Installing Scratchbox

Valtteri Rahkonen

valtteri.rahkonen@movial.fi

Installing Scratchbox

by Valtteri Rahkonen

Revision history

Version:	Author:	Description:
2005-04-21	Savola	Updated environment variables
2005-02-17	Savola	Updated environment variables
2005-02-06	Savola	Updates
2005-01-05	Rahkonen	Initial version

Table of Contents

Chapter 1. Introduction

Scratchbox [1] is a cross-compilation toolkit designed to make embedded Linux application development easier. It provides a full set of tools to integrate and cross-compile an entire Linux distribution. Scratchbox supports cross-compiling for ARM and PowerPC targets. Especially Debian [2] is supported, but Scratchbox has also been used to cross-compile e.g. Slackware for ARM.

Scratchbox provides a sandboxed build environment which offers a controlled set of tools and utilities needed for cross-compilation. It is an environment in which it is easy to assure that the intended versions of libraries, headers and other similar files are used during the build. Most of the higher level software built using GNU Autotools do not cross-compile well in their as-is form, Scratchbox solves this problem by allowing the small test programs (used by the configure script to test for availability of features in the environment) to run transparently either using an emulator or through sbrsh protocol between Scratchbox and actual target device. In practice software configuration and building using Scratchbox is quite identical to how it's done for the desktop.

Scratchbox has been designed to allow multiple application developers work simultaneously on a single host machine. Each developer has his private user account, and all configuration is developer-specific.

This document describes Scratchbox install procedure and how Scratchbox can be used to cross-compile applications to ARM target platform.

1.1. Requirements for the host system

Requirements for using Scratchbox are:

- Linux distribution installed on the host.
  - Debian GNU/Linux is recommended.
- x86 platform.
  - For multiple users using Scratchbox on the same host a dual CPU machine or Hyper-Threading capable processor is recommended.
- 512 MB of memory.
  - For multiple users using Scratchbox on the same host at least 1 GB is recommended.
- Full Scratchbox installation requires 1 GB of hard disk space, but you should reserve at least 1 GB extra space for each Scratchbox user.
  - For heavy development you might need at least 10 GB of space.
- Your kernel has the binfmt_misc module which is required by the CPU-transparency feature. On most Linux distributions it is available by default (not on RedHat Enterprise Linux 3).

Recommendations for using Scratchbox are:

- Working networking environment.
- Working NFS server.

1.2. Requirements for an ARM device

When using sbrsh for implementing CPU transparency, a physical target device is needed. Requirements for an ARM-based target device are:

- A Linux ARM device with networking. An iPAQ is recommended as it has known stable Linux support. Handhelds.org has information about Linux support for iPAQs [3]. Scratchbox reference target device is Compaq iPAQ H3600 series PDA. Scratchbox has been tested and verified to work with the following devices:
  - Compaq iPAQ H3630
  - Compaq iPAQ H3870
  - Compaq iPAQ H5550
  - Iyonix PC
- You have a suitable ARM distribution installed on the device. Recommended Linux distributions are Debian and Familiar 0.8 [4].
- Network between your host machine and the ARM device has been setup and works correctly, see: Support distribution installation instructions or How to setup iPAQ networking in Familiar. Having sshd on the device is recommended.

Section 2.4 describes how to set up a Scratchbox target to use the ARM device.

Chapter 2. Installing Scratchbox

2.1. Scratchbox packages

Base (required):

- scratchbox-core - environment, common tools and host compiler
- scratchbox-libs - libraries required by core, devkits and toolchains

Development kits (optional):

- scratchbox-devkit-debian - environment and tools for Debian development
- scratchbox-devkit-doctools - document generation tools
- scratchbox-devkit-perl - additional Perl modules

Toolchains (optional):

- scratchbox-toolchain-arm-gcc3.3-glibc2.3
- scratchbox-toolchain-i686-gcc3.3-glibc2.3
- scratchbox-toolchain-arm-gcc3.2-uclibc20040229
- scratchbox-toolchain-i386-gcc3.2-uclibc20040229

You can also build custom toolchain packages. See *Scratchbox toolchains* [5].

Scratchbox packages are available for Debian and RPM-based systems. You can install Scratchbox from binary tarballs for other Linux distributions. Installing from Debian packages and tarballs is described in the following sections.

2.2. Installing Scratchbox on the host system

2.2.1. Installing Scratchbox on Debian GNU/Linux

You will need root privileges for this part of the Scratchbox installation.

1. Add this line to the /etc/apt/sources.list file:

   ```
   deb http://scratchbox.org/debian ./
   ```

2. Update the package list with command:

   ```
   # apt-get update
   ```

3. Install packages:

   # **apt-get install <packages>**

   Note: If '/' directory does not contain enough space for Scratchbox it can be installed to some other partition by creating a symbolic link from '/scratchbox' to desired place with command: **ln -s /opt/sb /scratchbox**.

4. After downloading Scratchbox will be unpacked to '/scratchbox' directory and the installation procedure will ask you some questions about the group and user accounts. Default group to Scratchbox users is 'sbox'. Group can be renamed but default should be fine unless you have LDAP or similar network authentication scheme. If network authentication is used using an existing group is recommended.

   Users who will be using Scratchbox should be added using command:

   # **sb-adduser <username>**
   It will automatically include users to the Scratchbox group, create user directories under '/scratchbox/users' directory and mount several directories (/dev, /proc, /tmp) under user directory.

This is all that needs to be done for basic Scratchbox functionality. To be able to cross-compile more complex software you need to enable the sbrsh feature (see Chapter 3).

2.2.2. Installing Scratchbox on other Linux distributions

You need root privileges for this part of the Scratchbox installation.

1. Obtain necessary packages from Scratchbox download area [6].
2. Uncompress tarballs to the / directory (for each tarball):

   # **tar zxf <package> -C /**

   Note: If '/' directory does not contain enough space for Scratchbox it can be installed to some other partition by creating a symbolic link from '/scratchbox' to desired place with command: **ln -s /opt/sb /scratchbox**.

3. After extraction configure Scratchbox with following command:

   # **/scratchbox/run_me_first.sh**

Answer questions (defaults should be fine). This creates 'sbox' user group and setup Scratchbox.

4. Add users to the Scratchbox with command:

   # **/scratchbox/sbin/sbox_adduser <username>**

   This adds the user to 'sbox' user group, creates the Scratchbox user directory under '/scratcbox/users' directory and mounts several directories (/dev, /proc, /tmp) under user directory.

2.3. Starting Scratchbox

1. If you were logged into the Scratchbox machine before you were added as a Scratchbox user, you may need to re-login to your machine, so that you get 'sbox' group privileges needed for running Scratchbox. You can check this by running the following command:

   $ **groups**

   If it prints out the sbox group name, you're ready to start Scratchbox.

2. Start Scratchbox with command:

   $ **/scratchbox/login**

   Note: If you have installed Scratchbox from Debian or RPM packages, you can use **/usr/bin/scratchbox** to start Scratchbox.

   When you login for the first time, Scratchbox copies the terminfo terminal capability database to your Scratchbox home directory (/scratchbox/users/<username>/home/<username>/) and creates the default HOST target.

   Note: Because Scratchbox is a fully self-contained sandbox, it doesn't inherit anything from the outside system except for the environment variables. You can copy your configuration files from your normal home directory into your Scratchbox home directory: .bashrc (Scratchbox uses the bash shell), .bash_profile, .vimrc, .inputrc, .cvspass, etc.

2.4. Creating cross-compilation target for ARM

New targets can be created using the menu-based **sb-menu** utility. The same functionality is also available via the **sb-conf** and **sbrsh-conf** command-line utilities, but they are not described in this document.

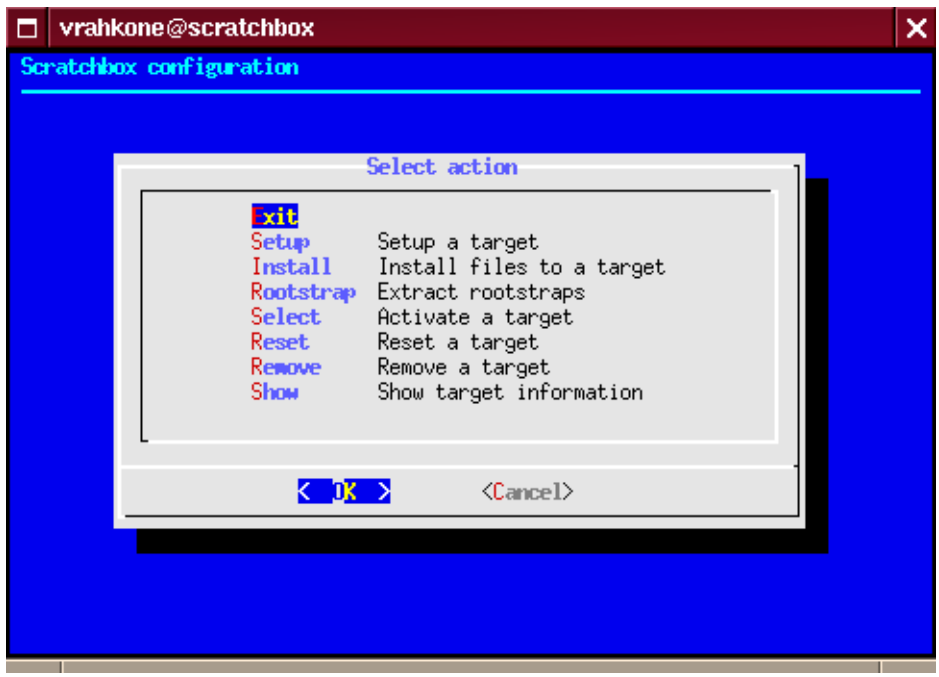
> **Note:** The sb-menu command is available inside and outside Scratchbox. If you are using the Debian or RPM packages, you can use the **/usr/bin/sb-menu** command outside Scratchbox; the **/scratchbox/tools/bin/sb-menu** command is included also in the tarballs.

Follow these steps to create a new cross-compilation target:

1. Start sb-menu:

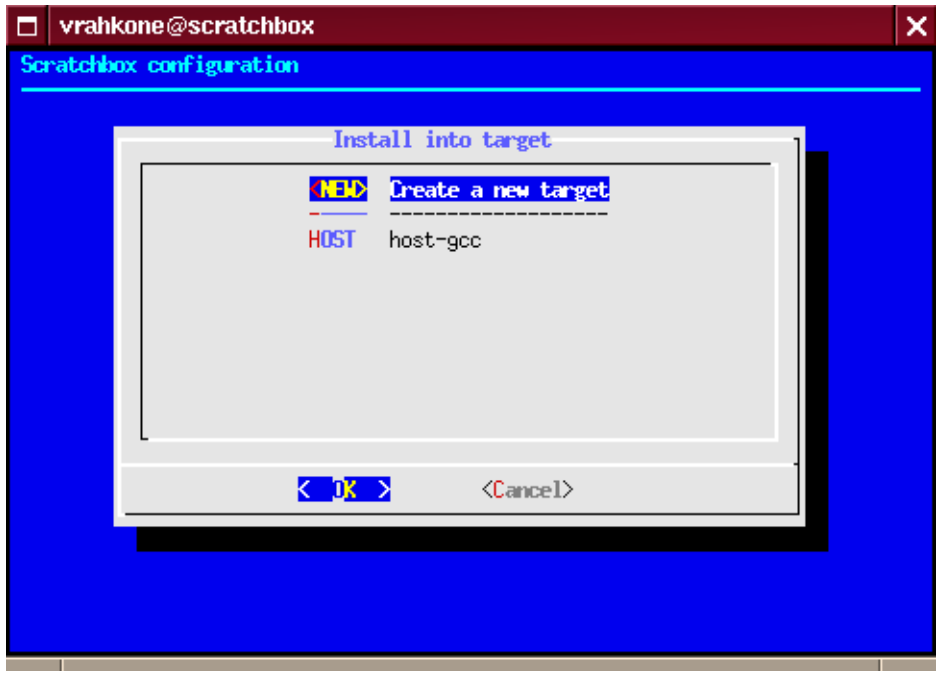
   ```
   [sbox-HOST: ~] > sb-menu
   ```

2. You will enter the main menu:



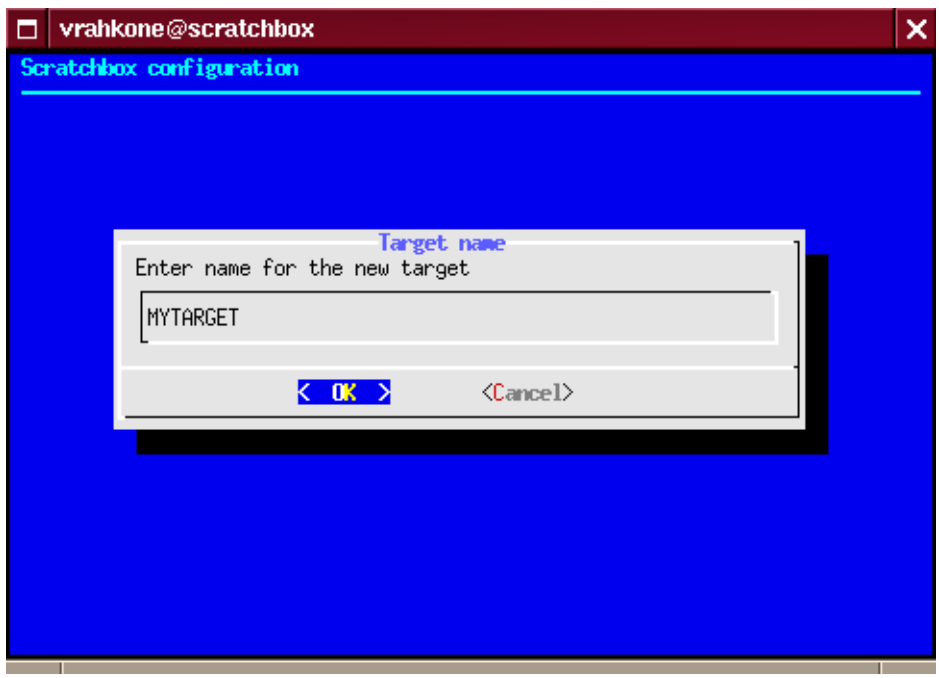
   Select the Setup option.

3. First it will ask if you want to create a new target or change the configuration of a previously created target:



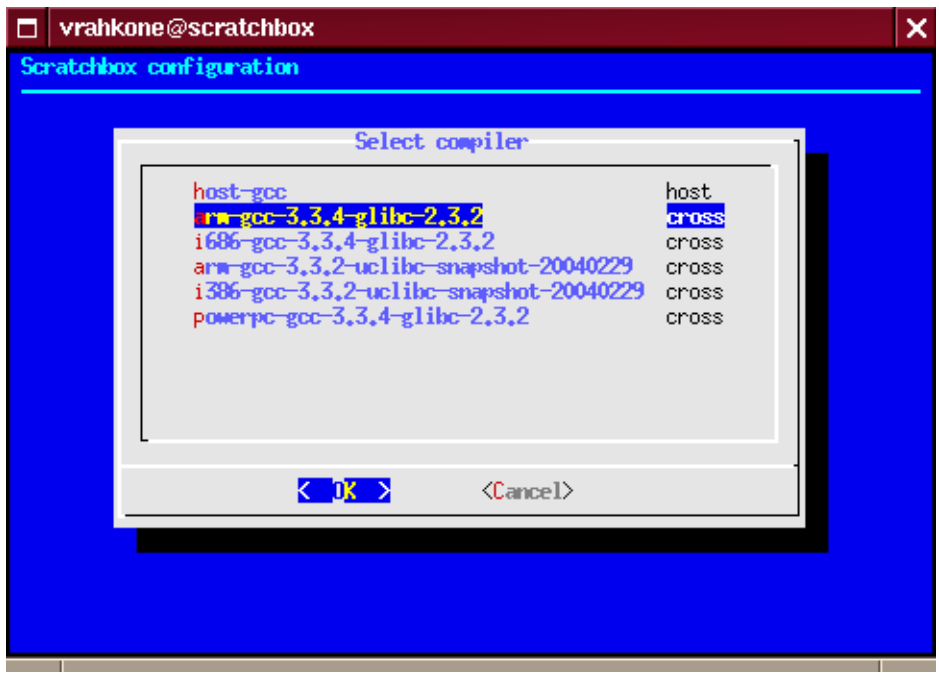
Select the NEW option.

4. Since we're creating a new target we need to specify a name:



We use "MYTARGET" as the target name in this example.

5. Next, the setup will ask which toolchain should be used to compile programs:

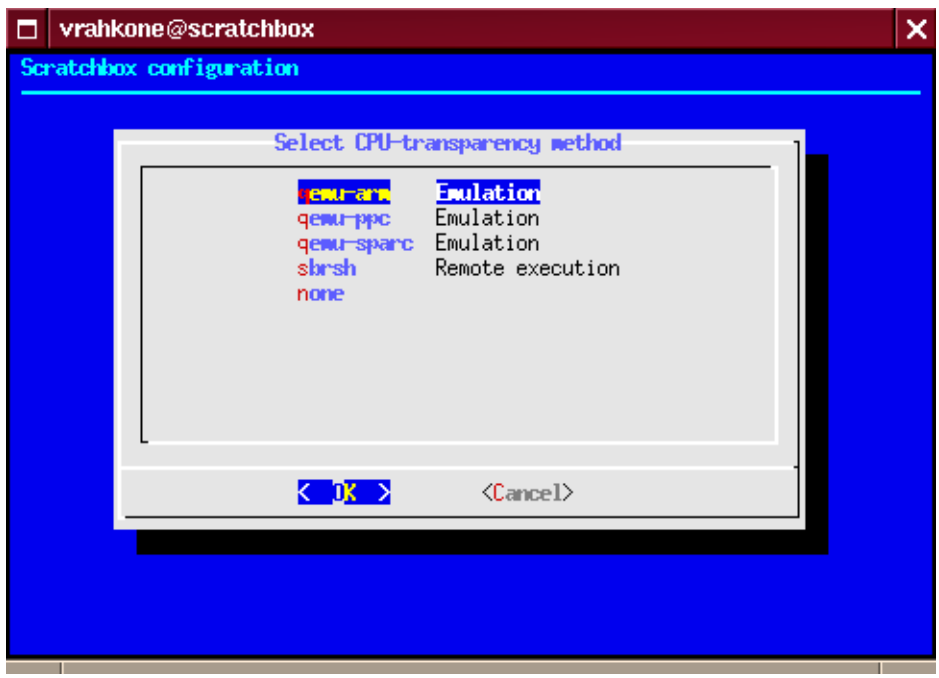


We selected gcc 3.3 that creates binaries for the ARM architecture and links them against glibc 2.3.

6. We can select optional development tools:

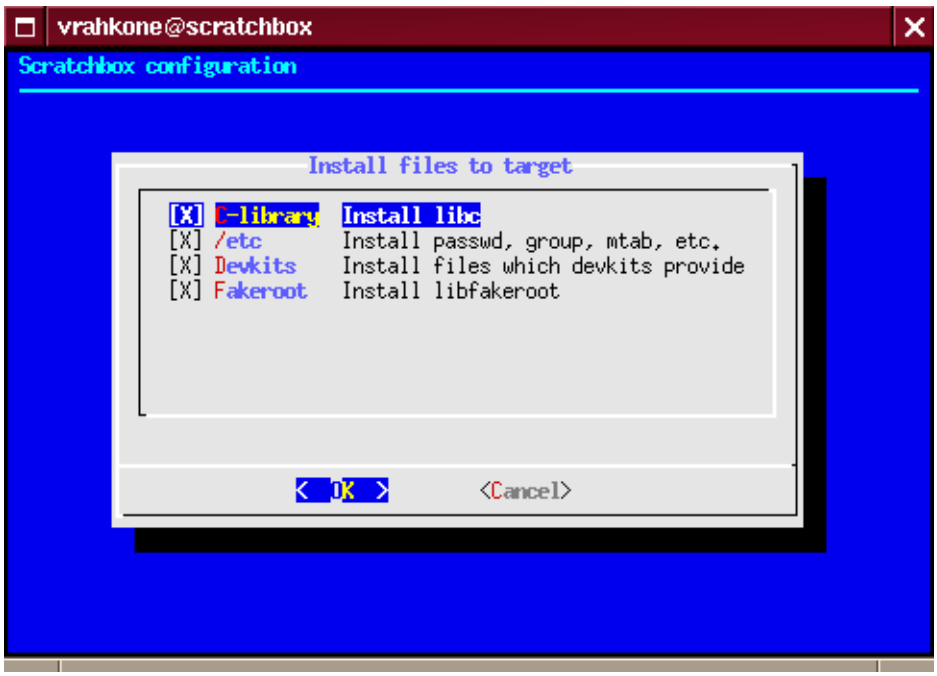
At this point we don’t need any. Just select the DONE option to proceed.

7. After development kit selection the CPU-transparency method is chosen:



The QEMU emulator is sufficient for our example target, so select ’qemu-arm’.

8. Now the setup is done, but sb-menu asks if we want to extract a rootstrap and/or install system files on the target. Answer no to the rootstrap question but choose to install files.
9. You can install the C-library (and related binaries) provided by the toolchain, some standard config files in /etc, config files required by the selected devkits (we have not selected any) and some target-specific binaries:



The default selection is fine, so you can just press enter to install them.

10. Everything is now ready and we can activate the target by answering yes to the last question.

Scratchbox is now ready for cross-compilation for ARM with the help of the QEMU emulator. QEMU is suitable for basic use but sbrsh with an actual target device should be used for advanced requirements (see Chapter 3).

2.5. Testing installation

The new target can be tested with a simple command-line program. Our example program uses GNU

Autotools for detecting compilation environment. It can be compiled by following these steps:

1. Extract hello-world's source code from the /scratchbox/packages directory:

   ```
   [sbox-MYTARGET: ~] > tar xfz /scratchbox/packages/hello-world.tar.gz
   ```

2. Go to the created 'hello-world' directory:

   ```
   [sbox-MYTARGET: ~] > cd hello-world
   ```

3. Generate the configure script, configure the program and generate its Makefile:

   ```
   [sbox-MYTARGET: ~/hello-world] > ./autogen.sh
   ```

4. Compile it:

   ```
   [sbox-MYTARGET: ~/hello-world] > make
   ```

5. We should now have an executable ARM binary named 'hello'. This can be verified with command:

   ```
   [sbox-MYTARGET: ~/hello-world] > file hello
   hello: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux 2.0.0,
   dynamically linked (uses shared libs), not stripped
   ```

6. Let's run the program:

   ```
   [sbox-MYTARGET: ~/hello-world] > ./hello
   Hello World!
   ```

If support for more complicated programs (for example GUI software) is needed, the libraries that they use should be installed on the target. One option is to compile them from sources and another is to use prebuilt binary packages. If you are using the Debian devkit, you can use apt-get to install packages on the target (see Section 4.3).

Chapter 3. Setting up sbrsh

Sbrsh is an alternative to QEMU for implementing the CPU-transparency feature of Scratchbox. It runs the configure scripts' test programs on a remote device with the CPU architecture used by the cross-compilation toolchain—typically the device that you are developing software for. Some build systems also benefit from it when they attempt to execute a target binary which is used to generate data files. Running programs on an actual target device is more reliable than emulating a specific device because emulators might not support all required features or there might not be an appropriate emulator available at all.

3.1. Setting up NFS environment

Sbrsh requires that the user's home (build) and target directories are available on the target device. This is done using NFS, so you have to setup your machine to export the required directories. NFS setup has to be done as root.

Note: NFS is inherently insecure so you should run it only on trusted networks.

1. On Debian, the 'nfs-common', 'nfs-kernel-server' and 'portmap' packages are needed. On RedHat you need the 'nfs-utils' and 'portmap' packages.
2. Add the user's Scratchbox home and target directories to the '/etc/exports' file on the host machine. The export lines should look something like this:

   ```
   /scratchbox/users/<USER>/targets/<TARGET> <IP>(rw,all_squash,anonuid=<UID>,
   anongid=<GID>)
   /scratchbox/users/<USER>/home <IP>(rw,all_squash,anonuid=<UID>,anongid=<GID>)
   ```

   where <IP> is replaced with the hostname or IP-address of your target device, <UID> with the user's user ID and <GID> with the user's primary group ID. (You can get the last two values from the output of command: **id <USER>**.)
3. Now the NFS server can be started.
   - On Debian it is done with command:

     # **/etc/init.d/nfs-kernel-server start**
   - On RedHat it is done with command:

     # **service nfs start**

This is all that needs to be done for the NFS configuration in order to use sbrsh.

More information on NFS is available at [7]. However, the following commands might be useful:

- If you make changes to the /etc/exports file, you need to tell the NFS server to reread it:

```
# /usr/sbin/exportfs -rv
```

Alternatively the NFS server can be restarted.

- On Debian it is done with command:

  ```
  # /etc/init.d/nfs-kernel-server restart
  ```

- On RedHat it is done with command:

  ```
  # service nfs restart
  ```

- If you don't want to export filesystems anymore, run this:

  ```
  # /usr/sbin/exportfs -rv
  ```

  You could also stop the NFS server.

  - On Debian it is done with command:

    ```
    # /etc/init.d/nfs-kernel-server stop
    ```

  - On RedHat it is done with command:

    ```
    # service nfs stop
    ```

Note: If you remove a Scratchbox target and create a new target with the same name, you need to restart the NFS server before you can use it with sbrsh. If you just want to clear the target, you should *reset* it instead of removing since it does not require an NFS restart.

3.2. Installing sbrsh daemon

In order to use sbrsh you need to install and start the sbrsh daemon (sbrshd) on the target device. It needs to be run as root, so you need root access.

If you are have a Debian installation on the target device you can install sbrshd from a Debian package. You can find it from the `/scratchbox/device_tools` directory tree of your Scratchbox installation or from the APT repository at scratchbox.org.

- If you want to use APT, follow these steps at the target device:

  1. Add the following line to the '/etc/apt/sources.list' file:

     ```
     deb http://scratchbox.org/debian/device ./
     ```

2. Update the package database:

   ```
   my-device:~# apt-get update
   ```

3. Install the sbrshd package:

   ```
   my-device:~# apt-get install sbrshd
   ```

- If you want to copy the package to the device (for example, the device is not connected to the Internet or scratchbox.org does not provide a package suitable for your device), you need ssh or some other means to transfer files between your Scratchbox host and the device. In this example we assume you have sshd running at the device:

  1. Copy the sbrshd package suitable for your device over to the device (replace 'my-device' with the proper hostname or IP-address):

     ```
     $ scp /scratchbox/device_tools/sbrsh-6.6/arm-gcc-3.3.4-glibc-2.3.2/
       sbrshd_6.6_arm.deb root@my-device:
     ```

  2. Install the sbrshd package on the target device:

     ```
     my-device:~# dpkg -i sbrshd_6.6_arm.deb
     ```

If you are not using Debian on the device, you have to install the files by hand. You need ssh or some other means to transfer files between your Scratchbox host and the device. In this example we assume you have sshd running at the device:

1. Copy the sbrshd binary suitable for your device to the /usr/sbin directory of the device (replace 'my-device' with the proper hostname or IP-address):

   ```
   $ scp /scratchbox/device_tools/sbrsh-6.6/arm-gcc-3.3.4-glibc-2.3.2/
     sbin/sbrshd root@my-device:/usr/sbin/
   ```

2. Copy the sbrshd init script to the /etc/init.d directory (or some other directory depending on the Linux distribution installed on the device):

   ```
   $ scp /scratchbox/device_tools/sbrsh-6.6/arm-gcc-3.3.4-glibc-2.3.2/
     etc/init.d/sbrshd root@my-device:/etc/init.d/
   ```

3. Start sbrshd on the target device:

   ```
   my-device:~# /etc/init.d/sbrshd start
   ```

4. If you want to start sbrshd automatically at boot, you need to create links at the runlevel directories. On some distributions you can do so using the **update-rc.d** command. The following example works on Familiar:

   ```
   my-device:~# /usr/sbin/update-rc.d sbrshd defaults
   ```

3.2.1. sbrsh daemon configuration

sbrshd does not accept any connections by default. The '/etc/sbrshd.conf' file configures the IP-addresses and user accounts that are allowed to access the daemon. You can either edit sbrshd.conf by hand or add

access rights using the following command:

```
my-device:~# /usr/sbin/sbrshd add <user>@<address>
```

where <user> is a user account on the Scratchbox host with IP-address <address>. <user> can be the wildcard "*" which means that all users from that host are granted access. The IP-address can also contain the wildcard at its end. Here are a few examples:

```
my-device:~# sbrshd add bob@10.0.0.1
my-device:~# sbrshd add *@192.168.0.200
my-device:~# sbrshd add alice@10.0.*
my-device:~# sbrshd add *@*
```

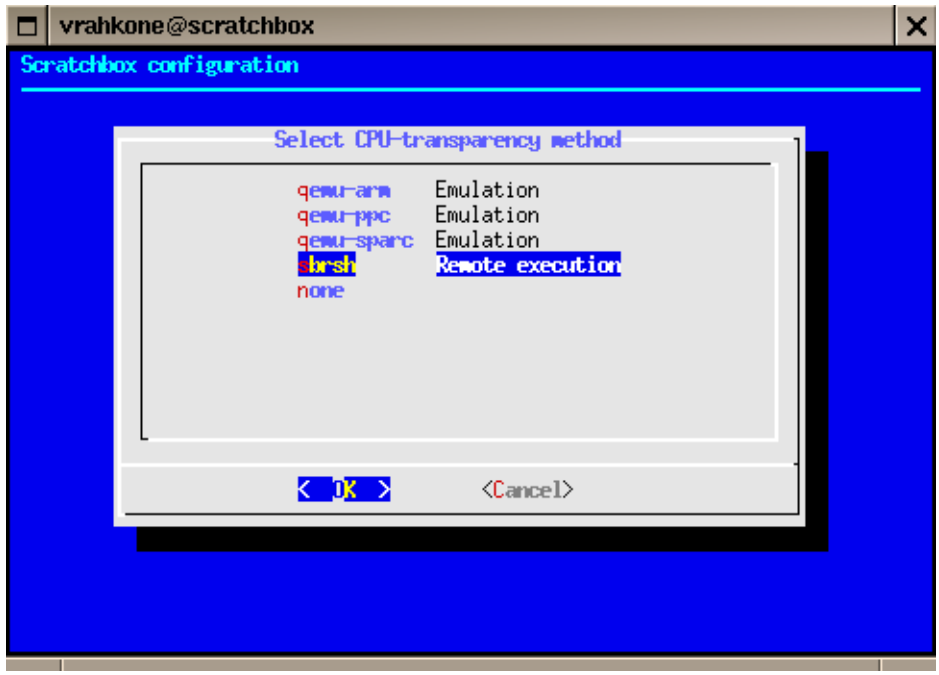
See *Scratchbox Remote Shell* [8] for detailed sbrshd usage information.

3.3. Configuring sbrsh

Sbrsh needs to be configured for each Scratchbox target it is used with. The configuration is saved in the '/targets/<targetname>.sbrsh' file inside Scratchbox. It contains the hostname of the target device and the port where the sbrshd listens at, and lists the NFS filesystems that need to be mounted and directories that need to be bound in order to recreate the Scratchbox sandbox at the device end.

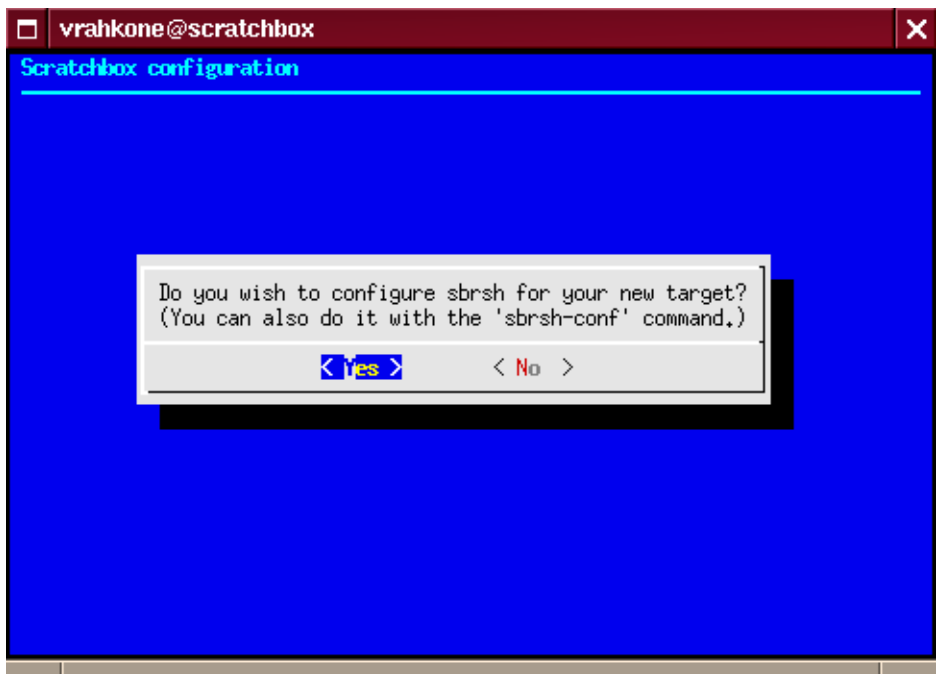
In this example we are going to reconfigure the target that was created before. It can be done with the following steps:

1. Start **sb-menu** and proceed to setup the existing target called "MYTARGET". Select the same toolchain we selected before (see Section 2.4) and no devkits. After those steps you reach the CPU-transparency choice:



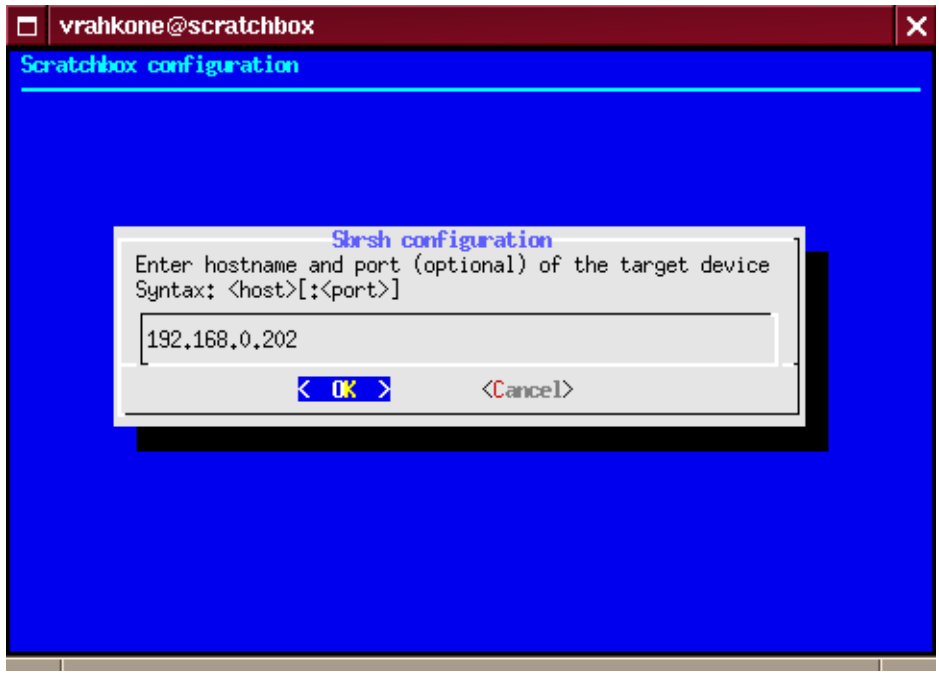
This time select ’sbrsh’.

2. Since you selected sbrsh, the setup asks if you would like to configure it:



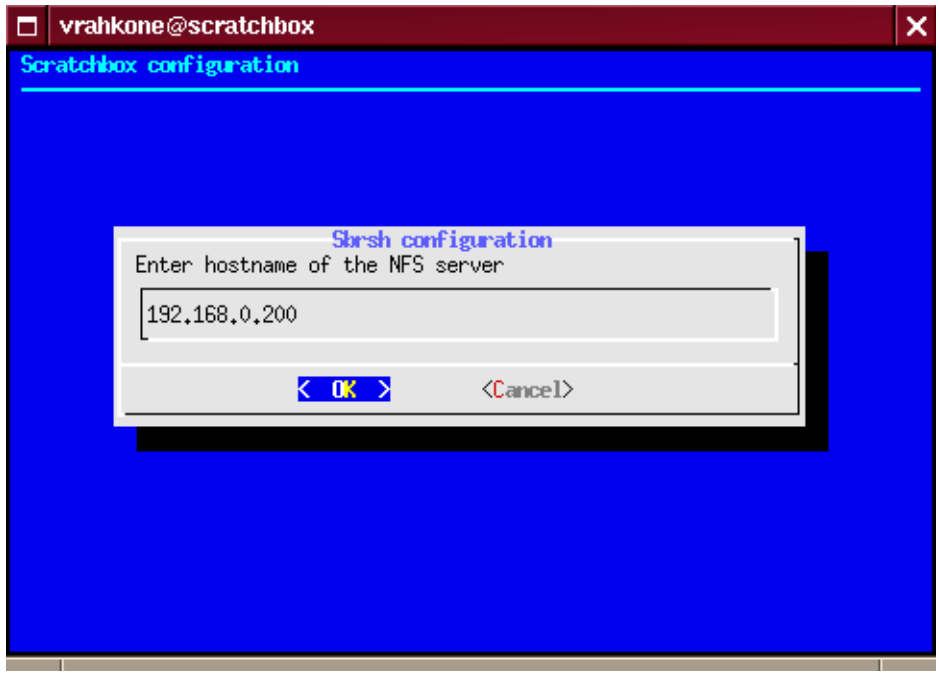
Say Yes.

3. First we need to specify the hostname or IP-address of the target device. If you have configured the sbrshd to use a nonstandard port you can specify that as well.



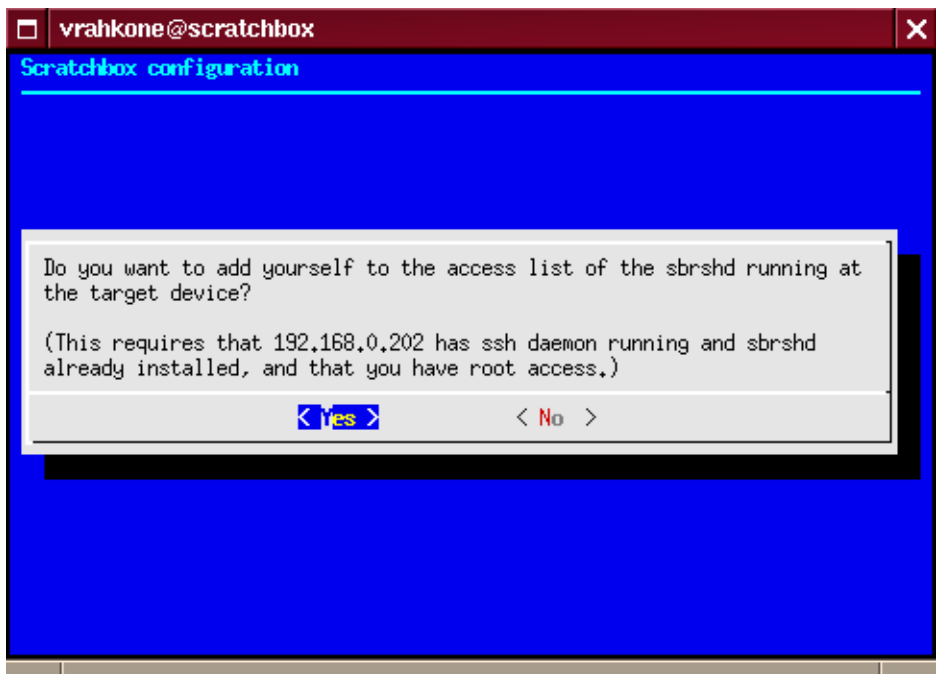
   In our example the device has been connected to the Scratchbox host via a local USB network and we have assigned the IP-address 192.168.0.202 for it. We are running sbrshd at the default port (1202) so we don't need to specify it.

4. The next dialog shows the default hostname of your Scratchbox host. It is used to mount the NFS filesystems on the target device, so you may need to change it to an address that is known to the device.



We need to specify the IP-address that is assigned to the Scratchbox host in the USB network. In this example it is the gateway: 192.168.0.200.

5. Now you have the possibility to add yourself to sbrshd's access list if you have root access (see Section 3.2.1):



You should say Yes if you didn’t configure the permissions while installing sbrshd.

6. Now sbrsh is configured for the target and you have returned to main menu. Select exit and proceed to test your new configuration.

3.4. Testing sbrsh

Sbrsh can be tested by repeating same steps that were done in testing installation section (see Section 2.5). This time the only difference is that target binaries are executed on real hardware instead of being emulated.

Chapter 4. Setting up Debian environment

This section describes how to set up a Debian environment inside Scratchbox. The Debian development kit offers tools for creating Debian packages in the same way they are created on a native Debian system.

4.1. Installing Debian devkit

If you did not already do so when installing Scratchbox (see Chapter 2), you need to install the optional 'scratchbox-devkit-debian' package in order to do Debian development. If you installed Scratchbox from Debian packages, you can install Debian devkit with the following command:

```
# apt-get install scratchbox-devkit-debian
```

Otherwise you need to obtain the tarball package (see [6]) and extract it to the '/' directory:

```
# tar xfz scratchbox-devkit-debian-1.0-i386.tar.gz -C /
```

After the devkit is installed, a target that uses Debian devkit must be created (or an existing target must be reconfigured). You can follow the instructions in Section 2.4 but select 'debian' at the devkit selection menu.

The building of Debian packages needs to be done inside a 'fakeroot' environment. By default Scratchbox can only run its host tools inside fakeroot. In order to run target binaries in fakeroot, the libfakeroot binary needs to be installed on the target. If you used the default settings in the file installation menu (see Section 2.4), then you have installed fakeroot and you are set. Otherwise you need to go back to **sb-menu** and install fakeroot.

4.2. Testing Debian devkit

Debian devkit is tested by building some Debian packages. In this section we will build and install ncurses. It is easy to build because it does not have dependencies to other libraries than the C library and it does not require exotic build tools. Building more demanding packages is described later (see Section 4.3).

1. Select the target that you have configured to use the Debian devkit.
2. Update the package database:

   ```
   [sbox-MYTARGET: ~] > apt-get update
   ```

3. Get the ncurses source package:

   ```
   [sbox-MYTARGET: ~] > apt-get source ncurses
   ```

4. Go to the ncurses source directory:

   ```
   [sbox-MYTARGET: ~] > cd ncurses-5.4
   ```

5. Build the ncurses binary packages:

   ```
   [sbox-MYTARGET: ~/ncurses-5.4] > dpkg-buildpackage -b -rfakeroot
   ```

6. Go back to the home directory where the built Debian packages are located:

   ```
   [sbox-MYTARGET: ~/ncurses-5.4] > cd ..
   ```

7. Install the libncurses5 and ncurses-bin packages that were created:

   ```
   [sbox-MYTARGET: ~] > dpkg -i libncurses5_5.4-4_arm.deb
   [sbox-MYTARGET: ~] > dpkg -i ncurses-bin_5.4-4_arm.deb
   ```

8. Check that the 'tic' program was installed (from ncurses-bin) and that it is an ARM binary:

   ```
   [sbox-MYTARGET: ~] > file /usr/bin/tic
   /usr/bin/tic: ELF 32-bit LSB executable, ARM, version 1 (ARM), for
   GNU/Linux 2.0.0, dynamically linked (uses shared libs), stripped
   ```

9. Run it (check its version):

   ```
   [sbox-MYTARGET: ~] > tic -V
   ncurses 5.4.20040208
   ```

4.3. Installing Debian packages on the target

In the previous section we were able compile ncurses because Scratchbox provides its only explicit build dependency: debhelper (it has more build dependencies but they do not apply on the ARM architecture). Scratchbox provides only build tools and other utility programs; any libraries and related development files a package depends on must be installed on the target.

The preferred way to install packages is to use apt-get. The default is to use 'main' packages of the unstable distribution, located on the primary Debian mirror. You can change this by editing '/etc/apt/sources.list':

```
[sbox-MYTARGET: ~] > vi /etc/apt/sources.list
```

For example if you are located in Finland, you should change the server to ftp.fi.debian.org:

```
deb     ftp://ftp.fi.debian.org/debian/ unstable main
deb-src ftp://ftp.fi.debian.org/debian/ unstable main
```

If you want to use packages of all sections of Debian Sarge, your configuration could look like this:

```
deb     ftp://ftp.fi.debian.org/debian/ sarge    main contrib non-free
deb-src ftp://ftp.fi.debian.org/debian/ sarge    main contrib non-free
```

You can install arbitrary packages on an empty target, but since the Debian policy [9] does not require that packages declare dependencies on *essential* packages, you should install them first to prevent problems. Scratchbox provides the **sb-install-base-packages** command for that purpose:

```
[sbox-MYTARGET: ~] > apt-get update
[sbox-MYTARGET: ~] > sb-install-base-packages
```

It proposes to install about 45 packages. libc6 is installed over the C-library you installed using sb-menu. base-files wants to overwrite the '/etc/nsswitch.conf' file (also installed by sb-menu), which is okay.

> **Note:** sb-install-base-packages does not install bash even though it is essential, because its Debian package has known problems installing on Scratchbox. This should not be a big problem though, since Scratchbox provides bash.

We need more than just the base system in order to build complex applications. Let's install all required libraries and development files for GTK+ 2.x development:

```
[sbox-MYTARGET: ~] > fakeroot apt-get install libgtk2.0-dev
```

This time apt-get proposes to download about 62 packages.

> **Note:** You should use sbrsh to install GTK+ libraries, because their post-installation scripts are known to fail with QEMU.

> **Note:** It is not necessary to use fakeroot with Scratchbox's apt-get, but some packages check that their post-installation scripts are run as root, so it's best to use fakeroot when installing packages.

4.3.1. Creating rootstraps

Downloading all these packages might be slow depending on your Internet connection. You might also prefer to use a carefully selected, stable set of packages for your own or your team's development work. Once you have installed (and configured) the packages on the target, you can create a tarball snapshot of the target's filesystem. We call this sort of tarballs *rootstraps*. You can create a rootstrap like this:

```
[sbox-MYTARGET: ~] > tar cfz my_rootstrap.tar.gz -C /targets/MYTARGET
```

You can extract a rootstrap on a target using **sb-menu**. The rootstrap can either be selected from the local filesystem or downloaded via HTTP or FTP. If the file is located on the filesystem outside of Scratchbox, you can use the **/scratchbox/tools/bin/sb-menu** command to extract it outside Scratchbox.

4.4. Building a GUI application

Now that we have the required dependencies on the target we can cross-compile a more sophisticated example program:

1. Extract hello-world-gtk's source code from the /scratchbox/packages directory:

   ```
   [sbox-MYTARGET: ~] > tar xfz /scratchbox/packages/hello-world-gtk.tar.gz
   ```

2. Go to the created 'hello-world-gtk' directory.

3. Configure it:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > ./autogen.sh
   ```

4. Compile it:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > make
   ```

5. Verify that the ARM executable 'hello-gtk' was created:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > file hello-gtk
   hello-gtk: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux
   2.0.0, dynamically linked (uses shared libs), not stripped
   ```

6. Before the program can be run the DISPLAY environment variable should be set to point to the desired X-server:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > export DISPLAY=ipaddress:displaynumber
   ```

   where ipaddress is the address to the desired host and displaynumber is the host's X-display (typically 0).

7. The host's X-server should be configured so that connections from outside are allowed:

   ```
   # xhost +
   ```

   > **Note:** Allowing remote programs to use host's display is considered to be unsafe. This should be done only on trusted networks.

8. Now we can run the program:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > ./hello-gtk
   ```

   The program should open a dialog that shows the 'Hello World!' text and contains a 'Close' button.

Chapter 5. Scratchbox maintenance

5.1. Starting Scratchbox

If Scratchbox was installed from Debian or RPM packages, the **/etc/init.d/scratchbox-core** init script was installed and Scratchbox should start automatically when the system is rebooted. However, if you installed Scratchbox from tarballs then rebooting your machine will clear away all the mounts and binfmt_misc registrations that Scratchbox requires to work. To get your Scratchbox working again after reboot, you have to run the following command as root:

```
# /scratchbox/sbin/sbox_ctl start
```

Alternatively you can add sbox_ctl as an init script to the /etc/init.d directory and create the appropriate links at your system's runlevel directories. This procedure works on some systems:

```
# ln -s /scratchbox/sbin/sbox_ctl /etc/init.d/scratchbox-core
# /usr/sbin/update-rc.d scratchbox-core defaults
```

Refer to your system's documentation for the correct instructions.

5.2. Upgrading Scratchbox

If Scratchbox is installed on Debian system, it can be easily upgraded with following commands:

```
# apt-get update
# apt-get dist-upgrade
```

> **Note:** In case the toolchain versions are changed, the users may need to create a new target or reconfigure an existing target to use the new compiler name (see Section 2.4).

If Scratchbox is installed on a non-Debian system, you need to obtain newer packages [6] and extract them at the '/' directory.

> **Note:** Upgrading Scratchbox from tarballs will not replace old toolchains if the toolchain versions have changed. If you want to remove the old toolchains, you can do so by removing the directories from /scratchbox/compilers by hand.

5.3. Uninstalling Scratchbox

5.3.1. Uninstalling Scratchbox on Debian GNU/Linux

If Scratchbox was installed from Debian packages, all Scratchbox packages can be removed from the system with following command:

```
# apt-get remove scratchbox-libs
```

> **Note:** All Scratchbox packages depend on scratchbox-libs so removing it with apt-get should be enough.

Now the /scratchbox directory should be empty except for the 'users' directory. If you want to remove the user directories, double-check that there are no active mounts under /scratchbox (or if it's a symlink, the path it points to) with the 'mount' command. After that you can remove the user directories by hand:

```
# rm -rf /scratchbox/users
```

> **Note:** It is recommended that you copy any valuable data from the users' home directories before removing Scratchbox.

5.3.2. Uninstalling Scratchbox on other Linux Distributions

If Scratchbox was installed from tarballs, removing it is somewhat more complicated because Scratchbox mounts some directories from the host system to the users' sandbox directories. As a result of this, Scratchbox can't be removed by simply doing 'rm -r' (that would also remove files and sockets from the system's /tmp directory and break /dev if you were using udev).

You have to use sbox_ctl to stop Scratchbox before removing it or any of its user directories:

```
# /scratchbox/sbin/sbox_ctl stop
```

You can use the ’mount’ command to check if something is still mounted. After that you can remove Scratchbox with command:

```
# rm -r /scratchbox
```

> **Note:** It is recommended that you copy any valuable data from the users’ home directories before removing them.

Chapter 6. Additional information

More information and support can be obtained through the following channels:

- Scratchbox website [1].
- Scratchbox IRC-channel on freenode [10]: #scratchbox.
- Scratchbox mailing list (see[1] for more information).
- Commercial support is provided by Movial [11].

References

[1] *Scratchbox website (http://scratchbox.org/)*.

[2] *The Debian Project (http://www.debian.org/)*.

[3] *How to Run Linux on iPAQ Handhelds (http://www.handhelds.org/handhelds-faq/handhelds-faq.html)*, Jamey Hicks.

[4] *The Familiar Project (http://familiar.handhelds.org/)*.

[5] *Scratchbox toolchains (http://scratchbox.org/documentation/docbook/toolchain.html)*, Ricardo Kekki.

[6] *Scratchbox download page (http://scratchbox.org/download/)*.

[7] *Linux NFS-HOWTO (http://www.tldp.org/HOWTO/NFS-HOWTO/index.html)*, Tavis Barr, Nicolai Langfeldt, Seth Vidal, Tom McNeal.

[8] *Scratchbox Remote Shell (http://scratchbox.org/documentation/docbook/sbrsh.html)*, Timo Savola.

[9] *Debian Policy Manual (http://www.debian.org/doc/debian-policy/)*.

[10] *freenode IRC network (http://freenode.net/)*.

[11] *Movial website (http://www.movial.fi/en/products/Scratchbox/)*.

Appendix A. Scratchbox environment variables

This is a list of Scratchbox specific environment variables that can be set by the user. Some of them are already set in /scratchbox/etc/profile.

```
SBOX_REDIRECT_FROM_DIRS
SBOX_REDIRECT_TO_DIRS
        colon-separated list of directories to be used as the
        sources/destinations of binary redirection

SBOX_REDIRECT_BINARIES
        comma-separated list of <source>:<target> pairs which specify
        explicit binary redirection rules (for example
        "/usr/bin/make:/scratchbox/tools/bin/make,/bin/cat:/host_usr/bin/dog")

SBOX_REDIRECT_IGNORE
        colon-separated list of binaries which should not be redirected

SBOX_REDIRECT_LOG
        specifies the filename where each binary redirection will be logged;
        logging will be disabled if this is not set

SBOX_UNAME_SYSNAME
SBOX_UNAME_NODENAME
SBOX_UNAME_RELEASE
SBOX_UNAME_VERSION
SBOX_UNAME_MACHINE
        override an uname field; see uname(1) or uname(2) manpage

SBOX_DISABLE_CPUTRANSPARENCY
        "yes" or "no" (CPU-transparency is enabled by default)

SBOX_CPUTRANSPARENCY_LOG
        specifies the filename where each CPU-transparency invocation will be
        logged; logging will be disabled if this is not set

SBOX_CPUTRANSPARENCY_VERBOSE
        if enabled, a notice is printed to stderr when execution jumps to
        target system; "yes" or "no"

SBOX_CPUTRANSPARENCY_NOWARN
        do not issue a warning when trying to execute a binary that is under
        /tmp with sbrsh; "yes" or "no"

SBOX_CPUTRANSPARENCY_METHOD
        overrides the SBOX_CPUTRANSPARENCY_METHOD field in scratchbox.config,
        which contains the path to the actual target binary interpreter;
        good choices are "sbrsh", "qemu-arm", "qemu-ppc", etc.

SBOX_CPUTRANSPARENCY_FILTER
```

path to an executable (that is compiled for the target) that gets run whenever a command needs to be run on the target; the executable receives the command and the command's arguments as its arguments; this can be a shell script if a shell is compiled for the target system

SBOX_SCRATCHBOX_CONFIG
overrides the location of /targets/links/scratchbox.config, used by the compiler wrapper

SBOX_DEFAULT_GCC_PREFIX
overrides the SBOX_DEFAULT_GCC_PREFIX field in scratchbox.config, used by the compiler wrapper

SBOX_LD_FAKE_NATIVE
if set, the linker wrapper uses ld_fake_native instead of ld_orig

SBOX_USE_CCACHE
should the compiler wrapper automatically use ccache? "yes" or "no"

SBOX_EXTRA_ARGS
passes additional parameters to all programs executed via the compiler wrapper (see below for a list of affected programs)

SBOX_BLOCK_ARGS
suppresses selected parameters of all programs executed via the compiler wrapper (see below for a list of affected programs)

SBOX_EXTRA_COMPILER_ARGS SBOX_BLOCK_COMPILER_ARGS
affects only compilers; overrides the more general form

SBOX_EXTRA_CC_ARGS SBOX_BLOCK_CC_ARGS
SBOX_EXTRA_CXX_ARGS SBOX_BLOCK_CXX_ARGS
SBOX_EXTRA_CPP_ARGS SBOX_BLOCK_CPP_ARGS
affects only specific compiler type; overrides the more general forms

SBOX_EXTRA_LD_ARGS SBOX_BLOCK_LD_ARGS
affects only the linker; overrides the more general form

SBOX_EXTRA_ADDR2LINE_ARGS SBOX_BLOCK_ADDR2LINE_ARGS
SBOX_EXTRA_AR_ARGS SBOX_BLOCK_AR_ARGS
SBOX_EXTRA_AS_ARGS SBOX_BLOCK_AS_ARGS
SBOX_EXTRA_CXXFILT_ARGS SBOX_BLOCK_CXXFILT_ARGS
SBOX_EXTRA_GCCBUG_ARGS SBOX_BLOCK_GCCBUG_ARGS
SBOX_EXTRA_GCOV_ARGS SBOX_BLOCK_GCOV_ARGS
SBOX_EXTRA_NM_ARGS SBOX_BLOCK_NM_ARGS
SBOX_EXTRA_OBJCOPY_ARGS SBOX_BLOCK_OBJCOPY_ARGS
SBOX_EXTRA_OBJDUMP_ARGS SBOX_BLOCK_OBJDUMP_ARGS
SBOX_EXTRA_RANLIB_ARGS SBOX_BLOCK_RANLIB_ARGS
SBOX_EXTRA_READELF_ARGS SBOX_BLOCK_READELF_ARGS
SBOX_EXTRA_SIZE_ARGS SBOX_BLOCK_SIZE_ARGS
SBOX_EXTRA_STRINGS_ARGS SBOX_BLOCK_STRINGS_ARGS
SBOX_EXTRA_STRIP_ARGS SBOX_BLOCK_STRIP_ARGS
affects only a specific utility; overrides the more general form

SBOX_BLOCK_<program>
blocks the execution of a program completely; the value can be either "yes" (normal return), "no" or a return code that will be used when blocking the program; <program> is one of the programs used with the *_ARGS forms (except "COMPILER")

SBOX_ENV_<var>
sets <var> to the environments of commands executed with sbrsh and qemu; "(UNSET)" removes <var> from the environment

SBOX_RLIMIT_<key>
sets a resource limit before executing commands with CPU-transparency; see sbrsh/README or setrlimit(2) manpage for details

SBOX_DEFAULT_AUTOMAKE
Specifies which automake version the 'automake' command points to. Available versions are "1.4", "1.7" and "1.8" (default).

SBOX_DEFAULT_AUTOCONF
Specifies which autoconf version the 'autoconf' command points to. Available versions are "2.13" and "2.59" (you can also specify "2.50" which is an alias for 2.59). The default is to automatically guess the appropriate version, so you should not set this variable under normal circumstances.

SBOX_DEFAULT_FLEX
Specifies which flex version the 'flex' command points to. Available versions are "0.4.5" and "0.4.31" (default). "old" is an alias for "0.4.5".

This is a list of Debian devkit specific environment variables that can be changed by the user. Default values are set in /scratchbox/devkits/debian/etc/environment.

SBOX_DPKG_INST_ARCH
Specifies the architecture that dpkg will use. Set automatically by the environment script if the selected target uses a non-x86 toolchain.

SBOX_DPKG_BUILDDEPS
If set to "yes", dpkg-checkbuilddeps will consider the tools provided by Scratchbox as valid build dependencies.