# Integration with IDE tools

**Valtteri Rahkonen**

**valtteri.rahkonen@movial.fi**

# Integration with IDE tools

by Valtteri Rahkonen

Revision history

| Version: | Author: | Description: |
|---|---|---|
| 2005-02-04 | Savola | Updates |
| 2005-01-10 | Rahkonen | Initial version |

# Table of Contents

# Chapter 1. Introduction

It would be really nice if people could just install Scratchbox [1] to their machines, install required target device(s), check “Scratchbox cross-compilation and testing support” checkbox in their Eclipse, KDevelop or Anjuta Integrated Development Environments and then re-compile and test their software as usual, but for different CPU architecture than the one in their own workstation. However this is not yet possible, because of lack of support (or plug-ins) for IDE’s.

This documents purpose is to describe requirements for integrating Scratchbox to an IDE. We also describe how these tasks could be implemented and thus with this documents instructions you will be able to use your favourite IDE and cross-compile, test and debug your cross-compiled applications at least in somewhat automated manner.

# Chapter 2. IDE integration

For integrating cross-compiling to your preferred IDE you need to be able to cross-compile applications, test them and of course run them inside debugger. Following subsections will show how these tasks can be achieved.

## 2.1. Running commands in Scratchbox

For integrating Scratchbox to your preferred IDE, first requirement is that you need to be able to execute commands inside Scratchbox from your IDE. You need to be able to do this, because otherwise you would just use your hosts normal build tools and compile it for your own host machine CPU architecture. So you need to be able to execute those configure scripts and cross-compile your program inside Scratchbox to enable development to different CPU architecture.

Scratchbox already provides methods for executing commands inside currently selected target. For this you can use same script that you use to login yourself to Scratchbox: **/scratchbox/login**. Of course if you want to cross-compile something inside Scratchbox, you need to keep your sources where you can access them from Scratchbox. Later on we assume that you keep your sources inside under your Scratchbox's home directory: /scratchbox/users/<username>/home/<username>.

If you want to compile program that uses autotools build system you can do it with following steps:

1. Select your preferred target that is created for your target CPU architecture. You can do that with following command from your host:

   ```
   /scratchbox/tools/bin/sb-conf select TARGETNAME
   ```

2. Obtain source directory to your Scratchbox's home directory. We assume later on that they are located at source-dir directory.
3. Use configure script to find necessary libraries and to create makefiles:

   ```
   /scratchbox/login -d $HOME/source-dir ./configure
   ```

4. Cross-compile your program:

   ```
   /scratchbox/login -d $HOME/source-dir make
   ```

You have now cross-compiled your software to different CPU architecture than your own.

**Note:** If you have installed Scratchbox from Debian or RPM packages, you can use the **/usr/bin/scratchbox** and **/usr/bin/sb-conf** commands instead of /scratchbox/login and /scratchbox/tools/bin/sb-conf.

## 2.2. Running and testing the cross-compiled programs

Command line programs can be run just by using **/scratchbox/login** (like we executed configure script and make in previous section), but X programs need an X server. For this the you can just run Xnest (or Xephyr) outside the Scratchbox:

```
Xnest :1 -ac -geometry WIDTHxHEIGHT &
```

And then export a suitable DISPLAY variable for the target binaries:

```
DISPLAY=HOST-IP:1 /scratchbox/login -d $HOME/source-dir ./prog
```

If you need to export more environment variables and they do not vary much you can always place them to a script that in your Scratchbox's home directory (named as run.sh for example). Script can something like this:

```
#!/bin/sh -l
# Go to directory given as first arg and execute rest of the args there
cd $1
shift

# Export environment variables required by the command to execute
export DISPLAY=HOST-IP:1

# Run command
$*
```

It's user specific so that users can (through the IDE or manually) set their own environment variables etc. After you have created that script you can issue commands with it:

```
/scratchbox/login ./run.sh $HOME/source-dir ./prog
```

## 2.3. Running the cross-compiled programs in a debugger

If you want to debug a program that you are running on a target device, you need to use either a gdb or gdbserver compiled for that target platform. Gdbserver doesn't need any significant amount of memory (unlike gdb) so its use is desirable.

In Scratchbox there are gdbserver's compiled for the various target CPU architecture (found under /scratchbox/device_tools/gdb-6.1 directory). In our example we assume that you have copied the gdbserver for your target CPU architecture to you Scratchbox's home directory and renamed it to for example arm-gdbserver (if your are developing for ARM) for simpler usage. Now you can execute the gdbserver with **/scratchbox/login** (inside Scratchbox target binaries are run automatically on the target when invoked):

```
/scratchbox/login -d $HOME/source-dir $HOME/arm-gdbserver HOST-IP:PORT prog
```

Where HOST-IP is the IP-number of your workstation and PORT is the port number on target you want gdbserver to use for network communication.

Then you can use e.g. DDD (see 2) that is installed outside Scratchbox to run a GDB (installed inside Scratchbox):

```
ddd --debugger '/scratchbox/login -d $HOME/source-dir gdb prog'
```

And in DDD prompt tell GDB to contact and control the arm-gdbserver running on the target:

```
(gdb) target remote HOST-IP:PORT
(gdb) break main
(gdb) cont
```

(After gdbserver has stopped the arm binary at the 'main' breakpoint, gdb should have loaded the symbols for the arm binary so that you see them in DDD.)

The reason why GDB has to be compiled for target CPU architecture and run inside the Scratchbox is that otherwise it wouldn't load the correct libraries etc. for the program you're debugging. Paths to them are different outside the Scratchbox.

# Chapter 3. Conclusions

If you want to cross-compile applications to different CPU architecture than your own host’s and use your favourite IDE to do so, you need to be able to integrate your cross-compiling environment as described in previous section. Of course these tasks might not be able to function directly from your IDE unless there is appropriate plug-in created for these (if you cannot change for example debugger starting sequence). At least for easier deployment to your development support or plug-ins for cross-compiling would be preferred.

However with this document’s instructions you will be able to use your IDE and cross-compile, test and debug your software for various different architectures with Scratchbox.

## References

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

[2] *Data Display Debugger - GNU Project (http://www.gnu.org/software/ddd/)* .