) all system calls are remapped into the target rootstrap, except access to the current working directory. Not remapping the current working directory allows easy access to, e.g.., recently built packages, allowing them to be installed to the target rootstrap without first copying the package inside the rootstrap.

The actual remapping rules of the simulated execution universe are available at /usr/share/scratchbox2/lua_scripts/pathmaps/emulate/00_default.lua.

2. Good to Know First

2.1 Maemo Rootstrap Directories

The maemo rootstrap manager (the "maemo-rootstrap" command line tool) saves the target-specific maemo rootstraps in your home directory under ~/.maemo-sdk. Do not remove the contents of this directory if you need the environment on your system.

Maemo rootstraps are archives stored in .tar.gz format and they contain header files, libraries and tools that are needed to compile and build applications for a specific Internet Tablet device.

When developing software for some of the Nokia Internet Tablet devices, you need to a use rootstrap matching the Tablet OS. For example, do not try to use Bora rootstraps for developing software for N810 Tablet.

Please, keep your own software projects in a separate place (e.g.. ~/MyProjects/ProjectX), not inside rootstraps. By doing this you can erase and reinstall rootstraps easily and develope for multiple rootstraps at the same time.

2.2 Maemo SDK+ Architecture

Maemo SDK+ is based on Scratchbox 2. If you are familiar with Scratchbox 1, it is good to know that technically Scratchbox 2 has pretty much nothing in common with the old version. It is based on a completely new idea on how to cross-compile. It is not a box any more and it certainly is not from scratch either.

When Scratchbox 2 runs a software build, it takes target libraries and header files (and some other files as well) from a target rootstrap. The build tools, however, are taken from a specific build tool environment. The build tool environment is a standard Linux distribution (currently Debian Etch). In addition, Scratchbox 2 replaces all compiler executions with a cross-compiler execution. Multiple cross-compilers can exist on the system at the same time and a rootstrap can easily be reconfigured to use a different compiler version.

The idea behind Scratchbox 2 is path remapping. Path remapping works like a Unix shell on top of the user's shell, wrapping all system calls to check if the file system is accessed. Depending on the access type and Scratchbox 2 mode (either build or emulation mode), the file path may be modified to, for example, access the file from the target rootstrap.

You can have the standard autotool scripts (, i.e., configure scripts), because the checks are run under the path-mapped environment. Autotool tests and some packages generate executables during builds but that is fine because those executables are run under target emulation (using QEMU emulator in user-mode emulation mode). However, because all time consuming build tools are run natively on your host the cross-compilations under Scratchbox 2 are fast.

2.3 Limitations

If you have any problems with Perl libraries during package building, use the following command:

```
$ export SBOX_REDIRECT_FORCE=/usr/bin/perl
```

The issue with Perl libraries is not a bug in the SDK+ itself but in the libraries provided in the rootstraps that Perl uses. The above command forces sb2 to use the Perl interpreter from the Tools Distro and thus avoids the Perl rootstrap problem.

3 How to Use maemo SDK+

3.1 How to install rootstraps

Once the SDK+ installation and rootstrap installation is complete you can start the development. If you have not yet installed any rootstraps, or, you want to install the latest rootstraps, please, update the network catalogue with the following command.

```
$ maemo-sdk reload catalogue
```

This command reloads the rootstrap index file as well as tools and toolchain index files. If you do not want to use the default network location, please, specify the alternative network catalogue location with --index-url option. Now you are ready to start installing new rootstraps. A rootstrap can be installed with the following command.

```
$ maemo-sdk install rootstrap
```

because we did not specify the rootstrap, a menu is displayed from which the rootstrap can be selected. maemo-sdk command is build in a way that if a specifier needed is not given, a menu with alternatives is shown. The same method works with all commands. The rootstrap installation command would then show, for example, the following menu:

```
 0 ... exit              To exit selection

final releases:
 1 ... diablo4.1.2_armel   For OS2008.43-7 (N810/N800)    2008-12-18 (289.3M)
 2 ... diablo4.1.2_i386    For OS2008.43-7 (i386 SDK)     2008-12-18 (291.6M)
 3 ... diablo4.1.1_armel   For OS2008.36-5 (N810/N800)    2008-11-13 (289.3M)
 4 ... diablo4.1.1_i386    For OS2008.36-5 (i386 SDK)     2008-11-13 (291.6M)
 5 ... diablo4.1_armel     For OS2008.23-14 (N810/N800)   2008-11-13 (280.3M)
 6 ... diablo4.1_i386      For OS2008.23-14 (i386 SDK)    2008-11-13 (282.7M)
10 ... scirocco2.2_armel   For latest OS2006 (Nokia 770)  2006-12-22 (107.4M)
11 ... scirocco2.2_i386    For latest OS2006 (i386 SDK)   2006-12-07 (117.3M)
12 ... mistral2.0_armel    For older OS2006 (Nokia 770)   2006-07-05 (138.9M)

alpha releases:
 7 ... fremantle5.0alpha_armel For OS2009Alpha (armel SDK)    2008-12-08 (248.8M)
 8 ... fremantle5.0alpha_i386 For OS2009Alpha (i386 SDK)     2008-12-08 (268.8M)

Select (0..12)?
```

If you now select option 2, for instance, diablo4.1.2_i386 rootstrap is downloaded and installed.

3.2 Examples of the most often used sb2 commands

This chapter explains the most often needed and used sb2-commands that the developer uses.

The sb2 command line options are as follows:

```
 $ sb2 -h

Options:
    -v          display version
    -L level     enable logging (levels=one of error,warning,notice,info,debug,
              noise,noise2,noise3)
    -d           debug mode: log all redirections (logging level=debug)
    -h           print this help
    -t TARGET    target to use, use sb2-config -d TARGET to set a default
    -e           emulation mode
    -m MODE      use mapping mode MODE
    -M file      read mapping rules from "file"
    -s DIRECTORY load mapping scripts from alternative location
    -Q BUGLIST   emulate bugs of the old scratchbox 1 (BUGLIST consists of
              letters: 'x' enables exec permission checking bug emulation)
    -r           do not create reverse mapping rules
    -O options   set options for the selected mapping mode ("options" is
              a mode-specific string)
    -R           use simulated root permissions (currently activates
              "fakeroot" for this functionality)
    -S file      Write session information to "file" (see option -J)
    -J file      Don't create a new session; join an existing one (see -S)
    -D file      delete an old session (see -S). Warning: this does not
              check if the session is still in use!
    -W dir       Use "dir" as the session directory when creating the session
              ("dir" must be absolute path and must not exist. N.B. long
              pathnames here may cause trouble with socket operations)
    -c           When creating a session, also create a private copy
              of target_root (rootstrap). Note that this can be
              really slow, depending on the size of the orig.target_root
  -C dir      When creating a session, create copy of "dir" and use it as the
              target_root (rootstrap).
  -T dir      use "dir" as tools_root (overriding the value from config file)
```

All of the examples below use the default-target ie rootstrap for all operations.

./configure

```
$ sb2 ./configure
```

Using the plain sb2 ./configure command activates the devel-mode mappings for sb2. This means that most tools that are needed during package building process are taken from the installed Tools Distro and not from the rootstrap.

Use this if you are compiling source code that comes from plain .tar.gz archives.

make

```
$ sb2 make
```

Similar than above. This runs the make command in devel mode.

apt-get source

```
$ cd ~/src/some_directory_name
$ sb2 apt-get source your_source_package
```

Getting debian source packages is done using the above command. You don't need the -eR options in this use case.

dpkg-checkbuilddeps

```
$ cd ~/src/your_source_package_name
$ sb2  dpkg-checkbuilddeps
```

The dpkg-bcheckbuilddeps command must be run with the -eR options because it installs packages into the rootstraps.

apt-get build-dep

```
$ cd ~/src/your_source_package_name
$ sb2 -eR apt-get build-dep your_source_package_name
```

The apt-get build-dep command must be run with the -eR options because it installs packages into the rootstraps.

The command fails to run without these options.

dpkg-buildpackage

```
$ cd ~/src/your_source_package_name
$ sb2 dpkg-buildpackage -rfakeroot -b -us -uc
```

Running dpkg-buildpackage is done in the devel-mode. The above example compiles and builds your package and generates .deb binary package(s).

dpkg -i

```
$ cd ~/src/your_source_package_name
$ sb2 -eR dpkg -i your_binary_package_name
```

Running dpkg -i is done in the emulate & fakeroot-mode. The operation needs to install packages to rootstrap and the -R flag provides the simulated sudo access.

apt-get update

```
$ sb2 -eR apt-get update
```

Running apt-get update must be done in the emulate & fakeroot-mode.

apt-cache search

```
$ sb2 -eR apt-cache search some_package_name
```

Running apt-cache search must be done in the emulate & fakeroot-mode.

apt-get install

```
$ sb2 -eR apt-get install your_binary_package_name
```

Running apt-get install must be done in the emulate & fakeroot-mode.

apt-get remove

```
$ sb2 -eR apt-get remove your_binary_package_name
```

Running apt-get remove must be done in the emulate & fakeroot-mode.

apt-get autoremove

```
$ sb2 -eR apt-get autoremove
```

Running apt-get autoremove must be executed in the emulate & fakeroot-mode.

In general, most operations that alter the rootstrap content must be run using the -eR options for sb2. If these options are not used, sb2 uses programs from the Tools Distro and does not allow the process to write anything to the rootstrap area.

The next chapter also uses and demonstrates some of the commands explained above.

3.3 How to build software

The recommended way is to create a folder for your projects in your home directory. Do not use the rootstrap directory as a working directory because you might later want to update its contents with the rootstrap manager. Abetter option would be to create separate a working directory:

```
$ mkdir -p ~/MyProjects/maemo
$ cd ~/MyProjects/maemo
```

Next, fetch sample source code by using the following commands:

```
$ sb2 -eR -t diablo4.1.2_armel apt-get update
$ sb2 -t diablo4.1.2_armel apt-get source maemopad
```

Now the ~/MyProjects/maemo directory should have a maemopad-2.1 subdirectory. Note that we used the nickname diablo4.1.2_armel to explicitly specify the rootstrap to use for building. To set up a default rootstrap in Scratchbox 2, use the following command:

```
$ maemo-sdk set rootstrap diablo4.1.2_armel
```

The rootstrap to use does not need to be explicitly specified every time. Once Diablo 4.1.2 ARMEL has been set as the default rootstrap, build the Maemopad by using the following command:

```
$ sb2 dpkg-buildpackage -rfakeroot -d -b
```

This builds a fresh maemopad_2.4_armel.deb package in the ~/MyProjects/maemo directory. Copy this file to a N810 device running OS2007 and install it using the application installer.

Build dependencies can be checked so that dpkg-buildpackage can be run without -d option. Alternatively, you can use the explicit -D option. When run for the first time, the process takes a minute or two. This is because a database is built by collecting package information from multiple sources (from the rootstrap and build environment apt databases).

To build from the source using ./configure or make, prefix all commands with sb2.

```
$ sb2 ./configure && sb2 make
```

3.4 How to install software with make

Installing the software using make install requires the target destination directory for the command because the ARM binaries should not be installed into the build environment. In order to specify the target rootstrap for the make install, use the normal method of passing the destination directory for your specific Makefile. Usually it involves defining a DESTDIR variable. Thus, to use the make based method to install maemo SDK+ use the following command:

```
$ sb2 make install DESTDIR=/target_root
```

To explicitly specify the rootstrap to be used, use the following command:

```
$ sb2 -t diablo4.1.2_armel make install DESTDIR=/target_root
```

3.5 How to install packages into the runtime environment

To use apt-get for installing packages into the runtime, use the following command:

```
$ sb2 -eR -t diablo4.1.2_armel apt-get install maemopad
```

To install a newly created package, such as maemopad_2.4_armel.deb, go to the directory where the .deb file is and use the following command:

```
$ sb2 -eR -t diablo4.1.2_armel dpkg -i maemopad_2.4_armel.deb
```

4 Runtime debugging

4.1 Starting runtime

To start the runtime debugging environment, use the following command:

```
$ maemo-sdk start gui diablo4.1.2_armel
```

The runtime is up and running in a few seconds. Note that Xephyr might not be able to start properly if your X server does not support the color bit depth specified for the rootstrap (most likely 16 bits). If the GUI does not start up, try forcing maemo-sdk to use some other color bit depth, e.g.. 24 bits, by using the following command:

```
$ maemo-sdk set gui-bit-depth 24
```

4.2 Stopping runtime

To stop the runtime environment, use the following command:

```
$ maemo-sdk stop gui diablo4.1.2_armel
```

4.3 Entering runtime shell

To give shell commands for an SDK+ runtime, either enter into a runtime shell or give commands one by one. Scratchbox 2 can provide multiple concurrent sessions for one rootstrap. Each session has its own /tmp folder mapping, for instance. Thus, if your specific command requires GUI access, for instance, and you use the sb2 -e command directly, provide the session identifier as well. To make all this easy, open a shell that accesses the same runtime session where the previously started GUI operates. To do this, use the following command:

```
$ maemo-sdk enter runtime diablo4.1.2_armel
```

To launch an application from inside the diablo4.1.2_armel session, use the following command:

```
[SB2 emulate diablo4.1.2_armel] $ run-standalone.sh maemopad
```

To exit the session, use the following command:

```
[SB2 emulate diablo4.1.2_armel] $ exit
```

To run commands without entering into the shell, use the following command:

```
$ maemo-sdk enter runtime diablo4.1.2_armel 'run-standalone.sh maemopad'
```

4.3 Entering runtime shell as root user

Root privileges are needed for some operations. To operate with root privileges in maemo SDK+, use the following command:

```
$ maemo-sdk enter sudo-runtime
```

Note that the sb2 -eR command does the same but is shorter.

5 Managing the build environment

5.1 Adding tools

To extend the build environment to include a new tool, for example, you can use maemo-tools get command. For example, to add Vala 0.7.7 support, use the following command:

```
$ sudo maemo-tools get vala-0.7.7
```

5.2 Searching for available tools

In general, to get a new tool into the build environment it must be made available for Debian Etch. To find out if the tool exists, use the following command:

```
$ sudo maemo-tools search [toolname]
```

5.3 Manually extending the build environment

If the tool is not available, try compiling it for Debian Lenny. Use the following commands (replacing tool with your own tool package name):

```
$ cp tool.tar.gz /opt/maemo/dists/etch/tmp
$ sudo chroot /opt/maemo/dists/etch
$ cd /tmp
$ gunzip tool.tar.gz
$ tar xvf tool.tar && cd tool
$ ./configure --prefix=/usr
$ make
$ make install
$ exit
```

5.4 Listing installed packages

If you want to display the whole configuration of the SDK+ (e.g.. to be attached into a bug report, for instance) you can use the following command. This command prints the content of both the build tools environment and the rootstrap as well as the version numbers of maemo SDK+ packages.

```
$ maemo-sdk show config
```

5.5 Changing toolchain (gcc cross-compiler) version

maemo SDK+ allows you to work with multiple toolchain versions. Currently it is recommended to use arm-2005q3 to build maemo software. If you want to use a different version, , i.e., arm-2007q3, please install it first by typing:

```
$ sudo maemo-sdk install toolchain arm-2007q3
```

The cross-compiler is fixed per rootstrap installation. To change the cross-compiler for a particular rootstrap, for example diablo4.1.2_armel to use arm-2007q3, use the following command:

```
$ maemo-sdk set toolchain diablo4.1.2_armel arm-2007q3
```

To change back to arm-2005q3, use the following command:

```
$ maemo-sdk set toolchain diablo4.1.2_armel arm-2005q3
```

6 Changing configuration

6.1 Changing the default rootstrap

To change the default rootstrap, use the following command:

```
$ maemo-sdk set rootstrap diablo4.1.2_armel
```

6.2 Autosudo

If you have set sudo properly for your username (in Ubuntu this is done by default) you can use the following command to turn on the autosudo method:

```
$ maemo-sdk set autosudo on
```

Now every time your maemo-sdk command requires super user privileges, the command automatically reruns itself under sudo. To disable this feature, use the following command:

```
$ maemo-sdk set autosudo off
```

6.3 Network catalogue index

To use a non-default location for the network index files, you can use the following command to change the setting:

```
$ maemo-sdk set url http://some_web_address/
```

To restore the default URL setting, use the following command::

```
$ maemo-sdk set url default
```

6.4 Network proxy setting

maemo-sdk uses the http_proxy environment variable setting as a proxy when accessing the network. To give a default proxy setting, which is used when http_proxy has not been set, for the maemo-sdk, use the following command:

```
$ maemo-sdk set proxy http://some_proxy_address:some_port_number
```

7 Feedback and further help

There are multiple ways of giving feedback about this product. If you have any problems or questions on how to use SB2 based maemo SDK+ development environment, feel free to contact us:

* Maemo SDK+ developer forum

* maemo-sdk-developers@maemo.org mailing list

* Maemo SDK+ garage for bug reports and feature requests.