



**uClinux-dist
Developers Guide**

1.1

December 28, 2004

Atmark Techno, Inc.
<http://www.atmark-techno.com/>

Armadillo Official Site
<http://armadillo.atmark-techno.com/>

SUZAKU Official Site
<http://suzaku.atmark-techno.com/>

Table of Contents

1.	Introduction	1
1.1.	About This Manual	1
1.2.	Typographical Conventions	1
1.3.	Command Input Example Conventions	1
1.4.	A Word of Thanks	2
2.	About uClinux-dist	3
3.	Default Image Build	4
3.1.	Acquiring Source Code	4
3.2.	Restoring Source Code Archive	4
3.3.	Configuration	5
3.4.	Build	6
3.5.	Image	6
3.6.	Summary	6
4.	Directory Structure	7
4.1.	Makefile	7
4.2.	config	7
4.3.	tools	7
4.4.	glibc and uClibc	8
4.5.	user	8
4.6.	vendors	8
5.	Basis of Make	10
6.	Basic Targets	12
6.1.	Configuration (config)	12
6.1.1.	Text Screen Configuration (make config)	12
6.1.2.	Menu Screen Configuration (make menuconfig)	12
6.1.3.	GUI Screen Configuration (make xconfig)	13
6.2.	Cleaning (clean)	13
6.3.	Resolving Dependency (dep)	14
6.4.	Default (all)	14
7.	Image File Creation	15
7.1.	Overall Work Flow	15
7.2.	Basic Operation of Menu Based Configuration	16
7.2.1.	Moving	16
7.2.2.	Submenu Selection	16
7.2.3.	Selection from List	16
7.2.4.	Applicable/Not-Applicable Selection	16
7.3.	Configuration	16
7.3.1.	Main Menu	17
7.3.2.	Vendor/Product Selection	17
7.3.3.	Kernel/Library/Defaults Selection	18
7.3.4.	Kernel Version (Kernel Selection)	18
7.3.5.	Libc Version (C Library Selection)	18
7.3.6.	Default all settings	18
7.3.7.	Customize Kernel Settings	18
7.3.8.	Customize Vendor/User Settings	18
7.3.9.	Update Default Vendor Settings	18
7.4.	Userland Settings	18
7.4.1.	Core Applications	18
7.4.2.	Library Configuration	18
7.4.3.	Flash Tools	18
7.4.4.	Filesystem Applications	19
7.4.5.	Network Applications	19
7.4.6.	Miscellaneous Applications	19

7.4.7.	Busybox	19
7.4.8.	Tinylogin	19
7.4.9.	MicroWindows	19
7.4.10.	Games	19
7.4.11.	Miscellaneous Configuration	19
7.4.12.	Debug Builds	19
7.5.	Resolving Kernel Source Code Dependency	19
7.6.	Building	19
7.7.	Detailed Build Work Flow	20
7.7.1.	subdirs Target	20
7.7.2.	romfs Target	21
7.7.3.	module	22
7.7.4.	module_install	22
7.7.5.	image	22
8.	Product Directory	23
8.1.	config.arch	23
8.2.	config.linux-2.4.x	23
8.3.	config.vendor	24
8.4.	config.uClibc	24
8.5.	Makefile	24
9.	romfs Installation Tool	25
9.1.	Overview	25
9.2.	Installing Files	27
9.3.	Installing Directories	27
9.4.	Creating Links	28
9.5.	Adding Information to Files	29
9.6.	Conditional Execution	29
10.	Adding New Applications	30
10.1.	Out of Tree Compiling	30
10.1.1.	Preparation	30
10.1.2.	Source Code	30
10.1.3.	Build (uClinux)	31
10.1.4.	Installing	32
10.1.5.	Creating the Image File	32
10.1.6.	Multiple Source Code Files	32
10.1.7.	pthread Support	34
10.2.	Product Specific Applications	34
10.2.1.	Directory Creation	34
10.2.2.	Preparing Source Code	35
10.2.3.	Application Configuration	35
10.2.4.	Build	35
10.3.	Merging to The user Directory	36
10.3.1.	Directory Creation	36
10.3.2.	Preparing Source Code	36
10.3.3.	Application Configuration	36
10.3.4.	Application Selection	37
10.3.5.	Build	37
10.3.6.	Configuration Naming Rule	37
10.3.7.	Multiple Applications	37
11.	Using Multiple Kernels	40
12.	Specific Applications	41
12.1.	NetFlash	41
12.2.	Flatfsd	42

List of Tables

Table 1-1 Typographical Conventions	1
Table 1-2 Relationship between Display Prompt and Execution Environment	1

List of Figures

Figure 6-1 Menu Based Configuration Screen	13
Figure 6-2 GUI Based Configuration Screen.....	13
Figure 7-1 Image Creation Work Flow.....	15
Figure 10-1 hello Added to the Menu	37

List of Examples

Example 3-1 File Name of uClinux-dist	4
Example 5-1 Creating Directory for Makefile Practice	10
Example 5-2 A Simple Makefile	10
Example 5-3 make Execution	10
Example 5-4 Execution of “make” Specifying a Goal	11
Example 5-5 Execution of “make” Specifying Two or More Goals	11
Example 6-1 Text Based Configuration	12
Example 6-2 Command for activating Menu Based Configuration	13
Example 6-3 Command for activating GUI Based Configuration	13
Example 7-1 Execution of make dep	19
Example 7-2 Execution of make	20
Example 7-3 Calling of Product Makefile From user/Makefile	21
Example 7-4 romfs Target Processing at The Top Level Makefile	21
Example 7-5 romfs Target Processing at user/Makefile	22
Example 9-1 Help for romfs-inst.sh	25
Example 9-2 romfs-inst.sh Command Syntax	26
Example 10-1 uClinux-dist/config/config.in Changes	36
Example 10-2 uClinux-dist/user/Makefile Changes	36
Example 10-3 Changes to uClinux-dist/config/config.in (Multiple Applications)	38
Example 10-4 Changes to uClinux-dist/user/Makefile (Multiple Applications)	38
Example 10-5 Makefile (Multiple Applications)	38

1. Introduction

1.1. About This Manual

This document covers the following areas of information necessary for using uClinux-dist distributed by uClinux.org.

- Kernel and Userland Builds
- Customization
- Application Development
- Particular Applications

Unless otherwise specified, the work PC is assumed to run on a Linux-based OS. Under Windows, a Linux environment can easily be constructed using coLinux. For more information on coLinux, refer to <http://www.colinux.org/>.

We hope the information in this document will help you get the best functionality out of the uClinux-dist.

1.2. Typographical Conventions

The following typographical conventions are used in this manual.

Table 1-1 Typographical Conventions

Font Example	Description
Standard Font	Standard Text
[PC ~]\$ ls	Prompt and user input character strings
⋮	Omission of output after command execution.

1.3. Command Input Example Conventions

The command input examples contained in this manual are based on the assumed execution environment associated with the respective display prompt. The directory part “/” will differ depending on the current directory. The home directory of each user is represented by “~”.

Table 1-2 Relationship between Display Prompt and Execution Environment

Prompt	Command Execution Environment
[PC /]#	To be executed by a privileged user on the work PC
[PC /]\$	To be executed by a general user on the work PC
[Target /]#	To be executed by a privileged user on the target board
[Target /]\$	To be executed by a general user on the target board

1.4. A Word of Thanks

The software used in uClinux-dist is composed of Free Software / Open Source Software. This Free Software / Open Source Software is the result of efforts from developers from all over the world. We would like to take this chance to express our gratitude.

uClinux is supported by the work of Mr. D. Jeff Dionne, Mr. Greg Ungerer, Mr. David McCullough and all person's participating in the uClinux development list.

uClibc and Busybox are developed and maintained by Mr. Eric Andersen and their respective communities.

2.About uClinux-dist

uClinux-dist is a source code based distribution distributed by uClinux.org.

While uClinux-dist was first made for uClinux¹, it is not solely for uClinux use. Modifications were made to make it compatible with uClinux, and it can also be used with MMU CPUs such as i386, ARM, and PowerPC by selecting existing Linux at configuration time.

As there are some restrictions in uClinux that existing Linux does not have, it was necessary to make existing Linux applications compatible with uClinux on an individual basis. uClinux-dist automates the process of combining the uClinux compatible application with the Linux Kernel build system and generating the write files to be written to flash memory etc.

Existing Linux distributions are mainly designed for desktop and server PCs. Therefore, there were problems with file size and memory usage etc. when employing them as a base for embedded devices. As uClinux-dist assumes the use of embedded devices from the beginning, it allows for the selection of only the required functions at compile time.

Moreover, uClinux-dist has many other excellent features. By storing configuration settings for individual products, it allows for simple creation of image files for a specific product. Also, Kernel and Userland configuration can be carried out from the same menu, and then both outputted as one image file.

Therefore, the merits of uClinux-dist are not restricted to products that use uClinux, and even extend to embedded devices that run Linux and include a MMU.

¹ uClinux is a Linux distribution designed to run on microcomputers without a MMU. It is generally utilized in products that do not have large-capacity auxiliary storage such as a HDD. The results of uClinux development have been merged in Linux 2.6 series.

3. Default Image Build

This section describes how to make a default image for the chosen target board.

All operations should be performed as a general user. Mistakes made as a root user may damage the operating environment of the development machine.

3.1. Acquiring Source Code

uClinux distributed by Atmark Techno, Inc. can be downloaded from the URL shown below:

<http://download.atmark-techno.com/dist/>

It can be also acquired from the Atmark Techno, Inc. product portal site or development kit.

The original uClinux-dist as distributed by uClinux.org has a file name that includes the date, as in “uClinux-dist-YYYYMMDD.tar.gz”. Here, “YYYY” stands for year, “MM” for month and “DD” for day. Files distributed by Atmark Techno, Inc. have “atYYYYMMDD” added.

Example 3-1 File Name of uClinux-dist

```
[PC ~]$ ls
uClinux-dist-at20040408.tar.gz
```

3.2. Restoring Source Code Archive

uClinux-dist can be restored anywhere in the system. Choose a place that is convenient to work with and where the hard drive has enough room to store it. Up to 500MB after restoring and up 1GB after compiling may be required. Here, uClinux is restored to the user home directory (~/) for convenience.

```
[PC ~]$ tar xvzf uClinux-dist-atYYYYMMDD.tar.gz
:
:
[PC ~]$ ls
uClinux-dist-atYYYYMMDD/
```

Hereinafter, uClinux-dist-atYYYYMMDD is described as uClinux-dist.

Unlike uClinux-dist distributed by uClinux.org, uClinux-dist distributed by Atmark Techno, Inc. does not contain Linux Kernel source code. The Kernel source code suitable for the product being used can be downloaded from the product portal site, or obtained from the CD-ROM included in the development kit.

The Linux Kernel source code should be restored within uClinux-dist. Restore the Kernel for each product, and create a symbolic link with the name linux-2.4.x within the uClinux-dist directory.

```
[PC ~]$ ls
linux-2.4.24-foo.tar.gz uClinux-dist-YYYYMMDD-atX/
uClinux-dist-atYYYYMMDD.tar.gz
[PC ~]$ tar xvzf linux-2.4.24-foo.tar.gz
:
:
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ ln -s ../linux-2.4.24-foo ../linux-2.4.x
[PC ~/uClinux-dist]$ ls -l linux-2.4.x
lrwxrwxrwx          linux-2.4.x -> ../linux-2.4.24-foo
```

3.3. Configuration

Configure uClinux for the target board. Start configuration by entering the command like the following example.

```
[PC ~/uClinux-dist]$ make config
```

Enter “**AtmarkTechno**” when asked which board vendor to use.

```
[PC ~/uClinux-dist]$ make config
config/mkconfig > config.in
#
# No defaults found
#
*
* Vendor/Product Selection
*
*
* Select the Vendor you wish to target
*
Vendor (3com, ADI, Akizuki, Apple, Arcturus, Arnewsh, AtmarkTechno, Atmel,
Avnet, Cirrus, Cogent, Conexant, Cwlinux, CyberGuard, Cytek, Exys, Feith,
Future, GDB, Hitachi, Imt, Insight, Intel, KendinMicrel, LEOX, Mecel,
Midas, Motorola, NEC, NetSilicon, Netburner, Nintendo, OPENcores, Promise,
SNEHA, SSV, SWARM, Samsung, SecureEdge, Signal, SnapGear, Soekris, Sony,
StrawberryLinux, TI, TeleIP, Triscend, Via, Weiss, Xilinx, senTec)
[SnapGear] (NEW) AtmarkTechno
```

You will then be prompted for the board name. Enter the board name being used. “**Armadillo-9**” is entered here as an example.

```
*
* Select the Product you wish to target
*
AtmarkTechno Products (Armadillo-9, SUZAKU) [Armadillo-9] (NEW) Armadillo-9
```

The C library to be used is specified next. The supported library depends on the board being used. For the Armadillo-9, “None” is selected. For more information, refer to the software manual of the product being used.

```
*
* Kernel/Library/Defaults Selection
*
*
* Kernel is linux-2.4.x
*
Libc Version (None, glibc, uC-libc, uClibc) [uClibc] (NEW) None
```

Next you will be asked whether or not to use default settings. Select “Yes”

```
Default all settings (lose changes) (CONFIG_DEFAULTS_OVERRIDE) [N/y/?]
(NEW) y
```

Enter “No” for the last three questions.

```
Customize Kernel Settings (CONFIG_DEFAULTS_KERNEL) [N/y/?] n
Customize Vendor/User Settings (CONFIG_DEFAULTS_VENDOR) [N/y/?] n
Update Default Vendor Settings (CONFIG_DEFAULTS_VENDOR_UPDATE) [N/y/?] n
```

The Build system will start configuration after all questions have been answered. You will be returned to the prompt once the configuration has finished.

3.4. Build

Enter the following command to execute a build. If asked any questions during the build process, hit the “Enter” key.

```
[PC ~/uClinux-dist]$ make dep all
```

3.5. Image

Image files will be created in the “images” directory. The number and type of files created may differ dependant on the product selected. Refer to the software manual of the product being used.

Also refer to the software manual of the product being used for the method of writing a created image file to the product.

3.6. Summary

The series of commands entered and options selected in this chapter have been summarized below.

```
[PC ~]$ ls
linux-2.4.24-foo.tar.gz  uClinux-dist-YYYYMMDD-atX.tar.gz
[PC ~]$ tar xvzf uClinux-dist-YYYYMMDD-atX.tar.gz
[PC ~]$ tar xvzf linux-2.4.24-foo.tar.gz
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ ln -s ../linux-2.4.24-foo ../linux-2.4.x
[PC ~/uClinux-dist]$ make config
Vendor AtmarkTechno
AtmarkTechno Products Armadillo-9
Libc Version None
Default all settings y
Customize Kernel Settings n
Customize Vendor/User Settings n
Update Default Vendor Settings n
[PC ~/uClinux-dist]$ make dep all
[PC ~/uClinux-dist]$ ls image/
image.bin  linux.bin  romfs.img
```

4. Directory Structure

This chapter explains about the directory structure of uClinux-dist. Comprehending the directory structure is of great importance when carrying out development based on uClinux-dist.

This chapter provides an overview of each directory while introducing any other chapters that contain related explanations.

```
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ tree -L 1 -F
.
|-- COPYING
|-- Documentation/
|-- Makefile
|-- README
|-- SOURCE
|-- bin/
|-- config/
|-- freeswan/
|-- glibc/
|-- lib/
|-- linux-2.4.x -> ../kernel/linux-suzaku/
|-- tools/
|-- uClibc/
|-- user/
`-- vendors/

11 directories, 4 files
```

4.1. Makefile

uClinux-dist uses the make command to carry out all operations. This uClinux-dist/Makefile will be called the top level Makefile to distinguish it from Makefiles in other directories. The top level Makefile is an important file that controls the build of uClinux-dist. The target defined here is explained in detail in Chapter “6. Basic Target”. The Make command explained briefly in chapter “5. Basis of Make”.

4.2. config

The config directory contains the scripts and Makefiles necessary for configuration”.

Makefiles and config.in are covered in detail in chapter “10.2 Product Specific Applications”.

4.3. tools

The “tools” directory contains various tools necessary for building. The romfs-inst.sh file in this directory is often used by the Makefile of product directories. It is explained in detail in chapter “9. romfs Installation Tools”.

cksum is sometimes used with NetFlash. For information on NetFlash, refer to chapter “12.1 NetFlash”.

4.4. glibc and uClibc

GNU C library (glibc) and uClib are the C libraries employed in uClinux-dist.

4.5. user

Userland applications are stored in this directory. While most are GNU/Linux applications adapted for use on uClinux, applications developed specifically for uClinux are also included.

Applications characteristic of those produced from carrying out software development based on uClinux-dist are detailed in chapter “12. Unique Applications”.

4.6. vendors

The vendors directory is shown below.

```
[PC ~/uClinux-dist]$ tree vendors
vendors
:
:
|-- AtmarkTechno
|   |-- Armadillo
|   |-- Armadillo-9
|   |-- Armadillo-J
|       |-- Makefile
|       |-- config.arch
|       |-- config.linux-2.4.x
|       |-- config.uClibc
|       |-- config.vendor
|       |-- default
|       |-- etc
|   |-- SUZAKU
|       |-- Makefile
|       |-- config.arch
|       |-- config.linux-2.4.x
|       |-- config.uClibc
|       |-- config.vendor
|       |-- default
|       |-- etc
:
:
|-- config
|   |-- arm
|       |-- config.arch
|   |-- armmommu
|       |-- config.arch
|   |-- h8300
|       |-- config.arch
|   |-- i386
```

```
| |      `-- config.arch
| |      |-- i960
| |      `-- config.arch
| |      |-- m68knommu
| |      `-- config.arch
| |      |-- microblaze
| :
| :
```

The “vendors” directory contains many directories with vendor names, with AtmarkTechno being one of those. Each vendor directory contains subdirectories for their products. The AtmarkTechno directory contains various products compatible with uClinux-dist including the Armadillo. These directories will be referred to as the product directory. The product directories contain the various files required to build the images for each product. For more information on the product directories, refer to chapter “8 Product Directories”.

The vendors directory also contains the config directory. The default settings for each architecture are contained under the name config.arch in each directory. These config.arch files are referenced from the config.arch file in the product directories. For more information, refer to chapter “8.1 config.arch”.

5. Basis of Make

The make command is widely used to compile programs. It decides what source code and in what way to compile and executes the necessary commands for compiling. A Makefile is required to use the make command. Information such as the dependency of source code to be compiled and the commands necessary for compiling is written in the Makefile.

The uClinux-dist build system is also based on Makefiles. The following briefly explains about the make command and basics of Makefiles.

Firstly, create a directory for practice use. Here it is named “maketest”.

Example 5-1 Creating Directory for Makefile Practice

```
[PC ~]$ mkdir maketest
[PC ~]$ cd maketest
[PC ~/maketest]$
```

Next, create a simple Makefile in the maketest directory.

Example 5-2 A Simple Makefile

```
hello:
    echo 'Hello World'

bye:
    echo 'Bye bye'
```

In the Makefile, “hello” and “bye” are called targets. A target should always end with colon (“:”). The line following the target is the command that will actually be executed. Here, the echo command is executed to output “Hello World” without compiling. Commands to be executed must be preceded by a tab.

Example 5-3 make Execution

```
[PC ~/maketest]$ ls
Makefile
[PC ~/maketest]$ make
echo 'Hello World'
Hello World
[PC ~/maketest]$
```

The above is the result of actually executing the make command. Once the make command is executed, the command in the Makefile, echo ‘Hello World’ and its result “Hello World” are displayed.

The “make” command can accept targets defined in the Makefile as arguments. The specified target is called a goal.

Example 5-4 Execution of “make” Specifying a Goal

```
[PC ~/maketest]$ make hello
echo 'Hello World'
Hello World
[PC ~/maketest]$ make bye
echo 'Bye bye'
Bye bye
[PC ~/maketest]$
```

When the make command does not take an argument, it will execute the topmost target in the Makefile as the goal. Therefore, “make” and “make hello” produce the same result. In many Makefiles, the topmost target is defined with the name “all”.

Two or more goals can also be specified. In this case, they are sequentially executed from the left.

Example 5-5 Execution of “make” Specifying Two or More Goals

```
[PC ~/maketest]$ make hello bye
echo 'Hello World'
Hello World
echo 'Bye bye'
Bye bye
[PC ~/maketest]$
```

For more information on make and Makefiles, refer to a make manual or info.

6. Basic Targets

This section explains about targets often used when building with uClinux-dist. As has already been explained, the purpose of uClinux-dist is “creating image files to be written to the target board”. The following describes in detail targets used for “default image builds”.

6.1. Configuration (config)

The following three kinds of target have been prepared for configuration use. All three are targets used for uClinux-dist configuration.

6.1.1. Text Screen Configuration (make config)

make config is the configuration method used in chapter “3. Building Default Image”. It is useful when carrying out simple configuration like creating a default image.

Example 6-1 Text Based Configuration

```
[PC ~/uClinux-dist]$ make config
config/mkconfig > config.in
#
# Using defaults found in .config
#
*
* Vendor/Product Selection
*
* Select the Vendor you wish to target
*
Vendor (3com, ADI, Arcturus, Arnewsh, AtmarkTechno, Atmel, Avnet,
Conexant, Cwlinux, Cytek, Exys, Feith, Future, GDB, Hitachi, Imt, Insight,
Intel, KendinMicrel, LEOX, Mecel, Midas, Motorola, NEC, NetSilicon,
Netburner, Nintendo, OPENcores, Promise, SSV, SWARM, Samsung, SecureEdge,
SnapGear, Soekris, TeleIP, Triscend, Via, Weiss, Xilinx, senTec)
[AtmarkTechno]
    defined CONFIG_DEFAULTS_ATMARKTECHNO
*
* Select the Product you wish to target
*
AtmarkTechno Products (SUZAKU) [SUZAKU]
    defined CONFIG_DEFAULTS_ATMARKTECHNO_SUZAKU
*
* Kernel/Library/Defaults Selection
*
* Kernel is linux-2.4.x
*
Libc Version (glibc, uC-libc, uClibc) [uClibc]
```

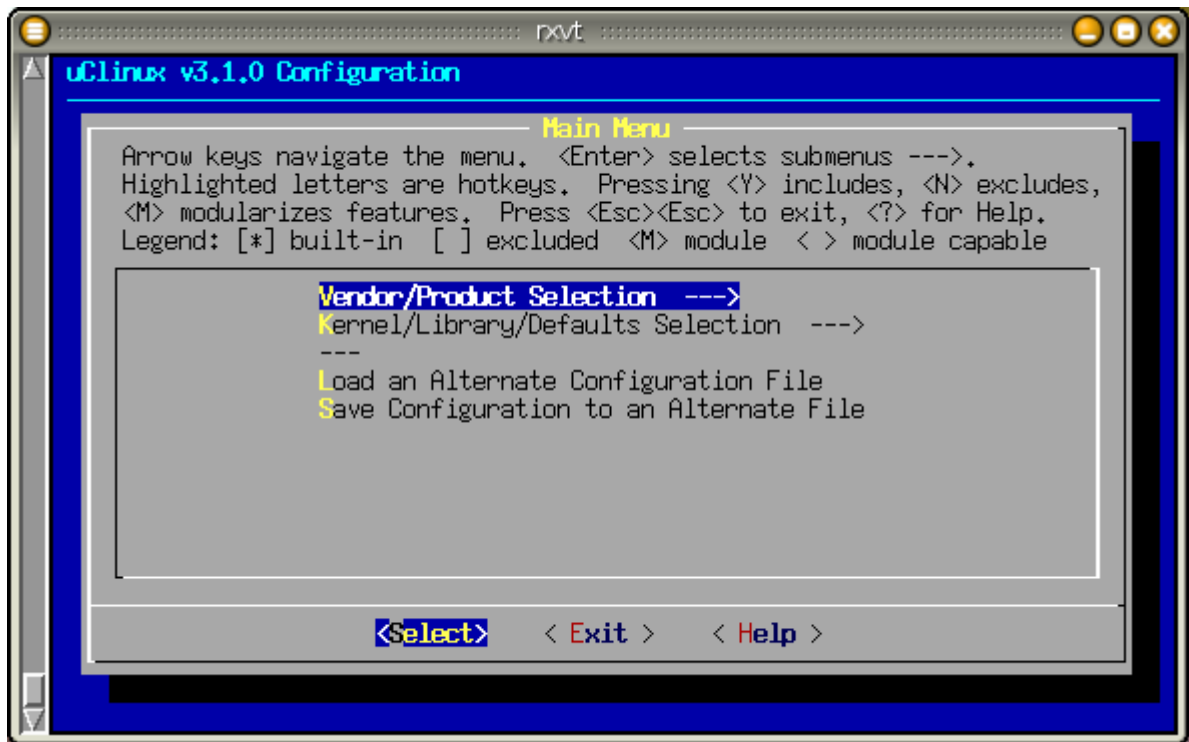
6.1.2. Menu Screen Configuration (make menuconfig)

make menuconfig allows for configuration using a menu screen. The menu screen is drawn with Ncurses. The Ncurses library and header file are required to build programs that the control screen when this target is specified. Most distributions include a package for Ncurses development, so please do install it.

Example 6-2 Command for activating Menu Based Configuration

```
[PC ~/uClinux-dist]$ make menuconfig
```

Figure 6-1 Menu Based Configuration Screen



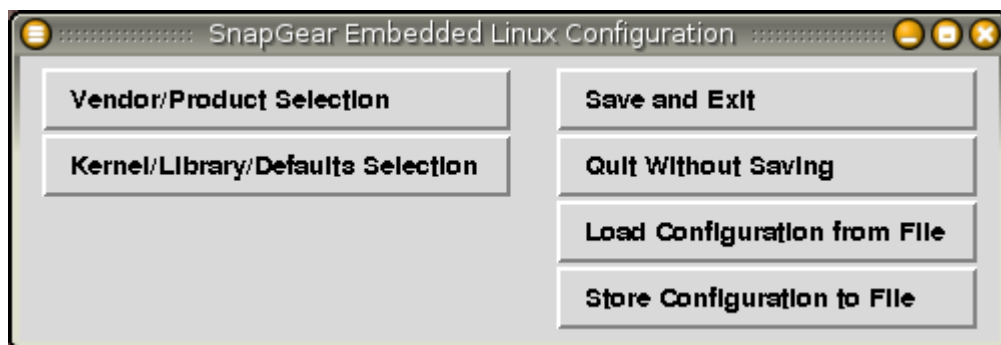
6.1.3. GUI Screen Configuration (make xconfig)

The make xconfig target can be used in an environment where the X Window System is available. Mouse operation is possible in “make xconfig” unlike the Menu Screen in the terminal.

Example 6-3 Command for activating GUI Based Configuration

```
[PC ~/uClinux-dist]$ make xconfig
```

Figure 6-2 GUI Based Configuration Screen



6.2. Cleaning (clean)

This is a target to clean the inside of uClinux-dist. The following three kinds of “clean” target are available in uClinux-dist.

make clean

Cleans the Kernel directory, and deletes files in the romfs/image directory.

make real_clean

Deletes the romfs/image directory, and deletes configuration files such as .config etc.

make distclean

Executes the distclean target in Kernel directory etc.

6.3. Resolving Dependency (dep)

It is necessary to resolve dependency before executing a make in Linux Kernel build systems 2.4 series and earlier. This target resolves dependency in the Kernel directory. This target should always be executed after restoring the uClinux-dist source archive or after executing distclean.

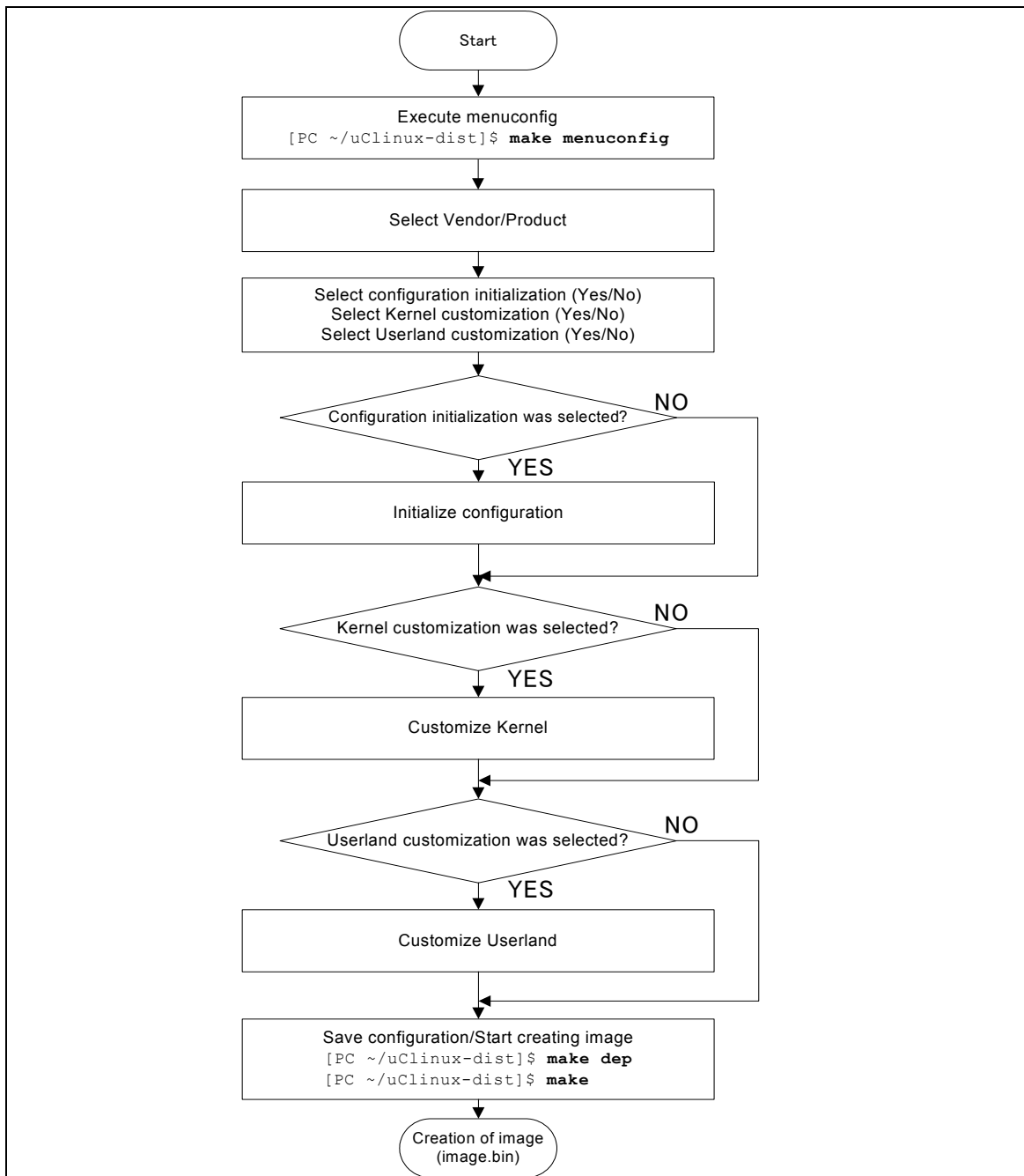
6.4. Default (all)

The “all” target is the uClinux-dist default target. This target is executed by executing a make command without any options. The “all” target compiles necessary code (Kernel, Userland applications and libraries) and then creates the image file to be transferred to the target board.

7. Image File Creation

7.1. Overall Work Flow

Figure 7-1 Image Creation Work Flow



The following explains in order the procedure to create a default image. This procedure can be roughly divided into three stages.

- Configuration

- Resolving dependence of Kernel source code
- Building

7.2. Basic Operation of Menu Based Configuration

7.2.1. Moving

Menus can be navigated using the cursor keys.

7.2.2. Submenu Selection

Submenus can be selected by pushing the Enter key. Submenus are indicated with '--->'.

7.2.3. Selection from List

Items displayed with small parentheses “()” can be selected from a list. Move to the list screen with the Enter key, move to the target with the up and down cursor keys, and make the selection with the Enter key.

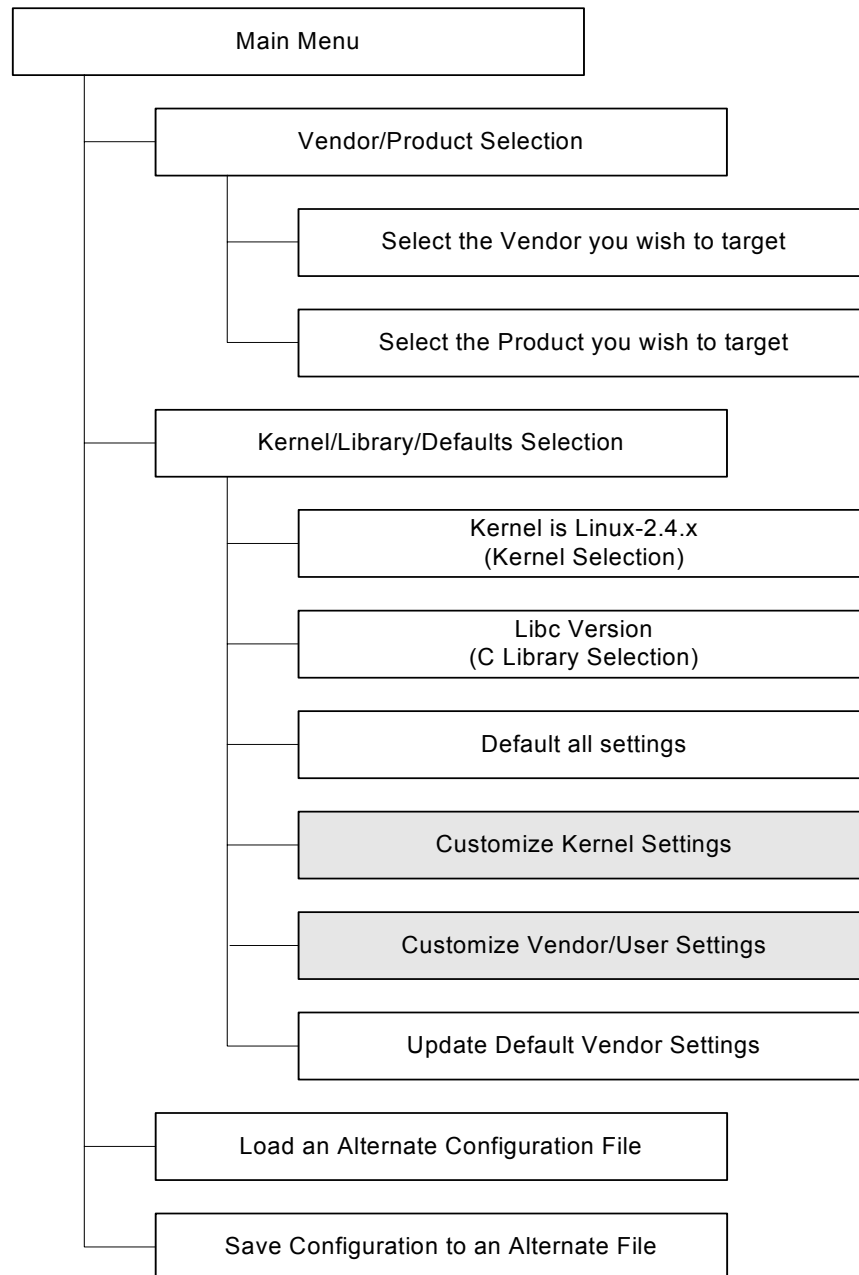
7.2.4. Applicable/Not-Applicable Selection

Items displayed with brackets “[]” provide for applicable/not-applicable selection. An asterisk “*” is displayed in the brackets when the item is selected.

7.3. Configuration

Configuration can be carried out using either one of the commands introduced in “6.1 Configuration (config)”. The following explanation makes use of the make menuconfig command.

Configuration can be roughly divided as follows.



When “Kernel Customization” or “Vendor/User Customization” is selected, the current menu will once close and then the appropriate configuration menu will be automatically displayed.

Configuration is started as follow.

```
[PC ~/uClinux-dist]$ make menuconfig
```

7.3.1. Main Menu

The Main Menu Screen is displayed after executing the menuconfig command. Each submenu can be reached from this screen.

7.3.2. Vendor/Product Selection

Select the vendor and product. The vendor is selected first, followed by the product.

7.3.3. Kernel/Library/Defaults Selection

Select the Kernel, library and defaults. The following menus are part of this submenu.

7.3.4. Kernel Version (Kernel Selection)

Select the Kernel versions to build for products that support two or more Kernels. Kernels currently supported by Atmark Techno, Inc. are limited to 2.4 series, so other versions can not be selected.

7.3.5. Libc Version (C Library Selection)

Select the C libraries to build for products that support two or more C libraries. The following four choices can be selected.

- None
- glibc – GNU C library
- uC-libc
- uClibc

Binaries previously installed in the development environment are used when “None” is selected. Code included in uClinux-dist is built in all other options.

As compatible libraries differ by product, please refer to the software manual of the product.

7.3.6. Default all settings

Set all settings to their default state. Note that all previously changed configuration information will be lost.

7.3.7. Customize Kernel Settings

Select whether or not to perform Linux Kernel configuration. When this is selected, the Kernel configuration screen will appear automatically after uClinux-dist configuration has completed.

7.3.8. Customize Vendor/User Settings

Select whether or not to perform Userland configuration. When this is selected, the Userland configuration screen will appear automatically after uClinux-dist configuration has completed. The Userland menu is detailed in the following section.

7.3.9. Update Default Vendor Settings

Replace default settings with present settings. Note that there is no way to return to previous defaults after this update has been made.

7.4. Userland Settings

The Userland menu is divided into the following items. This section briefly describes each of these items.

7.4.1. Core Applications

Contains the basic applications required to operate as a system. init for system initialization and login for user identification can be selected in this section.

7.4.2. Library Configuration

Libraries required by applications can be selected.

7.4.3. Flash Tools

Applications related to Flash memory can be selected. Netflash, a network update application described in a later chapter, has been selected here.

7.4.4. Filesystem Applications

Applications related to the file system can be selected. Flatfsd explained in a following chapter can be selected here. Apart from this, mount, fdisk, the ext2 file system, the Reiser file system and Samba etc. are included here.

7.4.5. Network Applications

Applications related to networking can be selected. ppp and utilities for wireless networks etc. are included as well as dhcpcd-new, ftpd, ifconfig, inetd and tftpd.

7.4.6. Miscellaneous Applications

Applications that do not belong to any other category described above are included here. General Unix commands (cp, ls, rm), editors, audio related applications, and a script language interpreter etc. are included.

7.4.7. Busybox

Customization of Busybox can be performed. Busybox is a single command that includes multiple functions, and is well used in embedded Linux. As it has many customizable items, it is categorized in its own separate section.

7.4.8. Tinylogin

Tinylogin is also a single application that includes multiple functions. It offers functions related to user identification, such as login, password and getty etc. As it has many customizable items, it is categorized in its own separate section.

7.4.9. MicroWindows

MicroWindows is a graphical window environment targeted towards embedded devices. It can be used when devices with LCDs etc. are developed.

7.4.10. Games

Games. No explanation needed.

7.4.11. Miscellaneous Configuration

Various settings have been brought together here. The root user password also can be changed here.

7.4.12. Debug Builds

Various debug options have been brought together here. These can be used when debugging applications under development.

7.5. Resolving Kernel Source Code Dependency

Image builds also include kernel builds. Dependencies must be resolved before carrying out a build of Linux kernels 2.4 or earlier. This applies to uClinux also.

The make dep command is used.

Example 7-1 Execution of make dep

```
[PC ~/uClinux-dist]$ make dep
:
:
[PC ~/uClinux-dist]$
```

7.6. Building

The build is carried out after resolving dependency. As the build system will perform everything automatically, the developer only needs to enter the make command.

Example 7-2 Execution of make

```
[PC ~/uClinux-dist]$ make
:
:
[PC ~/uClinux-dist]$ ls images
image.bin  linux.bin  romfs.img
```

The image.bin file should be created in the images directory after the make command has completed.

7.7. Detailed Build Work Flow

By understanding how the build proceeds internally and how it eventually becomes an image file, it becomes possible to build only the required parts. This can greatly reduce the time required for the build. Moreover, this understanding may be indispensable in performing customization for a specific product.

While carrying out a default build does create the image file, it executes many targets in doing so. The build rule of the default target is defined in the Makefile in the uClinux-dist root directory as follows.

```
ifeq (.config,$(wildcard .config))
include .config

all: subdirs romfs modules modules_install image
else
all: config_error
endif
```

From this, it can be seen that once the default build is executed, the build proceeds in this order: subdirs, romfs, modules_install and image.

Following this order, each target is explained below.

7.7.1. subdirs Target

The subdirs build rule of the Makefile in the uClinux-dist root directory is shown below.

```
DIRS      = $(VENDOR_TOPDIRS) include lib include user
:
:
subdirs: linux
        for dir in $(DIRS) ; do [ ! -d $$dir ] || $(MAKEARCH_KERNEL) -C
$$dir ||
        exit 1 ; done
```

When building the subdirs target, the build proceeds in the order of linux, \$(VENDOR_TOPDIRS), include lib and include user.

The linux target performs the build for the selected Linux Kernel. Although Kernel versions 2.0, 2.4 and 2.6 are supported in uClinux-dist, Atmark Techno, Inc. products currently only support Linux Kernel 2.4.

While VENDOR_TOPDIRS can be specified by config.arch in the product directory, it seems that not many products do in fact specify this directory. Many products that do specify VENDOR_TOPDIRS specify a directory named boot. Atmark Techno, Inc products do not use this variable.

The lib directory contains libraries. While uClibc and glibc are not included in the lib directory, a symbolic link is created in the directory at time of configuration.

The user directory contains the Userland applications. The user directory has an exclusive Makefile, and the top level Makefile hands control to this exclusive Makefile.

The Makefile in the product directory is called from the Makefile in the user directory. The following example is an extract from user/Makefile.

Example 7-3 Calling of Product Makefile From user/Makefile

```
VEND=$(ROOTDIR)/vendors

#
# must run the vendor build first
#
dir_v = $(VEND)/$(CONFIG_VENDOR)/$(CONFIG_PRODUCT)/.
dir_p = $(ROOTDIR)/prop

dir_y =
dir_n =
dir_ =
:
:
:
:

all: config
    for i in $(sort $(dir_y)) $(dir_v) $(dir_p); do \
        [ ! -d $$i ] || make -C $$i || exit $$? ; \
    done
```

7.7.2. romfs Target

The romfs target is called recursively for each directory. In most cases, romfs-inst.sh is used to install the necessary files to the uClinux-dist/romfs directory.

Example 7-4 romfs Target Processing at The Top Level Makefile

```
DIRS    = $(VENDOR_TOPDIRS) include lib include user
:
:
:
romfs:
    for dir in $(DIRS) ; do [ ! -d $$dir ] || $(MAKEARCH) -C $$dir
```

```
romfs || exit 1 ; done
    -find $(ROMFSDIR)/. -name CVS | xargs -r rm -rf
done
```

Example 7-5 romfs Target Processing at user/Makefile

```
VEND=$(ROOTDIR)/vendors

#
# must run the vendor build first
#
dir_v = $(VEND)/$(CONFIG_VENDOR)/$(CONFIG_PRODUCT)/.
dir_p = $(ROOTDIR)/prop

dir_y =
dir_n =
dir_ =
:
:
:
:

romfs:
    for i in $(dir_v) $(sort $(dir_y)) $(dir_p) ; do \
        [ ! -d $$i ] || make -C $$i romfs || exit $$? ; \
    done
```

When the build is performed, the build for the selected applications in the user directory is performed first, and then the product Makefile is called.

However, the Makefile in the product directory is called first for the romfs target. While, this is to provide for flexibility for each product, it can be ignored unless performing a considerably complex build.

7.7.3. module

In the Linux Kernel, a large number of drivers etc. can be separated in the form of a module. This target executes module targets within the Linux Kernel build system.

7.7.4. module_install

The Kernel modules built with the above target are installed in romfs. Specifically, they are installed in romfs/lib/modules.

7.7.5. image

The image target executes the image targets of product Makefiles. Many products combine the Kernel image with the Userland image to make one image file.

As the targets of product Makefiles are called directly, the developer can freely handle the process. A general flow is shown below.

1. Create the Linux Kernel binary file (Convert from elf to binary)
2. Create the image file from the romfs directory (Using genext2fs, genromfs)
3. Combine the above two files into one
4. Create the checksum etc. for NetFlash and attach it to the binary file

8. Product Directory

The product directories are a directory group that exists under uClinux-dist/vendor/[Vendor Name]/. For instance, the names of Atmark Techno, Inc. products are listed under uClinux-dist/vendor/AtmarkTechno/.

The product directory contains the parts that differ product to product within the build system, such as the build process defining Makefiles, the config files used for defaults values at build time, and configuration files required by the applications.

The following explains the config files used for default settings as well as Makefiles.

8.1. config.arch

The config.arch files contain architecture depend configuration information. As default values for each architecture are already described, only settings to be overwritten need to be entered.

The following variables can be specified.

CPUFLAGS

Compile flag for CPU can be specified.

VENDOR_CFLAGS

CFLAGS for vendor dependence can be specified.

DISABLE_XIP

Specify "1" to disable XIP (Execute In Place).

DISABLE_SHARED_LIBS

Specify "1" to disable shared libraries.

DISABLE_MOVE_RODATA

Specify "1" to disable read-only data area movement.

LOPT

Specify options passed to the compiler when compiling libraries.

UOPT

Specify options passed to the compiler when compiling Userland applications.

CONSOLE_BAUD_RATE

Specify the serial console baud rate.

8.2. config.linux-2.4.x

config.linux-2.4.x stores the ".config" files created by the Linux Kernel build system under a different name in the product directories.

These files are used as the Kernel configuration default values. To change the Kernel values and then use those values as the defaults, please overwrite these files. It is also possible to use the menu as shown in "7.3.9 Update Default Vendor Settings".

8.3. config.vendor

config.vendor contains information on the Userland applications. It maintains information that the uClinux-dist build system sets with menuconfig etc.

This file is a copy of uClinux-dist/config/.config.

Default values can be changed by overwriting in the same way as config.linux-2.4.x

8.4. config.uClibc

config.uClibc is the configuration file of uClib. uClibc cannot currently be configured from the dist menu. uClibc configuration can however be performed by moving to the uClinux directory and using the uClibc configuration tools there. uClibc configuration employs the Linux Kernel build system in the same way as uClinux-dist.

8.5. Makefile

The Makefile controls the creation of the actual image file. This includes control of the program and device files install destinations, install destinations of data and configuration files required by the program, and checksum generation.

9.romfs Installation Tool

Compiled applications and the various configuration files are installed in the uClinux-dist/romfs directory by the romfs target in the Makefile. Configuration files and data files etc. required by the applications are also installed in the uClinux-dist/romfs directory at the same time.

The reason why romfs is used as the directory name is because, unlike the ext2, ext3, reiserfs, xfs used in desktop and server Linux systems, romfs is used in most embedded systems. However, the romfs directory itself is not dependant on romfs. As described later, jffs2 etc. also use the same romfs directory.

The romfs directory structure is the same as the directory structure that can be seen on the target system when it is booted. The romfs directory becomes the root directory, and holds the bin, dev and etc and so on under it.

uClinux-dist includes a script named romfs-inst.sh that allows for simple installation of files into the romfs directory. This script is contained in the uClinux-dist/tools directory.

romfs-inst.sh is assigned to a variable named ROMFSINST by the Makefile in the uClinux-dist directory. Therefore, it is common to use this script as the variable ROMFSINST without paying attention to where the script is in the Makefile of each directory in uClinux-dist.

9.1. Overview

romfs-inst.sh displays a simple help when the romfs directory specifying environment variable ROMFSDIR is not set.

Example 9-1 Help for romfs-inst.sh

```
[PC ~/uClinux-dist]$ tools/romfs-inst.sh
ROMFSDIR is not set
tools/romfs-inst.sh: [options] [src] dst
    -v          : output actions performed.
    -e env-var  : only take action if env-var is set to "y".
    -o option   : only take action if option is set to "y".
    -p perms    : chmod style permissions for dst.
    -a text     : append text to dst.
    -A pattern  : only append text if pattern doesn't exist in file
    -l link     : dst is a link to 'link'.
    -s sym-link : dst is a sym-link to 'sym-link'.

if "src" is not provided,  basename is run on dst to determine the
source in the current directory.

multiple -e and -o options are ANDed together.  To achieve an OR
affect use a single -e/-o with 1 or more y/n/"" chars in the
condition.

if src is a directory,  everything in it is copied recursively to
dst with special files removed (currently CVS dirs).
```

The command syntax of romfs-inst.sh is shown below.

Example 9-2 romfs-inst.sh Command Syntax

```
romfs-inst.sh [options] [src] dst
```

The “[]” selections can be omitted. The basename command is applied to dst and src used as the return value when src has not been specified. romfs-inst.sh retrieves the value from the current directory (the product directory in the case of product Makefiles).

```
[PC ~]$ basename /foo/bar  
bar
```

For the src directory, all files and directories under it are installed. The CVS directory however is not copied.

The following is a brief explanation of each option.

-v

Display content actually executed.

-e env-var

Specified action only executed when env-var is “y”.

-o option

Specified action only executed when option is “y”.

-p perms

Specify permissions of dst with the chmod format.

-a text [-A pattern]

Add text to dst. text added to dst when -A pattern is specified and pattern is not contained in dst.

-l link

Create a hard-link to link with the name specified in dst.

-s sym-link

Create a symbolic-link to sym-link with the name specified in dst.

The following section details an example of using romfs-inst.sh.

9.2. Installing Files

When file is to be installed, enter the “romfs” target description in the Makefile as follows.

```
romfs:
    $(ROMFSINST) src.txt /etc/dst.txt
```

This means that `src.txt` in the product directory will be installed as `/etc/dst.txt` in the `romfs` directory. If the `romfs` directory name is `/home/myname/uClinux-dist/romfs`, then the file `/home/myname/uClinux-dist/romfs/etc/dst.txt` will be created.

If the `src.txt` file in the product directory is changed to `dst.txt`, the description can be simplified as follows.

```
$(ROMFSINST) /etc/dst.txt
```

As mentioned above, `romfs-inst.sh` retrieves the file from the current directory using the basename of `dst` when `src` is omitted.

Thus the basename of `/etc/dst.txt` is `dst.txt`, and the description above has same meaning as the one below.

```
$(ROMFSINST) dst.txt /etc/dst.txt
```

9.3. Installing Directories

Installing by directory makes installing many files to a target device easier. For instance, when installing many configuration files to the `/etc` directory of the target.

Create a directory named `etc` in the product directory and place the necessary files there. Then enter the following into the Makefile.

```
$(ROMFSINST) /etc
```

As `src` is omitted in this example, `romfs-dist.sh` uses the basename of `dst`. With the basename of `/etc` being `etc`, `romfs-inst.sh` retrieves the file or directory with the name `etc` in the product directory. Then if the name specified is a folder, `romfs-inst.sh` installs the folder along with the contained files.

This can be easily confirmed by using the `tree` command as follows.

```
[PC ~/uClinux-dist]$ tree ~/uClinux-dist/vendor/AtmarkTechno/test/etc
:
[PC ~/uClinux-dist]$ tree ~/uClinux-dist/romfs/etc
:
```

Of course, it is also possible to install the directory with a name different to the saved name.

```
$(ROMFSINST) /etc /var
```

This command installs the directory named etc from the product directory to romfs/var.

9.4. Creating Links

Links can be easily created using romfs-inst.sh. However, it is important to fully understand both hard links and symbolic links.

The -s option is used to create a symbolic link. A symbolic link to a.txt is created here as an example. The romfs target of the product Makefile is as follows.

```
romfs:
    [ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
    $(ROMFSINST) /a.txt
    $(ROMFSINST) -s a.txt /b.txt
```

```
[PC ~/uClinux-dist]$ make clean; make romfs
:
:
[PC ~/uClinux-dist]$ ls -l romfs
total 0
-rw-r--r--  1 yashi yashi 0 Sep 24 05:43 a.txt
lrwxrwxrwx  1 yashi yashi 5 Sep 24 05:43 b.txt -> a.txt
```

The following shows an example of a hard link, using the -l option.

```
romfs:
    [ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
    $(ROMFSINST) /a.txt
    $(ROMFSINST) -l a.txt /b.txt
```

```
[PC ~/uClinux-dist]$ make clean; make romfs
:
:
[PC ~/uClinux-dist]$ ls -il romfs
6077732 a.txt
6296750 b.txt
[PC ~/uClinux-dist]$ ls -il vendors/AtmarkTechno/test/a.txt
6296750 vendors/ATmarkTechno/test/a.txt
```

That b.txt in the romfs directory is a hard link to a.txt in the product directory and not a hard link to a.txt in romfs can be seen from the inode number.

As the use of hard links can be the source of confusion, their use is not recommended unless hard drive space is severely restricted. It also seems that hard links are not highly used in the uClinux-dist romfs.

9.5. Adding Information to Files

Information can be easily added to existing files using rom-inst.sh.

The syntax is as follows.

```
$(ROMFSINST) -a "Character strings" File name in romfs directory
```

```
romfs:
    [ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
    $(ROMFSINST) -a 'Hello' /a.txt
    $(ROMFSINST) -a 'World' /a.txt
```

```
[PC ~/uClinux-dist]$ make clean; make romfs
:
:
[PC ~/uClinux-dist]$ cat romfs/a.txt
Hello
World
```

9.6. Conditional Execution

Conditional execution control is possible with romfs-inst.sh.

```
$(ROMFSINST) -e variable name command to be executed
```

Environment variable names often start with CONFIG_.

```
romfs:
    [ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
    $(ROMFSINST) -e CONFIG_DEFAULTS_ATMARKTECHNO -a 'Hello' /a.txt
    $(ROMFSINST) -e CONFIG_DEFAULTS_UNKNOWN -a 'World' /a.txt
```

```
[PC ~/uClinux-dist]$ make clean; make romfs
:
:
[PC ~/uClinux-dist]$ cat romfs/a.txt
Hello
```

As CONFIG_DEFAULTS_UNKNOWN is not defined, it does not apply as a condition and the “World” character string is not written to a.txt. CONFIG_DEFAULTS_ATMARKTECHNO is defined in uClinux-dist/.config when AtmarkTechno is selected as the vendor name.

10. Adding New Applications

This chapter explains how to create applications that operate on the target board and then how to add the created applications to uClinux-dist.

10.1. Out of Tree Compiling

“Out of Tree” compiling is a method that allows for easy development without the need to make any changes to uClinux-dist. As it uses the uClinux-dist build system, there is no need to write complex Makefiles. It has this name as the directory structure of uClinux-dist can be seen as a tree, and compiling is performed outside of that directory structure.

The following shows how to actually create “Hello World” for a target board.

10.1.1. Preparation

As the build system and library groups contained in uClinux-dist are used in the “Out of Tree” compile, a uClinux-dist that has been compiled once is required. First please check that the uClinux-dist has been configured and built for the target board.

10.1.2. Source Code

Next make a directory for the application outside of the uClinux-dist directory structure. The Makefile and necessary C source code and header files will be stored in this directory.

```
[PC ~]$ ls
uClinux-dist
[PC ~]$ mkdir hello
[PC ~]$ ls
hello uClinux-dist
[PC ~]$ ls hello
Makefile hello.c
```

hello.c, as shown below, is a simple program that can be seen in any C text book.

```
#include <stdio.h>

int main(int argc, char * argv[])
{
    printf("Hello World\n");
    return 0;
}
```

The Makefile shown below is used.

```
# ROOTDIR=/usr/src/uClinux-dist ----- ①
ifndef ROOTDIR
ROOTDIR=../uClinux-dist
endif
ROMFSDIR = $(ROOTDIR)/romfs
```

```
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch

EXEC = hello ----- ②
OBJS = hello.o ----- ③

all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
        -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
        $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
        $(CC) -c $(CFLAGS) -o $@ $<
```

This Makefile can be used as a template when applications besides Hello World are developed. The following three points need to be changed in accordance with the environment.

- ① It is assumed that the uClinux-dist directory exists in parallel to the current directory when ROOTDIR is not specified. When uClinux-dist resides in a different place, remove the comment in this line and change directory name as appropriate.
- ② Specify the executable file name to be created. hello is specified here.
- ③ Specify the object file that the executable file is to be created from. “hello.o” is specified here.

10.1.3. Build (uClinux)

The build for “hello” can be carried out once the Makefile and hello.c are ready. The make command is used for the build. The executable file hello will be created in the directory when the build completes. (This is for uClinux. A different file is created in Linux.)

```
[PC ~/hello]$ make
:
:
[PC ~/hello]$ ls hello*
hello  hello.c  hello.gdb  hello.o
[PC ~/hello]$ file hello*
hello:      BFLT executable - version 4 ram
```

```
hello.c:  ASCII C program text
hello.gdb: ELF 32-bit MSB executable, version 1 (SYSV), statically
linked, not stripped
hello.o:  ELF 32-bit MSB relocatable, version 1 (SYSV), not stripped
```

hello.gdb is an ELF format executable file and is used for debugging. hello is in the uClinux “Flat Binary” format.

10.1.4. Installing

To install the executable file in the uClinux romfs directory, romfs is specified in the make command.

```
[PC ~/hello]$ make romfs
romfs-inst.sh /bin/hello
[PC ~/hello]$ ls ../uClinux-dist/romfs/bin/hello
hello
```

10.1.5. Creating the Image File

After the make romfs command, move to the uClinux-dist directory, execute the make image command, and a target board image file that includes hello will be generated in the image directory.

```
[PC ~/hello]$ cd ../uClinux-dist
[PC ~/uClinux-dist]$ make image
:
:
[PC ~/uClinux-dist]$ ls images
image.bin  linux.bin  romfs.image
```

For more information on the image target, refer to “7.7.5.image”.

10.1.6. Multiple Source Code Files

Executables with multiple source code files can be supported by making changes in the Makefile. The following example generates the executable hello from hello.c and print.c.

```
[PC ~/hello]$ ls
Makefile  hello.c  print.c
```

hello.c

```
#include <stdio.h>

extern void print_hello(char * string);

int main(int argc, char * argv[])
{
    print_hello("World");
    return 0;
}
```

print.c

```
void print_hello(char * string)
{
    printf("Hello %s\n", string);
}
```

Add print.o to OBJS in the Makefile.

```
# ROOTDIR=/usr/src/uClinux-dist
ifndef ROOTDIR
ROOTDIR=../uClinux-dist
endif
ROMFSDIR = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch

EXEC = hello
OBJS = hello.o print.o ----- Add "print.o"

all: $(EXEC)

$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

```
[PC ~/hello]$ make
:
:
[PC ~/hello]$ ls
Makefile  hello  hello.c  hello.gdb  hello.o  print.c  print.o
```

10.1.7. pthread Support

It is necessary to link the thread specific libraries for applications that use threads. The Makefile is changed as follows.

```
# ROOTDIR=/usr/src/uClinux-dist
ifndef ROOTDIR
ROOTDIR=../uClinux-dist
endif
ROMFSDIR = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch

EXEC = hello
OBJS = hello.o

all: $(EXEC)

$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LIBPTHREAD) $(LDLIBS) ----- ①

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

① Add “\$(LIBPTHREAD)”.

10.2. Product Specific Applications

It is almost a certainty that product specific applications will exist when dealing with embedded devices. This section introduces the method of integrating these applications into uClinux-dist.

10.2.1. Directory Creation

Make a directory for the application under
uClinux-dist/vendors/\$(CONFIG_VENDOR)/\$(CONFIG_PRODUCT).
Change \$(CONFIG_VENDOR) and \$(CONFIG_PRODUCT) to correspond to the board being used. Here, it is assumed that the vendor is AtmarkTechno, the product is SUZAKU and the application is hello.


```
[PC ~]$ mkdir uClinux-dist/vendors/AtmarkTechno/SUZAKU/hello
[PC ~]$ ls -d uClinux-dist/vendors/AtmarkTechno/SUZAKU/hello
hello
```

10.2.2. Preparing Source Code

The hello.c used in the preceding section is used for the C source code here. The Makefile is simpler than the one used for the “Out of Tree” compile.

```
EXEC = hello
OBSJS = hello.o

all: $(EXEC)

$(EXEC): $(OBSJS)
    $(CC) $(LD_FLAGS) -o $@ $(OBSJS) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
    $(CC) -c $(C_FLAGS) -o $@ $<
```

10.2.3. Application Configuration

The added application is then added to the product Makefile. The product Makefile contains the variable DIRS to which the directory name of the application is added.

```
:
:
DIRS = hello
:
:
```

10.2.4. Build

The build method is the same as the method described in “7.6 Build”. As the all, clean and romfs targets are defined as follows, the appropriate commands will be executed for all directories specified with DIRS.

```
all:
    dirs=$(DIRS); \
    for i in $$dirs; do $(MAKE) -C $$i clean || exit $? ; done

clean:
    -dirs=$(DIRS); \
    for i in $$dirs; do [ ! -d $$I ] || $(MAKE) -C $$i clean; done
```

```
romfs:
:
:
dirs=$(DIRS); \
for I in $$dirs; do $(MAKE) -C $$i romfs || exit $?; done
```

10.3. Merging to The user Directory

There are times when an application first developed for a specific product later becomes used by various different products. In such cases, the application can be shared between the products by moving it to the user directory.

10.3.1. Directory Creation

Create a directory for the application under uClinux-dist/user. Here, hello is created.

```
[PC ~]$ mkdir uClinux-dist/user/hello
[PC ~]$ ls -d uClinux-dist/user/hello
hello
```

10.3.2. Preparing Source Code

The same C source code and Makefile used in “10.2. Product Specific Applications” is used here.

10.3.3. Application Configuration

Changes are made in uClinux-dist/config/config.in and uClinux-dist/user/Makefile. In this example, the application is added to Miscellaneous Applications. It is entered in alphabetical order.

Example 10-1 uClinux-dist/config/config.in Changes

```
--- config.in.orig      2004-04-18 04:03:57.000000000 +0900
+++ config.in           2004-05-26 17:58:38.000000000 +0900
@@ -515,6 +515,7 @@
bool 'gdbreplay'        CONFIG_USER_GDBSERVER_GDBREPLAY
bool 'gdbserver'        CONFIG_USER_GDBSERVER_GDBSERVER
bool 'hd'               CONFIG_USER_HD_HD
+bool 'hello'           CONFIG_USER_HELLO_HELLO
bool 'lcd'              CONFIG_USER_LCD_LCD
bool 'ledcon'           CONFIG_USER_LEDCON_LEDCON
bool 'lilo'             CONFIG_USER_LILO_LILO
```

Example 10-2 uClinux-dist/user/Makefile Changes

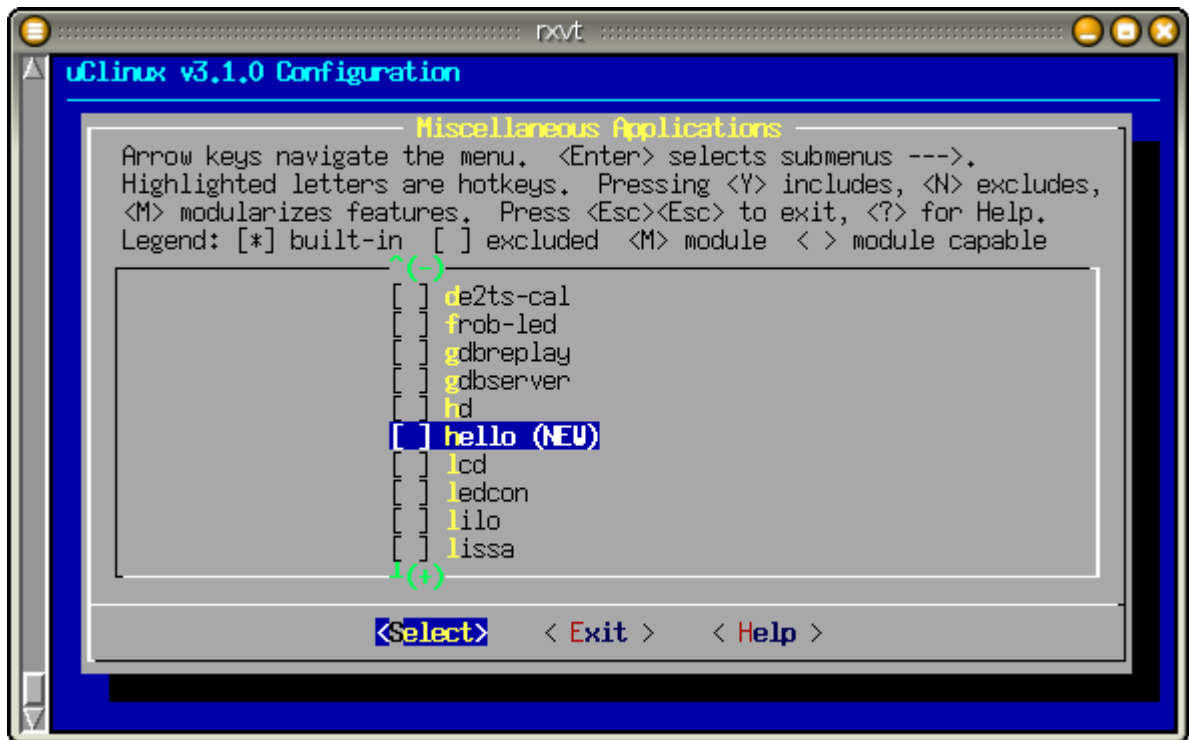
```
--- Makefile.orig       2004-02-20 13:22:55.000000000 +0900
+++ Makefile            2004-05-26 17:56:09.000000000 +0900
@@ -123,6 +123,7 @@
dir_$(CONFIG_USER_GDBSERVER_GDBSERVER) += gdbserver
dir_$(CONFIG_USER_GETTYD_GETTYD)       += gettyd
dir_$(CONFIG_USER_HD_HD)               += hd
+dir_$(CONFIG_USER_HELLO_HELLO)        += hello
```

```
dir_$ (CONFIG_USER_HOSTAP_HOSTAP)           += hostap
dir_$ (CONFIG_USER_HTTPD_HTTPD)             += httpd
dir_$ (CONFIG_USER_HWCLOCK_HWCLOCK)          += hwclock
```

10.3.4. Application Selection

Make sure that the application added with make menuconfig etc. is displayed in the Miscellaneous Applications section. Select the displayed hello and save the settings.

Figure 10-1 hello Added to the Menu



10.3.5. Build

The “In Tree” compiling build is the same as the method described in “7.6 Build”. Transfer the generated image file to the target board and make sure that hello runs properly.

10.3.6. Configuration Naming Rule

The configuration name used here is CONFIG_USER_HELLO_HELLO.

In uClinux-dist, the directory and applications names are used for Userland application selection.

- All configuration options begin with CONFIG_.
- Those under uClinux-dist/user begin with CONFIG_USER_.
- As the directory name is hello, the option begins with CONFIG_USER_HELLO_.
- Then after adding the application name, it becomes “CONFIG_USER_HELLO_HELLO”.

10.3.7. Multiple Applications

The following shows how to develop multiple applications in one directory. This is not limited to just “In Tree” compiling, and therefore can be applied to “Out of Tree” compiling as well.

Here, the application named hello2 is added to the uClinux-dist/user/hello directory, with the configuration character string being CONFIG_USER_HELLO_HELLO2.

First are the changes to uClinux-dist/config/config.in and uClinux-dist/user/Makefile as shown below.

Example 10-3 Changes to uClinux-dist/config/config.in (Multiple Applications)

```
--- config.in.orig      2004-04-18 04:03:57.000000000 +0900
+++ config.in           2004-05-26 17:58:38.000000000 +0900
@@ -515,6 +515,8 @@
 bool 'gdbreplay'        CONFIG_USER_GDBSERVER_GDBREPLAY
 bool 'gdbserver'        CONFIG_USER_GDBSERVER_GDBSERVER
 bool 'hd'               CONFIG_USER_HD_HD
+bool 'hello'            CONFIG_USER_HELLO_HELLO
+bool 'hello2'           CONFIG_USER_HELLO_HELLO2
 bool 'lcd'              CONFIG_USER_LCD_LCD
 bool 'ledcon'           CONFIG_USER_LEDCON_LEDCON
 bool 'lilo'             CONFIG_USER_LILO_LILO
```

Example 10-4 Changes to uClinux-dist/user/Makefile (Multiple Applications)

```
--- Makefile.orig       2004-02-20 13:22:55.000000000 +0900
+++ Makefile            2004-05-26 17:56:09.000000000 +0900
@@ -123,6 +123,8 @@
 dir_$(CONFIG_USER_GDBSERVER_GDBSERVER) += gdbserver
 dir_$(CONFIG_USER_GETTYD_GETTYD)       += gettyd
 dir_$(CONFIG_USER_HD_HD)               += hd
+dir_$(CONFIG_USER_HELLO_HELLO)         += hello
+dir_$(CONFIG_USER_HELLO_HELLO2)        += hello
 dir_$(CONFIG_USER_HOSTAP_HOSTAP)       += hostap
 dir_$(CONFIG_USER_HTTPD_HTTPD)         += httpd
 dir_$(CONFIG_USER_HWCLOCK_HWCLOCK)     += hwclock
```

In the Makefile, the configuration option is written on the left and the directory name on the right of the “+=”. Therefore, the directory name specified for hello and hello2 will be the same hello.

Next is uClinux-dist/user/hello/Makefile. This time, the application name is not assigned to a variable, but handled directly.

Example 10-5 Makefile (Multiple Applications)

```
OBJS_HELLO = hello.o
OBJS_HELLO2 = hello2.o

all: hello hello2

hello: $(OBJS_HELLO)
    $(CC) $(LDFLAGS) -o $@ $(OBJS_HELLO2) $(LDLIBS)

hello2: $(OBJS_HELLO2)
    $(CC) $(LDFLAGS) -o $@ $(OBJS_HELLO2) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) -e CONFIG_USER_HELLO_HELLO /bin/hello
    $(ROMFSINST) -e CONFIG_USER_HELLO_HELLO2 /bin/hello2

%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

Take note of the point that `-e CONFIG_USER_HELLO` is used in the `romfs` target.

The uClinux-dist build system is set to build all applications in the `hello` directory when any one of the applications there is selected. Therefore, the application to be installed must be selected using a condition in the `romfs` target. As described above, the application to be installed can be determined by using a conditional branch configuration option.

For a detail explanation on `romfsinst.sh`, refer to the “9. romfs Installation Tool”.

11. Using Multiple Kernels

It is also possible to develop using multiple versions of Linux Kernel source code. This can be achieved by changing over a symbolic link as shown below.

```
[PC ~]$ ls
kernel/      uClinux-dist/
[PC ~]$ ls kernel
linux-foo/    linux-bar/
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ ln -s ../kernel/linux-foo linux-2.4.x
[PC ~/uClinux-dist]$ ls -l linux-2.4.x
lrwxrwxrwx      linux-2.4.x -> ../kernel/linux-foo
```

12. Specific Applications

uClinux-dist contains applications not found in standard Linux. This chapter introduces applications that are especially suitable for use with embedded equipment.

12.1. NetFlash

The NetFlash application is used to download image files via the network and write them to Flash memory.

When using embedded systems, the ability to carry out system upgrades via the network brings great advantages, both for maintenance and user convenience. Using NetFlash, this can be done relatively simply.

NetFlash can use the http, ftp and tftp protocols for file downloads. Therefore, a server that can transmit using one of these protocols is required to run NetFlash.

When no options are specified when executing NetFlash, the process of writing to Flash memory follows these steps.

1. All processes are ended
2. The specified file is downloaded via the specified protocol from the server
3. The downloaded file's checksum is verified
4. The file is written to Flash memory
5. The system rebooted

These processes can be finely controlled by specifying the appropriate options. A command execution example and NetFlash's help are shown below for reference.

```
[Target /]# netflash http://embedded-server/images/base.img
netflash: killing tasks...
...
netflash: got "http://myserver.local/images/image.bim", length=4194304
:
:
```

```
[Targt /]# netflash -h
usage: netflash [-bCfFhijklntuv?] [-c <console-device>] [-d <delay>]
[-o <offset>] [-r <flash-device>] [<net-server>] <file-name>

-b      don't reboot hardware when done
-C      check that image was written correctly
-f      use FTP as load protocol
-F      force overwrite (do not preserve special regions)
-h      print help
-i      ignore any version information
-H      ignore hardware type information
-j      image is a JFFS2 filesystem
-k      don't kill other processes
        (ignored when root filesystem is inside flash)
-l      lock flash segments when done
```

-n	file with no checksum at end (implies no version information)
-p	preserve portions of flash segments not actually written.
-s	stop erasing/programming at end of input data
-t	check the image and then throw it away
-u	unlock flash segments before programming
-v	display version number

Process is interrupted without writing it into Flash memory, when error occurs during processing.

There is a possibility that system won't start up in the worst case when power outage occurs during writing. Therefore, enough caution is required when NetFlash is executed,

12.2.Flatfsd

One often requested function of embedded devices is the ability to upgrade the firmware without resetting any configuration information. In order to make this possible, it is common to build a file system allocated with a small memory area to hold the user settings.

The Flat Filesystem is well suited to this usage. The file system can be constructed from a minimum of just one sector, it is simply built and has proven stability.

The flatfsd program is used to produce these file systems.

flatfsd reads and writes the contents of the /etc/config directory mounted in ramfs to and from the /dev/flash/config device file. /dev/flash/config is created by specifying the major and minor numbers of the device used to store the configuration files.

```
[Target /]# ls -l /dev/flash/
crw----- 1 root    root      90,  14 Jan  1 09:00 config
crw----- 1 root    root      90,   6 Jan  1 09:00 image
[Target /]#
```

The flatfsd -r command is executed to restore the previously saved configuration file data.

This command reads the file data previously saved in /dev/flash/config, and copies it to /etc/config.

```
[Target /]# flatfsd -r
FLATFSD: created 6 configuration files (507 bytes)
[Target /]#
```

At this time, the consistency of the file data is verified with the stored checksum, and if there is any abnormality the /etc/config directory is rewritten using the contents of /etc/default. The flatfsd -r command is usually executed when the system is booted.

To save the changes made to the configuration files to Flash memory, the SIGUSR1 signal is sent to the flatfsd process. Therefore it is necessary to start the process by executing the flatfsd command when the system is booted. The process ID of the activated flatfsd can be acquired from the /var/run/flatfsd.pid file. The following example shows how to send the signal to flatfsd using the killall command.

```
[Target /]# killall -USR1 flatfsd
[Target /]#
```


Once the flatfsd process receives the SIGUSR1 signal, it writes the contents of the /etc/config directory to /dev/flash/config. At this time, the checksum is calculated and recorded to allow for verification of file data consistency.

When the flatfsd application is selected in the config of uClinux-dist, applications such as dhcpcd and passwd create their configuration files in /etc/config instead of /etc. Therefore, modified settings are preserved while the system is rebooted.

It is possible that saved data may be lost when flatfsd is writing data to the /dev/flash/config device and a power cut occurs.

Also note that the guaranteed number of times data can be written to Flash memory is around 100,000.

Revision History

Ver.	Revision Date	Description of Revision
1.0	2004/12/18	First edition Released
1.1	2004/12/28	<ul style="list-style-type: none">• Added notes concerning user privileges at build in “3. Default Image Build”• Changed the description of the link in “10.3.5 Build”

