# DROPS Building Infrastructure Proposal

Jork Löser — May 2002

## Abstract

To ease the writing of DROPS components, we wrote a document defining the unified infrastructure of the DROPS source tree. We implemented a set of macros and some tools providing the described functionality.

## 1 History

In 2000, some people (see Section 4) began to write down their expectations to a build system and defined its behavior. A macro-system implementing this behavior was implemented and is ready to use since then [2]. However, there were some features missing, such as support for an easy IDL compilation. Since then, the macros were modified and adapted a little bit, but they are still unused.

Within the last two years, the structure of the building system changed and today, we have a quite clean source tree with a lot of unwritten rules. Hence, writing a Makefile still bears some uncertainties. Users fall back to copy-and-paste usually, accompanied by praying or just hoping depending on personal taste.

We think, the macro-system written two years ago is not used for two reasons: Firstly, it was not well documented. There is an ASCII file describing some of the magic, but there is not everything in it. Secondly, the availability of the macro-system was not communicated. Especially for students, it should have been a big help: Writing 5 lines and getting every functionality we ever thought about should be much easier than reinventing all that stuff over and over again.

To sum it up, we give it another try. This time, there is a spec [1], 29 pages currently. It is comprising as there is nearly every functionality in it our DROPS system currently offers (and a bit more). We communicate that there is help for writing Makefiles. Finally, migrating to the proposed system can be done on a directory-by-directory base, placing no burden for keeping old code.

## 2 Features

### 2.1 Roles

Each directory in the source-tree has a distinct task, i.e., each directory builds special kinds of targets. This disctinct task is expressed by a role, while each role is one of the following:

| | |
|---|---|
| subdir: | be a container for other directories |
| prog: | build executable binaries |
| lib: | build libraries |
| idl: | translate IDL definitions into code |
| include: | hold include-files which have to be installed |

To define the role of a directory, the Makefile of that directory has to include an according Makefile-include, the *role-file*. The role-files are templates that define rules and variables to build the proper targets. Make-variables are used to control the behaviour of the role-files. Each Makefile must include exactly one role-file.

As usual, defining the Make-variables to control the role-files can happen in 4 different places: The Makefile, a local `Makeconf.local`, a packet-local `Makeconf.local`, and a global `Makeconf.local`.

### 2.2 Provided Make-Targets

With all roles, the following phony targets can be built using `make <target>`.

**all::** The default target. Builds binaries, libraries, generating .c and .h files from IDLs. It depends on the role, what exactly is done.

{**txt, old,**}**config::** Run the DROPS Configuration Tool. See Section 2.3 for details on this.

**install::** Installs the generated files into the installation-tree `$(DROPS_STDDIR)`.

**clean::** Deletes all intermediate files generated during compilation. This does *not* delete the generated targets such as binaries or libraries.

**cleanall::** Deletes all generated files. Depends on **clean::**.

**help::** Displays a short overview of the make-targets available with the role.

## 2.3 DROPS Configuration Tool

Most role-files provide support for an interactive configuration tool, which is derived from the *config* and *menuconfig* tools of Linux. It differs in that it can be configured using environment variables. Some minor extensions are build in too.

The configuration tool is run if the user runs "`make config`", "`make txtconfig`" or "`make oldconfig`".

As within Linux, the configuration tool creates two files: One to be included within the Makefile itself. This file also stores the configuration. The other generated file is an include file to be included in C source files. It is ensured that a default configuration file and a default configuration include file is created prior to compiling other source files. Do we have to mention that you can define every filename according to your personal taste, even `__many_underscores__in_name___`?[1]

## 2.4 Target-specific variables

With all building roles, there can be multiple targets. This is, you can build multiple binaries or libraries within one directory. This requires configuration variables to be interpreted on a per-target base. For an example, we look at the `SRC_C` variable. With the prog role, this variable defines which .c-files go into a binary. If you want the two binaries **A** and **B** to be created in the directory, with **A** having `a.c` and `x.c` compiled in, and **B** having `b.c` and `x.c` compiled in, specify `SRC_C=x.c`, `SRC_C_A=a.c` and `SRC_C_B=b.c`. Further specify `TARGET=A B`, and that's it.

[1] Though we do not recommend this example

## 2.5 Dependencies

Dependencies are built automatically. Each source-file is accompanied by an according `.x.d` file with x being replaced by the original filename. Typically these files contain header files. Also, each binary is accompanied by an according `.y.d` file with y being replaced by the binary filename. These files typically contain libraries. Dependency files are created automatically and they are used automatically. Removing a dependency (e.g., an include-file), does not harm the build process. If this is not the case, the build-system has a bug.

## 2.6 Relocation of Binaries

We support two flavors of finally linking: Relocating a formerly relocatable binary or build relocated binaries in a normal link process. With the former method, all binaries a built relocatable first. These relocatable binaries have all symbols resolved and all required libraries linked, but the final start-address is not yet defined. In a second step, invoked with `make reloc`, a program determines the memory ranges required by each binary and links them to different but contiguous locations.

The other method is the approach using the STATIC file. A default link address must be specified in the Makefile.

## 2.7 IDL files

The purpose of the IDL role is to translate IDL definition files into appropriate .c- and .h-files. Different compilers are supported and the user needs not to take care of the resulting filenames. To compile the generated files, directories with the prog or lib role are used. These import the generated files using IDL-related parameters.

## 2.8 Profiling and Shared libraries

Compiling profiling versions of libraries and binaries is easy: The user just has to specify one target-specific flag: BUILD_PROFILE. As a result, profiling versions of objects, libraries and binaries are built additionally to the normal ones.

Building shared libraries is easy too, the build-parameter is BUILD_SHARED resp. BUILD_PIC.

### 2.9 Installation

When includes, libraries or binaries are built, they are installed into a *local installation directory*. This local installation directory is a parameter of the according role-file and is `$(L4DIR)/include` for includes, `$(L4DIR)/bin` for binaries and `$(L4DIR)/lib` for libraries per default. The commands used for installation can be selected as well. Includes and libraries use symbolic links, binaries are copied and stripped.

Using `make install` installs includes, libraries and binaries into the *global installation directory*, aka `/home/drops`. Of course, no symbolic links are used here.

# 3 Open Issues

### 3.1 Architectures and Modes

Currently, there is something like a support for different architectures: We allow different sets of source-files and options depending on the architecture to build. When building for multiple architectures from one source-tree, each architecture is built into a separate subdirectory. However, for compatibility reasons all binaries and libraries are installed into the same installation directories. Hence, this support is really suboptimal.

There is no final consensus on how to support different target systems. This is, the same hardware platform and the same kernel, but different execution environments. We see the need to support the following ones:

**oskit10** aka `oskit10_l4env_full`. This mode allows to use drivers and the infrastructure from the Flux OSKit 0.97, together with the L4Env [3]. Binaries are linked against the huge freebsd-libc.

**loader** Like oskit10 mode, but libraries are dynamically linked. Binaries must be loaded with the L4Env loader [4].

**l4env** aka `oskit10_l4env_tiny`. This mode allows to use some basic libc functionality from the Flux OSKit 0.97, together with the L4Env. Binaries are linked against the small mini-libc `libmc`.

**sigma0** This has no support for L4Env, the result is a plain L4 application. Binaries are linked against the small mini-libc `libmc` from the Flux OSKit 0.6.

**l4linux** Use this mode to build $L^4$Linux user-mode applications with L4Env support. Binaries are linked statically against special L4Env libraries and dynamically against standard-libc.

**l4linux-kern** Use this mode to build $L^4$Linux kernel-mode modules with L4Env support. Modules are linked against special L4Env libraries.

**host** Build an application for the host operating system.

### 3.2 ”doc” Role

A role for documentation might be handy. This should have a nice environment for doxygenned documentation and should provide support for latex, xdvi, postscript and PDF generation. We also have to unify how to publish this documentation on the web.

**Request from Lars:** Allow to define the target directory in the global Makeconf.local.

**Answer:** Define a variable, e.g., `$(HTML_DOC_DIR)` specifying were to install the html-files. This could then be set to `$(HOME)/public_html/docs/$(PKGNAME)/`.

# 4 Acknowledgements

This document is based on the work of Lukas Grützmacher, Michel Hohmuth, Jork Löser and Lars Reuther done in 2000. Since then, Frank Mehnert proposed a lot of functionality which is included in this document as well.

# 5 References

[1] DROPS Building Infrastructure Proposal. Postsript, PDF and http versions at `http://os.inf.tu-dresden.de/~jork/local/drops_infra/`.

[2] DROPS Make Macros within 2000. Macros and some documentation can be found at the CVS: `/home/cvs/l4/drops/mk/`.

[3] L4Env. Documentation at http://os.inf.tu-dresden.de/l4env/.

[4] LOADER. Documentation at http://os.inf.tu-dresden.de/~fm3/doc/loader/.

# 6 Examples

Here we show some Makefile examples. As a base, we used the names package and modified its Makefiles. The names package has the following subtree structure:

```
pkg/
 +- names/
     +- examples/
     |    +- demo/
     |    +- lister/
     +- include/
     +- doc/
     +- idl/
     +- lib/
     |    +- src/
     +- man/
     +- server/
          +- src/
          +- include/
```

The Makefiles for the subdirectories names/, examples/, lib/ and server/ only differ in the values for $PKGDIR, we only list one.

Subdirs

```
PKGDIR   = .
L4DIR    ?= $(PKGDIR)/../..

include $(L4DIR)/mk/subdir.mk
```

examples/demo/Makefile

```
PKGDIR   = ../..
L4DIR    ?= $(PKGDIR)/../..

SRC_C            = demo.c
MODE             = sigma0

TARGET           = names_demo
DEFAULT_RELOC    = 0x00E00000

include $(L4DIR)/mk/prog.mk
```

examples/lister/Makefile

```
PKGDIR   = ../..
L4DIR    ?= $(PKGDIR)/../..

SRC_C    = main.c
TARGET   = names_lister
MODE     = host

PRIVATE_INCDIR = $(L4INCDIR)

LIBS     = -lnames $(ROOTLIB) -ll4sys

include $(L4DIR)/mk/prog.mk
```

lib/src/Makefile

```
PKGDIR   = ../..
L4DIR    ?= $(PKGDIR)/../..

TARGET   = lib$(PKGNAME).a
MODE         = sigma0
CLIENTIDL = names.idl

PRIVATE_INCDIR += ..

SRC_C    = $(wildcard *.c)

include $(L4DIR)/mk/lib.mk
```

server/src/Makefile

```
# scan the L4 global configuration file
PKGDIR          ?= ../..
L4DIR          ?= $(PKGDIR)/../..

DROPSCONF        = y

TARGET                   = $(PKGNAME)
DEFAULT_RELOC_x86 = 0x002d0000
DEFAULT_RELOC_arm = 0x00100000
MODE                   = sigma0
SERVERIDL           = names.idl

PRIVATE_INCDIR           = $(PKGDIR)/server/include

SRC_C                  += names.c predefined.c

ifeq ($(ARCH),x86)
LIBS                  += -llogserver
endif

ifeq ($(CONFIG_EVENT),y)
SRC_C                  += events.c
LIBS                  += -levents -ll4util -lnames
endif

include $(L4DIR)/mk/prog.mk
```

The server/include subdirectory contains no Makefile.

We did not look at the Makefile in the man/ subdirectory.