MOVIAL

Scratchbox Release Test Suite

Jussi Hakala

26th April 2006

Contents

1 Introduction **1**

2 Tests **1**

2.1 Basic tests 1

2.2 Advanced tests 4

2.3 More advanced tests 6

3 Tasks **8**

3.1 Installing Scratchbox 8

3.2 Operations for users 10

3.3 Operations for target 11

3.4 Compilation 14

3.5 Execution 15

Revision history

Version	Author	Description
2006-03-27	Jussi Hakala	First draft
2006-03-28	Jussi Hakala	Numerized test steps, added a new test, other minor improvements
2006-03-30	Jussi Hakala	Small corrections in the two first more advanced tests
2006-04-05	Jussi Hakala	Minor fixes
2006-04-10	Jussi Hakala	Moved toolchain specific information to Release test plan
2006-04-18	Jussi Hakala	Modified Basic test 3 to use Scratchbox installed in a custom location
2006-04-26	Timo Savola	Toolchain terminology disambiguated, minor changes and fixes

1 Introduction

For every release of Scratchbox, there is a number of tests to be performed before releasing it to the public. The first part of these tests include installing and configuring Scratchbox in a various platforms described by the *Scratchbox Release Test Plan* and using Scratchbox to compile software to a different architecture from the host architecture. The second part of the tests is testing of the compiled binaries using emulated and/or actual target architectures. Additionally, for each new feature of a Scratchbox release, a test is defined in the Scratchbox Release Test Plan to verify the function of that particular feature.

The tests are made of individual tasks. In a test, individual tasks form a path in along which the test progresses. This way, every operation in every test is discrete and the test can be used as a measurement of the overall function of the Scratchbox release.

There are both automatic and manual tests. Most of the tests described in this document can be executed either automatically or manually. In the primary platform, all tests are run manually. On other platforms, only a set of tests is selected and executed automatically. For information about platforms in which the tests are executed, please refer to Scratchbox Release Test Plan. The detailed results of the tests executed in different platforms can be found in the *Scratchbox Test Reports*.

2 Tests

2.1 Basic tests

Test 2.1.1 *Basic test 1*

Explanation: Installation of Scratchbox, create a target using arbitrary toolchain and test the target created environment
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using any toolchain [task 3.3.1]
5. Select the newly created target [task 3.3.2]
6. Check the target environment sanity [task 3.3.9]
7. Logout [task 3.2.3]

Test 2.1.2 *Basic test 2*

Explanation: Installation of Scratchbox, creation of a new target using a non-native toolchain and removing devkits from an existing target
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using any non-native toolchain and install all available devkits [task 3.3.1]
5. Remove a devkit from the target [task 3.3.4]
6. Logout [task 3.2.3]

Test 2.1.3 *Basic test 3*

Explanation: Installation of Scratchbox to a custom location, creation of a new target using a native toolchain and adding devkits to an existing target
Test steps:

1. Install Scratchbox to a custom location [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using a native toolchain and none of the available devkits [task 3.3.1]
5. Add a devkit to the target [task 3.3.3]
6. Logout [task 3.2.3]

Test 2.1.4 *Basic test 4*

Explanation: Upgrading the Scratchbox from a previous version, modifying of the existing targets from previous version, creating and modifying a new target and handling concurrent logins from users created with both versions of Scratchbox
Test steps:

1. Upgrade Scratchbox [task 3.1.2]
2. Create a new user [task 3.2.1]
3. Open a new terminal and login to Scratchbox with the user from the previous version [task 3.2.2]
4. Select a target from the old installation [task 3.3.2]
5. Install all available devkits to the target [task 3.3.3]
6. Open a new terminal and login to Scratchbox with the newly created user [task 3.2.2]
7. Create a target using any toolchain and install none of the available devkits [task 3.3.1]

8. Select the target [task 3.3.2]
9. With the user from the previous version, change the toolchain of the target [task 3.3.5]
10. With the user from the previous version, remove a devkit from the target [task 3.3.4]
11. With the user from the new version, change the toolchain of the target [task 3.3.5]
12. With the user from the new version, add a devkit to the target [task 3.3.4]
13. With the user from the previous version, logout [task 3.2.3]
14. With the user from the new installation, logout [task 3.2.3]

Test 2.1.5 *Basic test 5*

Explanation: Reinstalling Scratchbox, modifying targets from the previous installation, creating and modifying a new target and handling concurrent logins from users created with both installations of Scratchbox
Test steps:

1. Reinstall Scratchbox [task 3.1.3]
2. Create a new user [task 3.2.1]
3. Open a new terminal and login to Scratchbox with the user from the old installation [task 3.2.2]
4. Select a target from the old installation [task 3.3.2]
5. Remove a devkit from a target [task 3.3.4]
6. Open a new terminal and login to Scratchbox with the newly created user [task 3.2.2]
7. Create a target using any toolchain and install all the available devkits [task 3.3.1]
8. With the user from the old installation, remove an existing target [task 3.3.8]
9. With the user from the old installation, create a target with a same name than the removed one, using any toolchain other than the removed one was using and install all the available devkits [task 3.3.1]
10. With the user from the new installation, change the toolchain of the target [task 3.3.5]
11. With the user from the old installation, logout [task 3.2.3]
12. With the user from the new installation, logout [task 3.2.3]

Test 2.1.6 *Basic test 6*

Explanation: Installation of Scratchbox paraller to another version of Scratchbox running in the same host, installation and configuration of Maemo SDK using a native toolchain
Test steps:

1. Install Scratchbox with the previous version shutdown [task 3.1.4]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using appropriate toolchain and all the available devkits [task 3.3.1]
5. Install Maemo SDK [task 3.3.10]
6. Logout [task 3.2.3]

Test 2.1.7 *Basic test 7*

Explanation: Installation paraller to another version of Scratchbox running in the same host, installation and configuration of Maemo SDK using a toolchain suitable for target architecture and QEMU as the CPU transparency method
Test steps:

1. Install Scratchbox with the previous version running [task 3.1.4]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using a target suitable toolchain, QEMU as the CPU transparency method and install all the available devkits [task 3.3.1]
5. Install Maemo SDK [task 3.3.10]
6. Logout [task 3.2.3]

2.2 Advanced tests

Test 2.2.1 *Advanced test 1*

Explanation: Installation of Scratchbox and compiliation of essential packages for a Debian environment
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using any non-native toolchain, with all available devkits [task 3.3.1]

5. Compile essential packages for the Debian environment using Crocodile [task 3.4.2]
6. Logout [task 3.2.3]

Test 2.2.2 *Advanced test 2*

Explanation: Installation of Scratchbox and Maemo SDK, compilation of a small Maemo application for host architecture
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using a native toolchain, with all available devkits [task 3.3.1]
5. Compile a small Maemo application for the host architecture using documentation from maemo.org as a reference [task 3.4.3]
6. Logout [task 3.2.3]

Test 2.2.3 *Advanced test 3*

Explanation: Installation of Scratchbox and Maemo SDK, compilation of a small Maemo application for target architecture
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using a suitable toolchain for the target architecture, with all available devkits, using QEMU as the CPU transparency method [task 3.3.1]
5. Compile a small Maemo application for the target environment using documentation from maemo.org as a reference [task 3.4.3]
6. Logout [task 3.2.3]

Test 2.2.4 *Advanced test 4*

Explanation: Installation of Scratchbox, 2 concurrent logins with the same user, simultaneous compliation of hello world and essential packages for Debian environment
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]

3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target [task 3.3.1]
5. Select the target [task 3.3.2]
6. Open new terminal and login to Scratchbox with the same user [task 3.2.2]
7. Compile essential packages for the Debian environment using Crocodile in the new shell [task 3.4.2]
8. Compile hello world in the old shell [task 3.4.1]
9. Logout from the old shell [task 3.2.3]
10. Logout from the new shell [task 3.2.3]

2.3 More advanced tests

Test 2.3.1 *More advanced test 1*

Explanation: Installing Scratchbox, compiling hello world to a target architecture, executing the compiled binary using QEMU
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using any non-native toolchain, with all available devkits, using QEMU as the CPU transparency method [task 3.3.1]
5. Compile hello world [task 3.4.1]
6. Execute hello world [task 3.5.1]
7. Logout [task 3.2.3]

Test 2.3.2 *More advanced test 2*

Explanation: Installing Scratchbox, compiling hello world to a target architecture, executing the compiled binary using sbrsh
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using any non-native toolchain, with all available devkits, using sbrsh as the CPU transparency method [task 3.3.1]

5. Compile hello world [task 3.4.1]
6. Execute hello world [task 3.5.2]
7. Logout [task 3.2.3]

Test 2.3.3 *More advanced test 3*

Explanation: Installation of Scratchbox, compiling essential packages for a Debian environment for the host architecture using Crocodile, chrooting to rootfs and testing if the environment is ok
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using a native toolchain and with all available devkits [task 3.3.1]
5. Compile essential packages for the Debian environment using Crocodile [task 3.4.2]
6. Chroot into the compiled environment to verify that it is functioning [task 3.5.3
7. Logout [task 3.2.3]

Test 2.3.4 *More advanced test 4*

Explanation: Installation of Scratchbox, compiling essential packages for a Debian environment for a target architecture using Crocodile, mounting rootfs on a device with target architecture using nfs and testing if the environment is ok
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using a suitable toolchain for the target architecture, with all the available devkits [task 3.3.1]
5. Compile essential packages for the Debian environment using Crocodile [task 3.4.2]
6. Mount the rootfs in a machine with target architecture [task 3.5.4]
7. Logout [task 3.2.3]

Test 2.3.5 *More advanced test 5*

Explanation: Installation of Scratchbox and Maemo SDK, compiling a small Maemo application for host architecture and execution of the compiled application
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using a native toolchain and with all available devkits [task 3.3.1]
5. Compile a small Maemo application for the target environment using documentation from maemo.org as a reference [task 3.4.3]
6. Execute compiled Maemo application [task 3.5.5]
7. Logout [task 3.2.3]

Test 2.3.6 *More advanced test 6*

Explanation: Installation of Scratchbox and Maemo SDK, compiling a small Maemo application for a target architecture and execution of the compiled application
Test steps:

1. Install Scratchbox [task 3.1.1]
2. Create a new user [task 3.2.1]
3. Login to Scratchbox with newly created user [task 3.2.2]
4. Create a new target using a suitable toolchain for the target architecture, with all the available devkits and using QEMU as the CPU transparency method [task 3.3.1]
5. Compile a small Maemo application for the target environment using documentation from maemo.org as a reference [task 3.4.3]
6. Execute compiled Maemo application [task 3.5.7]
7. Logout [task 3.2.3]

3 Tasks

3.1 Installing Scratchbox

Task 3.1.1 *Install*

Explanation: Install Scratchbox
Requirements: none
Task steps:

1. Install all the Scratchbox release packages (see install documentation 2.1)
2. Configure and start Scratchbox (see install documentation 2.2.2)

How to determine if the task was passed:

- Create a new user [task 3.2.1]
- Login to Scratchbox with the created user [task 3.2.2]

Task 3.1.2 *Upgrade*

Explanation: Upgrade Scratchbox from previous version
Requirements: none
Task steps:

1. Make sure that the previous installation is properly shut down
2. Install all Scratchbox release packages (see install documentation 2.1 and 5.2)

How to determine if the task was passed:

- Directories created by users in the previous installation still exist in the new one
- Create a new user [task 3.2.1]
- Login to Scratchbox with the created user [task 3.2.2]

Task 3.1.3 *Reinstall*

Explanation: Reinstall current version of Scratchbox
Requirements: none
Task steps:

1. Make sure that the previous installation is properly shut down
2. Install all Scratchbox release packages (see install documentation 2.1)
3. Configure and start Scratchbox (see install documentation 2.2.2)

How to determine if the task was passed:

- Directories created by users in the previous installation still exist in the new one
- Create a new user [task 3.2.1]
- Login to Scratchbox with the created user [task 3.2.2]

Task 3.1.4 *Install next to a different version*

Explanation: Install Scratchbox to a custom location in a host where there is already an older version of Scratchbox running in the default location
Requirements: none
Task steps:

1. Install all Scratchbox release packages to `/opt/scratchbox-version` (see install documentation 2.1)
2. Configure and start Scratchbox (see install documentation 2.2.2)

How to determine if the task was passed:

- Create a new user [task 3.2.1]
- Login to new installation of Scratchbox with the created user [task 3.2.2]
- Login to old installation of Scratchbox with any user

3.2 Operations for users

Task 3.2.1 *Create a user*

Explanation: Create a user
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4
Task steps:

1. If the user you're creating doesn't exist in the system, create it
2. Create a Scratchbox user for the user with a user account in the system (see install documentation 2.2.2)

How to determine if the task was passed:

- Login to Scratchbox with the newly created user

Task 3.2.2 *Login*

Explanation: Login to Scratchbox
Requirements: 3.1.1 or 3.1.4
and 3.2.1 or 3.1.2 or 3.1.3 Task steps:

1. Login to Scratchbox (see install documentation 2.3)

How to determine if the task was passed:

- A welcome message is displayed

Task 3.2.3 *Logout*

Explanation: Logout from Scratchbox
Requirements: 3.1.1 or 3.1.4
and 3.2.1 or 3.1.2 or 3.1.3 Task steps:

1. Logout from Scratchbox

How to determine if the task was passed:

- You are returned to normal shell

3.3 Operations for target

Task 3.3.1 *Create a new target*

Explanation: Create a new target
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1
Task steps:

1. Create the target (see install documentation 2.4)
2. Select a toolchain for the target (see install documentation 2.4)
3. Install devkits to the target (see install documentation 2.4)

How to determine if the task was passed:

- No errors were reported by the sb-conf or the sb-menu

Task 3.3.2 *Select a target*

Explanation: Select a target
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Select the target

How to determine if the task was passed:

- The name of the target is displayed in the Scratchbox prompt

Task 3.3.3 *Install a devkit to a target*

Explanation: Modify a target
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Install devkits to the target (see install documentation 2.4)
2. Make sure the mutual dependencies of the devkits are handled correctly

How to determine if the task was passed:

- No errors were reported by the sb-conf or the sb-menu
- All things provided by the devkit(s) are present

Task 3.3.4 *Remove a devkit from a target*

Explanation: Modify a target
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Remove devkits from the target (see install documentation 2.4)
2. Make sure the mutual dependencies of the devkits are handled correctly

How to determine if the task was passed:

- No errors were reported by the sb-conf or the sb-menu
- Things provided by the removed devkit(s) are no longer present

Task 3.3.5 *Choose toolchain of a target*

Explanation: Modify a target
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Install devkits to the target (see install documentation 2.4)

How to determine if the task was passed:

- No errors were reported by the sb-conf or the sb-menu

Task 3.3.6 *Choose CPU transparency method of a target*

Explanation: Modify a target
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Choose CPU transparency method of the target (see install documentation 2.4)

How to determine if the task was passed:

- No errors were reported by the sb-conf or the sb-menu

Task 3.3.7 *Reset a target*

Explanation: Reset a target
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Reset the target

How to determine if the task was passed:

- Verify that the target files are removed from the target's directory

Task 3.3.8 *Remove a target*

Explanation: Delete a target
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Delete the target

How to determine if the task was passed:

- Verify that the target's directory doesn't exist
- Verify that the target's configuration file doesn't exist
- Verify that the target's sbrsh configuration file doesn't exist

Task 3.3.9 *Check a target*

Explanation: Run a test suite on target environment
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Test fakeroot function on the host and the target side
2. Try some primitive floating point operations
3. Create a generic library to test the linking
4. Check that error codes in library calls are passed correctly
5. Make sure you can create threads

How to determine if the task was passed:

- The test suite which run these tests returns no errors

Task 3.3.10 *Install Maemo SDK*

Explanation: Install Maemo SDK and all the required components to compile Maemo applications using maemo.org as a reference
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Create a target or select an existing one
2. Select a suitable toolchain for your target and install all available devkits
3. Install Maemo SDK (see maemo.org install tutorial)

How to determine if the task was passed:

- Execute Maemo hello world application

3.4 Compilation

Task 3.4.1 *Compile hello world*

Explanation: Compile hello world and test the compiled binary without executing it
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Compile hello world C and C++ sources
2. Check that the result binaries are of the correct type
3. Strip the result binaries
4. Check library dependencies of the result binaries

How to determine if the task was passed:

- No errors returned by the compiler
- The binaries are of the correct type when compared to the toolchain used to compile them
- The binaries can be stripped without errors
- All the libraries required for the binaries are present

Task 3.4.2 *Compile Crocodile*

Explanation: Compile Debian essentials using Crocodile
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1
Task steps:

1. Compile packages essential for a Debian environment using Crocodile

How to determine if the task was passed:

- No errors returned by the Crocodile

Task 3.4.3 *Compile Maemo application*

Explanation: Compile a simple Maemo application using maemo.org as a reference
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1 and 3.3.10
Task steps:

1. Compile an arbitrary Maemo application

How to determine if the task was passed:

- No errors returned by the compiler

3.5 Execution

Task 3.5.1 *Execute hello world under QEMU*

Explanation: Execute hello world using QEMU as CPU transparency method
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1 and 3.4.1
Task steps:

1. Execute hello world

How to determine if the task was passed:

- Hello world! is displayed in the screen

Task 3.5.2 *Execute hello world*

Explanation: Execute hello world using sbrsh as CPU transparency method
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1 and 3.4.1
Task steps:

1. Execute hello world

How to determine if the task was passed:

- Hello world! is displayed in the screen

Task 3.5.3 *Test the Debian rootfs in host architecture*

Explanation: Chroot to rootfs and run misc tests to determine if it is working
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1 and 3.4.2
Task steps:

1. Chroot to rootfs and do some misc testing

How to determine if the task was passed:

- There is no unusual anomalies during the test process

Task 3.5.4 *Test the Debian rootfs in target architecture*

Explanation: Mount rootfs as nfs to actual target architecture and run misc tests to determine wheter it is working or not
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1 and 3.4.2
Task steps:

1. Mount exported rootfs on a target architecture device
2. Do some misc testing

How to determine if the task was passed:

- There is no unusual anomalies during the test process

Task 3.5.5 *Execute Maemo application in host architecture*

Explanation:
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1 and 3.3.10 and 3.4.3
Task steps:

1. Execute a Maemo application on a host architecture

How to determine if the task was passed:

- The application works as to be expected and no errors are reported by the system

Task 3.5.6 *Execute Maemo application under QEMU*

Explanation:
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1 and 3.3.10 and 3.4.3
Task steps:

1. Execute a Maemo application on a host architecture using QEMU

How to determine if the task was passed:

- The application works as to be expected and no errors are reported by th e system

Task 3.5.7 *Execute Maemo application in target architecture*

Explanation:
Requirements: 3.1.1 or 3.1.2 or 3.1.3 or 3.1.4 and 3.2.1 and 3.3.1 and 3.3.10 and 3.4.3
Task steps:

1. Execute a Maemo application on actual target architecture

How to determine if the task was passed:

- The application works as to be expected and no errors are reported by th e system