S60

iMaker User Guide

Change history

Version	Date	Status	Comments
2.0	18.02.2009	Approved	Editorial changes
1.0	11.02.2008	Approved	

Contents

1 About this document

This document is a user guide for the iMaker ("image-Make-r") tool. These pages describe the main use aspects of the iMaker command line tool and interface.

2 What is iMaker?

2.1 Introduction

iMaker is an ROM-image creation tool which provides a simple, standardized and configurable ROM-image creation framework. iMaker is based on the standardized GNU Make system and is therefore platform-independent. iMaker is a tool that creates a flash image (see flash image on page 43) from a set of Symbian binary and data files. iMaker mainly functions on top of the Symbian buildrom utility. Symbian buildrom already offers the core functionality, but with the increasing demand of configuring devices on ROM building it is not enough. The benefit of having configuration choices at ROM creation time saves time (no need to build).

The iMaker tool itself runs the Make tool and consists of thin layer of Perl.

iMaker offers a standardized framework for defining configuration parameters for the ROM-image creation. The framework tools and configurability can easily be extended in the end customer interface, without changing the core iMaker functionality.

2.2 Features of iMaker

iMaker has the following features:

- Possibility to create different types of flash images easily: PRD/RND/SubCon/MiniOS/FOTA/ODP/ userdisk etc.
- *Revlon_conf.exe* is not used anymore. Therefore, *x_conf*s are no longer used.
- iMaker adds BB5 image headers to the image.
- Fully configurable ROM-image creation framework, easy to add new platform/product support.
- ROM-image size is dynamically calculated by iMaker. Only the start address needs to be configured, no hard-coded values are needed.
- In-built Multi-ROFS support.
- All non-standard configuration files (**.ini*, **.cmi*, **.conf*, **.txt*) are no longer needed; they are replaced by a single product-specific makefile.
  - Configuration is always done in the same format, i.e. makefile format.
- Language-variant image creation support.
- Parallel ROM-image creation support: multiple variant images can be created in parallel.
- Platform-independence: also Linux is supported.
- Possibility to use external tools with iMaker.
  - Extending the functionality is easy.

2.3 Requirements

The iMaker tool is intended to be executed in the S60 environment with the normal system requirements of S60. iMaker is a part of the S60 asset and is installed as a part of S60 build.

2.3.1 External tool dependencies

- Symbian tools (verified with Symbian 9.3 and Symbian 9.4 toolchains)
  - *buildrom.pl*
- Perl (v5.6.1, tested to work also with v5.8.8)

3 General concepts

The iMaker tool evolves around the GNU Make and includes therefore all the functionality of Make. The iMaker main concepts are very similar to Make which you can study more on the GNU Make reference.

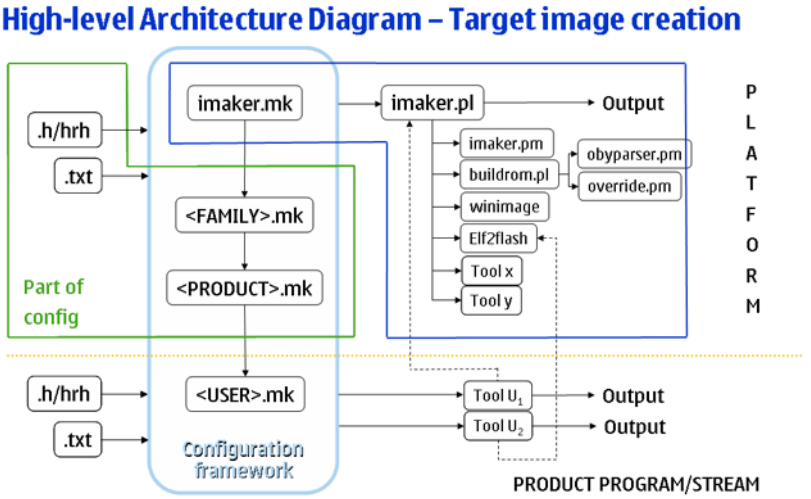

Figure 1: High-level architecture

3.1 iMaker configuration files

3.1.1 Makefile = configuration file

iMaker is a set of makefiles that can be considered as iMaker configuration files. All configuration files in iMaker are makefiles.

3.1.2 Importing other configuration files

In addition to the iMaker makefiles, iMaker can import data from other files, but for this a make routine must be written. In the high level architecture diagram an **.hrh* file is shown as a possible input file. In practice, this means that iMaker can import a set of #define(s) to iMaker as makefile variables.

Example: Import *aaa.hrh* to iMaker. This makes the settings inside *aaa.hrh* visible as iMaker variables.

```
aaa.hrh:                iMaker variable:
#define AAA 1    = >    AAA=1
```

3.2 iMaker configuration elements

The makefiles in iMaker can contain three types of configuration elements: variables, targets and steps. The variables and targets are normal makefile concepts and their documentation can be found in the GNU Make reference. The step is an iMaker specific concept.

3.2.1 iMaker variables

The iMaker variables are always makefile variables. Therefore, the variable concept always has typical make like mechanisms (set, override and append) that can be applied to it. This means that every variable in the default configuration can be overridden or extended in the end user's own configuration file and command line (see the example on overriding variables on page 32).

See the GNU Make reference Using Variables for more general information on makefile variables.

For a list of available variables in the iMaker default configuration, see the variables in iMaker API on page 23.

3.2.2 iMaker targets

iMaker targets are basically normal Make targets, but the target usage is different from the normal makefile usage. An iMaker target does not represent a target file or any output name which should be created. The iMaker targets are so called phony targets which actually link together a sequence of iMaker steps that are executed when the target is run.

3.2.3 iMaker step

The iMaker step is a sequence of actions that take place when the step is executed. An action is a single command to the system. Each step can contain CLEAN and BUILD substeps. When the step is executed it first runs the CLEAN step and then the BUILD step. This is to offer a possibility to remove old files before generating new ones, with a clear procedure.

A step is basically defined as makefile variable. The step definition can contain the CLEAN and BUILD substeps, but both of them are not required (i.e. the step definition can contain only the BUILD step).

The format of the step definition is:

- [CLEAN|BUILD]_<STEPNAME> = [command | [param]+]+

The step can be referred to with <STEPNAME> . The commands are internal iMaker commands on page 16 that make the system calls platform-independent.

3.2.3.1 Example step ROFS2VER

```
CLEAN_ROFS2VER = del | $(ROFS2_VERIBY) | del | $(ROFS2_FWIDFILE)
BUILD_ROFS2VER = /
  echo  | Generating ROFS2 version file(s)/n | /
  write | $(ROFS2_VERIBY) | /
    // Generated version file(s) for ROFS2 image creation/n/n /
    $(call iif,$(ROFS2_ROMVER),version=$(ROFS2_ROMVER)/n) /
    $(call iif,$(USE_FOTA),$(call _includever,$(ROFS2_FWIDFILE),fwid2.txt))
| /
  $(call iif,$(USE_FOTA),writeu | $(ROFS2_FWIDFILE) | id=$(ROFS2_FWID)/
nversion=$(ROFS2_FWIDVER)/n)
```

The step (ROFS2VER) defines a clean step, where it deletes two files. The files are given as makefile variables. E.g. $(ROFS2_VERIBY)

```
CLEAN_ROFS2VER = del | $(ROFS2_VERIBY) | del | $(ROFS2_FWIDFILE)
```

Then, it also defines a BUILD step, where it echoes a notification (to the screen and log),

```
echo  | Generating ROFS2 version file(s)/n | /
```

writes an IBY file (ROFS2_VERIBY)

```
write | $(ROFS2_VERIBY) | /
    // Generated version file(s) for ROFS2 image creation/n/n /
    $(call iif,$(ROFS2_ROMVER),version=$(ROFS2_ROMVER)/n) /
    $(call iif,$(USE_FOTA),$(call _includever,$(ROFS2_FWIDFILE),fwid2.txt))
| /
```

and, conditionally, writes also an *fwid2* file.

```
$(call iif,$(USE_FOTA),writeu | $(ROFS2_FWIDFILE) | id=$(ROFS2_FWID)/nversion=
$(ROFS2_FWIDVER)/n)
```

The conditional parts are implemented as normal makefile calls that check the Boolean value of a makefile variable. Therefore any makefile mechanism can be used in the iMaker steps to achieve the needed end result.

3.3 iMaker OBY conversion file format

iMaker contains the possibility to configure ROM/ROFS1 split and debug binaries (from the UDeb folder) via text file format (**.txt*). The text file format is the same for all of these files.

3.3.1 Format definition

The text file contains a list of file names or patterns delimited with end of line. The definitions in this file content try to match the target's file names in IBY/OBY files. E.g. in the following example, *aaa.dll* in the text file would match the */sys/bin/**aaa.dll*** in the IBY entry.

Example 1: Text file example

```
# Comment
aaa.dll
bbb.exe
ccc.dll
*.ldd
```

Example 2: IBY file match

```
/epoc32/release/armv5/urel/aaa.dll  /sys/bin/aaa.dll
```

The system does not report any error if a match is not found. This means that *bbb.exe* can exist in Example 1 without any errors even if it is not in the ROM image content at all.

4 iMaker files

iMaker contains the following files:

- *imaker.pl,* thin Perl front-end for iMaker internal commands
- *imaker.pm*, the iMaker commands Perl module
- *imaker.mk*, the common iMaker base configuration
- *imaker_readme.txt*, Readme file
- *mingw_make.exe*, the make EXE
- *changelog.txt*, changelog lists the changes for each release
- *imaker_version.txt*, iMaker version number

4.1 Internal tools inside iMaker

- Symbian Integration Toolkit (SITK)
  - *imgcheck.exe* (for checking the static dependencies between binaries' separate MultiROFS images)
  - *readimage.exe* (for extracting files from an existing image file)

4.2 Buildrom plug-ins

The main tool iMaker uses is Symbian buildrom. The iMaker tool contains in itself a set of plug-in modules to the buildrom tool, because some functionality needs to be executed inside the buildrom tool itself. The plug-ins are located in the buildrom_plugins project.

4.2.1 Buildrom plug-in obyparse

4.2.1.1 Description

General purpose buildrom plug-in for parsing IBY files. The current functionality contains two features:

- Splitting core image contents to ROM and ROFS1 sections
- Forcing components to UDeb/URel versions

4.2.1.2 Syntax

obyparse_<function><target>

function:rom|rofs1|udeb|urel

Target: _include <configfile>| <component>

<configfile> contains 0 or more <component>s. It can also include other files:

#include <relativeconfigfile>

<relativeconfigfile> is like <configfile> but its directory path must be relative to <configfile>'s path. Empty lines or lines, other than #include, starting with # are skipped.

4.2.1.3 Example

my.oby:

```
externaltool=obyparse                                    // Take the obyparse
plugin into use
obyparse_rofs1 *                                        // Put all components
to ROFS1
obyparse_rom_include   /epoc32/rombuild/romfiles.txt    // Put components in
romfiles.txt to ROM
obyparse_rofs1_include /epoc32/rombuild/rofsfiles.txt   // Put components in
```

```
rofsfiles.txt to ROFS1
obyparse_udeb *                                          // Take UDEB versions
of all the components into use (full UDEB)
```

/epoc32/rombuild/romfiles.txt:

```
# Extra ROM files
#include my_romfiles.txt

component1.dll
component?.exe
...
```

Output snippet from a buildrom command:

```
obyparse.pm: Reading /epoc32/rombuild/romfiles.txt
obyparse.pm: Reading /epoc32/rombuild/my_romfiles.txt
obyparse.pm: Reading /epoc32/rombuild/rofsfiles.txt
obyparse.pm: Processing...
obyparse.pm: Duration: 5 seconds
```

4.2.2 Buildrom plug-in override

Buildrom plug-in enabler for overriding file and data entries from platform IBY files in product and variant creation. The main reason for using override plug-in is to reduce the need for IBY file branching. Therefore, instead of branching a large number of IBY files owned by platform or product, it is recommended to create a single IBY file that overrides the needed files.

4.2.2.1 Example 1: Replacing an existing file with a different source file

This replaces the original line with the override line. Note: it is obligatory to define the ROM_IMAGE section for the overrides correctly (this is needed for the error case checking).

Some *platform.iby*

```
data=file.txt         sys/bin/file.txt // In ROM_IMAGE[2]
```

product.iby

```
ROM_IMAGE[2] data-override=file_product.txt        sys/bin/file.txt
```

output

```
data=file_product.txt         sys/bin/file.txt
```

4.2.2.2 Example 2: Removing an existing file from a platform IBY

This deletes the original line from the IBY structure.

Some *platform.iby*

```
data=file.txt         sys/bin/file.txt // In ROM_IMAGE[3]
```

product.iby

```
ROM_IMAGE[3] {
data-override=empty   sys/bin/file.txt
}
```

output

```
REM OVERRIDE data=file_product.txt    sys/bin/file.txt
```

4.2.3 Buildrom plug-in LOCALISE

4.2.3.1 Description

Adds a LOCALISE macro to the IBY files, which enables configuration of localized resource files. The localized language selection is done with the ADD_LANGUAGE macro.

Syntax: LOCALISE

- type=LOCALISE(source, target[, languages])
  - source: The source file. The section that needs to be localized should be marked with "**??**".
  - target: The target file. The section that needs to be localized should be marked with "**??**".
  - languages: A space delimited list of language codes.

Syntax: ADD_LANGUAGE

- ADD_LANGUAGE lang

4.2.3.2 Example

Example: Add languages to some included IBY.

```
ADD_LANGUAGE 01
ADD_LANGUAGE 02
ADD_LANGUAGE 03
```

Localize an Application resource file.

```
data=LOCALISE(APP_RESOURCE_DIR/App.r??, RESOURCE_DIR/app.r??)
```

Output:

```
data=LOCALISE(APP_RESOURCE_DIR/App.01, RESOURCE_DIR/app.r01)
data=LOCALISE(APP_RESOURCE_DIR/App.02, RESOURCE_DIR/app.r02)
data=LOCALISE(APP_RESOURCE_DIR/App.03, RESOURCE_DIR/app.r03)
```

5 Internal commands

The iMaker commands are system calls that offer a platform-independent layer for executing iMaker steps. An iMaker command can require parameters which are given as a pipe (|) separated list after the command (e.g. command | param | param).

5.1 File system commands

The file system commands execute a change to the file system, to a directory or a file.

5.1.1 cd | dir

Changes the current working directory to the given directory.

Parameters:

- dir: The directory to be changed.

```
Example:
cd | /epoc32/rom
```

5.1.2 copy | source | destination

Copy the given source file to the destination file. Parameters:

- source: The source file(s). Asterisks can be used.
- destination: The destination file or folder.

```
Examples:
copy | /epoc32/rom/tmp1.oby | $(WORKDIR)/tmp1.oby
copy | /epoc32/rom/tmp?.oby | $(WORKDIR)/
```

5.1.3 del | file

Delete the given file. Parameters:

- file: The target file to be deleted.

```
Examples:
del | /epoc32/rom/tmp1.oby
```

5.1.4 mkdir | dir

Create the given directory. Creates the given dir and all the directories in between if they do not exist. E.g. `/example/foobar/com` creates the entire given directory tree. Parameters:

- dir: The target directory to be created.

```
Examples:
mkdir | /epoc32/rombuild/$(PRODUCT)/core => creates a core directory to the
product folder where the $(PRODUCT) is defined as a makefile variable.
```

5.1.5 move | source | target

Move the given source file to the destination file. Parameters:

- source: The source file(s). Asterisks can be used.
- destination: The destination file or folder.

```
Examples:
move | /epoc32/rom/tmp1.oby | $(WORKDIR)/tmp1.oby
move | /epoc32/rom/tmp?.oby | $(WORKDIR)/
```

5.1.6 test | target

Test whether the given target file exists. The execution of iMaker stops if the test fails. Parameters:

- target: The target file to be checked.

```
Examples:
test | $(WORKDIR)/tmp1.oby
```

5.1.7 touch | file

Touch the given target file. In other words, change the time stamp to the current system time. Parameters:

- file: The target file to be touched.

```
Examples:
touch | $(WORKDIR)/tmp1.oby
```

5.1.8 type | file

Print the contents of the given file to the screen. Parameters:

- type: The target file to be printed.

```
Examples:
type | $(WORKDIR)/tmp1.oby
```

5.1.9 typeb | file

A binary file format overload of the type method.

5.1.10 typeu | file

A unicode file format overload of the type method.

5.1.11 write | file | data

Write the given data to the given file. Parameters:

- file: The target file where the data is written to.
- data: The data to be written.

```
Examples:
write | $(WORKDIR)/foobar.oby | // Hello foobar
```

5.1.12 writeb

A binary file format overload of the write method. Writes a binary file.

5.1.13 writeu

A unicode file format overload of the write method. Writes a unicode file which is needed if the data is read inside the device.

- Writes a normal unicode prefix to the file (FFFE).

5.1.14 headb | source | target | length

Similar to copy, but copies only the first n bytes, given with the parameter length. Parameters:

- source: The source file which is read.
- target: The target file where the data is copied to.

- length: The number of bytes that are copied from the beginning of the source file.

```
Examples:
headb| $(WORKDIR)/foo.bin | $(WORKDIR)/bar.bin | 10
```

5.1.15 tailb | source | target | length

Similar to copy but copies only the last n bytes, given with the parameter length. Parameters:

- source: The source file which is read.
- target: The target file where the data is copied to.
- length: The number of bytes that are copied from the end of the source file.

```
Examples:
headb| $(WORKDIR)/foo.bin | $(WORKDIR)/bar.bin | 10
```

5.1.16 imghdr | sourcefile | hdrfile | hdrstr | hdrsize | align

Create a file with an image header.

1 Create a new file with an image header.

2 Copy the contents of the source file to the end.

Similar to copy but copies only the last n bytes, given with the parameter length. Parameters:

- sourcefile: The source file which is read.
- hdrfile: The target file which is created.
- hdrstr: The header data which is first written to the hdrfile.
- hrdsize: The header section size.
- align: The alignment of the header section.

5.2 Verbose commands

5.2.1 echo | text

Echo (print) the given text parameter to the log and screen (stdout). Parameters:

- text: The message string that is echoed.

```
Examples:
echo | hello world!
```

5.2.2 echo[BITMASK]

Same as normal echo, but with a bitmask field for filtering out verbose messages. All echo values are printed to the log file, but only echo/echo1 (which actually are the same thing) are printed to the stdout (screen). The output to the screen can be controlled with the VERBOSE variable (see iMaker API on page 20). Parameters:

- text: The message string that is echoed.

```
Examples:
echo2 | hello world!
```

5.2.3 error | text ¶

Print an error message. Parameters:

- text: The message string that is echoed.

```
Examples:
error | this is an error!
```

5.2.4 warning | text

Print a warning message. Parameters:

- text: The message string that is echoed.

```
Examples:
warning | this is an error!
```

5.3 System commands

5.3.1 cmd | command

Execute a system shell command and pass the output to the screen (normal stdout). Parameters:

- command: The command string.

```
Examples:
cmd | perl maksym.pl $(ROM_LOG)
```

5.3.2 cmdtee | command | file

Execute a system shell command and pass the output to screen and file. Parameters:

- command: The command string.
- file: The file where the command output is also directed.

```
Examples:
cmdtee | perl maksym.pl $(ROM_LOG) | test.txt
```

5.3.3 parse | title | regexp

Parse the *buildrom.pl* output with the given keywords and print the given lines separately to log. Parameters:

- title: The title for the found data lines.
- regexp: A regular expression that the parse routine tries to match against the *buildrom.pl* log.

```
Examples:
parse | /nMissing file(s):/n | Missing file:
```

5.4 Other general commands

5.4.1 toolchk

Check if a required tool exists. Parameters:

- tool: The tool name.
- tool version command: The command that is needed to get the version.
- version regexp: A regular expression that tries to match with the version string.

```
Examples:
toolchk | $(MAKE)                    | $(MAKE) -v          | GNU Make
(?:version )?(.+?)(?:,.+)?$$ | /
```

6 iMaker API

6.1 iMaker targets

6.1.1 all

- Type: Target
- Description: Create all image sections and symbol files.

6.1.2 clean

- Type: Target
- Description: Clean all target files.

6.1.3 core

- Type: Target
- Description: Create the core image (ROM, ROFS1).

6.1.4 core-image

- Type: Target
- Description: Create the core image files (*rom.img*, *rofs1.img*).

6.1.5 flash

- Type: Target
- Description: Create all image section files. No symbol files.

6.1.6 flash-all

- Type: Target
- Description: Create all image sections and symbol files.

6.1.7 help-%

- Type: Target
- Description: Print help on help items matching the pattern.

6.1.8 help-%-list

- Type: Target
- Description: Print a list of help items matching the pattern.

6.1.9 help-config

- Type: Target
- Description: Print a list of available configurations in the current working environment.

6.1.10 help-target

- Type: Target
- Description: Print help on all targets (same as help-target-*).

6.1.11 help-target-%

- Type: Target
- Description: Print help on targets matching the pattern.

6.1.12 help-target-%-list

- Type: Target
- Description: Print a list of targets matching the pattern.

6.1.13 help-target-%-wiki

- Type: Target
- Description: Print help in wiki format on targets matching the pattern.

6.1.14 help-variable

- Type: Target
- Description: Print help on all variables (same as help-variable-*).

6.1.15 help-variable-%

- Type: Target
- Description: Print help on variables matching the pattern.

6.1.16 help-variable-%-all

- Type: Target
- Description: Print full help on variables matching the pattern.

6.1.17 help-variable-%-list

- Type: Target
- Description: Print a list of variables matching the pattern.

6.1.18 help-variable-%-value

- Type: Target
- Description: Print a list of variables with values matching the pattern.

6.1.19 help-variable-%-wiki

- Type: Target
- Description: Print help in wiki format on variables matching the pattern.

6.1.20 image

- Type: Target
- Description: Create only the image file(s) (**.img*).

6.1.21 print-*

- Type: Target
- Description: Print the value of the given variable to the screen.

6.1.22 release

- Type: Target
- Description: Run the RELEASE step that copies the created image files to the release folder. (See RELEASEDIR on page 27, RELEASEFILES on page 27 and RELEASENAME on page 27).

6.1.23 rofs2

- Type: Target
- Description: Create the ROFS2 image.

6.1.24 rofs2-image

- Type: Target
- Description: Create the ROFS2 image file (*rofs2.img*).

6.1.25 rofs3

- Type: Target
- Description: Create the ROFS3 image.

6.1.26 rofs3-image

- Type: Target
- Description: Create the ROFS3 image file (*rofs3.img*).

6.1.27 romsymbol

- Type: Target
- Description: Create the ROM symbol file.

6.1.28 step-*

- Type: Target
- Description: Generic target to execute any step inside the iMaker configuration. Any step (e.g. BUILD_*,CLEAN_*) can be executed with step-STEPNAME. An example: step-ROFS2PRE executes the CLEAN_ROFS2PRE and BUILD_ROFS2PRE commands.

6.1.29 toolinfo

- Type: Target
- Description: Print information about the tool.

6.1.30 variant

- Type: Target
- Description: Create the variant image (ROFS2, ROFS3).

6.1.31 variant-image

- Type: Target
- Description: Create the variant image files (*rofs3.img*,*rofs3.img*).

6.1.32 version

- Type: Target

- Description: Print the version information.

6.2 iMaker variables

6.2.1 BLDROBY

- Type: Variable
- Description: For passing extra OBY files (from command line) to *buildrom.pl*.
- Values: (string)

6.2.2 BLDROM_OPT

- Type: Variable
- Description: The default *buildrom.pl* options.
- Values: (string)

6.2.3 BLDROPT

- Type: Variable
- Description: For passing extra parameters (from command line) to *buildrom.pl*.
- Values: (string)

6.2.4 CONFIGROOT

- Type: Variable
- Description: Define the default configuration root directory.
- Values: (string)

6.2.5 CORE_CDPROMFILE

- Type: Variable
- Description: The name of the core Code Demand Paging ROM file (Code paging).
- Values: (string)

6.2.6 CORE_DIR

- Type: Variable
- Description: The working directory, when creating a core image.
- Values: (string)

6.2.7 CORE_FWIDFILE

- Type: Variable
- Description: The (generated) *_core_fwid.txt* file name.
- Values: (string)

6.2.8 CORE_IMEISVFILE

- Type: Variable
- Description: The (generated) *_core_imeisv.txt* file name.
- Values: (string)

6.2.9 CORE_IMEISVINFO

- Type: Variable
- Description: The content string for the *imeisv.txt* file.
- Values: (string)

6.2.10 CORE_MODELFILE

- Type: Variable
- Description: The (generated) *_core_model.txt* file name.
- Values: (string)

6.2.11 CORE_MODELINFO

- Type: Variable
- Description: The content string for the *model.txt* file.
- Values: (string)

6.2.12 CORE_MSTOBY

- Type: Variable
- Description: The generated master OBY file name which includes the CORE_OBY files.
- Values: (string)

6.2.13 CORE_NAME

- Type: Variable
- Description: The full name of the core image.
- Values: (string)

6.2.14 CORE_NDPROMFILE

- Type: Variable
- Description: The name of the core Non-demand Paging ROM file.
- Values: (string)

6.2.15 CORE_OBY

- Type: Variable
- Description: The OBY file(s) included to the core image creation.
- Values: (string)

6.2.16 CORE_ODPROMFILE

- Type: Variable
- Description: The name of the core On Demand Paging ROM file (ROM paging).
- Values: (string)

6.2.17 CORE_OPT

- Type: Variable
- Description: The core specific buildrom options.

- Values: (string)

6.2.18 CORE_PLATFILE

- Type: Variable
- Description: The (generated) *_core_platform.txt* file name.
- Values: (string)

6.2.19 CORE_PLATINFO

- Type: Variable
- Description: The content string for the *fwid.txt* file.
- Values: (string)

6.2.20 CORE_PRODFILE

- Type: Variable
- Description: The (generated) *_core_product.txt* file name.
- Values: (string)

6.2.21 CORE_ROFSFILE

- Type: Variable
- Description: The name of the core ROFS file.
- Values: (string)

6.2.22 CORE_ROMVER

- Type: Variable
- Description: The ROM version parameter passed to *version.iby*.
- Values: (string)

6.2.23 CORE_SWVERFILE

- Type: Variable
- Description: The (generated) *_core_sw.txt* version file name. This generated file is included in the CORE_VERIBY file.
- Values: (string)

6.2.24 CORE_SWVERINFO

- Type: Variable
- Description: The content string for the *sw.txt* file.
- Values: (string)

6.2.25 CORE_TIME

- Type: Variable
- Description: The time defined to the core image.
- Values: (string)

6.2.26 CORE_UDEBFILE

- Type: Variable

- Description: The name of the core UDeb file. See USE_UDEB.
- Values: (string)

6.2.27 CORE_VERIBY

- Type: Variable
- Description: The name of the generated core **version.iby* which includes version files and information.
- Values: (string)

6.2.28 CORE_VERSION

- Type: Variable
- Description: The version of the core. Used in *sw.txt* generation.
- Values: (string)

6.2.29 DEFAULT_LANGUAGE

- Type: Variable
- Description: The default language is the language where the device boots to (SIM language overrides this selection).
- Values: (string)

6.2.30 KEEPTEMP

- Type: Variable
- Description: Keep the *buildrom.pl* temp files (copied to WORKDIR). E.g. *tmp1.oby tmp2.oby...tmp9.oby*
- Values: ([0|1])

6.2.31 LABEL

- Type: Variable
- Description: A label to the NAME of the image.
- Values: (string)

6.2.32 LANGID

- Type: Variable
- Description: Language ID used in the *lang.txt* generation.
- Values: (string)

6.2.33 LANGUAGES

- Type: Variable
- Description: Languages are the languages that are taken to the image (SC language defaults to 01 in *languages.txt*).
- Values: (string)

6.2.34 NAME

- Type: Variable
- Description: The name of the image.

- Values: (string)

6.2.35 RELEASEDIR

- Type: Variable
- Description: RELEASEDIR is used in the RELEASE step that copies the created files to a separate folder defined by RELEASEDIR.
- Values: (string)

6.2.36 RELEASEFILES

- Type: Variable
- Description: RELEASEFILES define the source files that are copied in the RELEASE step. The files must be given as a space delimited list. Wild cards can be used. E.g. */epoc32/rombuild/product/theflash.fpsx /epoc32/rombuild/product/*.log*
- Values: (string)

6.2.37 RELEASENAME

- Type: Variable
- Description: The RELEASENAME is used in the RELEASE step that copies the created files to a separate folder with a new name.
- Values: (string)

6.2.38 ROFS2_DIR

- Type: Variable
- Description: The working directory when creating the ROFS2 image.
- Values: (string)

6.2.39 ROFS2_FOOTER

- Type: Variable
- Description: This variable can contain a footer section for the ROFS2 master OBY.
- Values: (string)

6.2.40 ROFS2_FWIDFILE

- Type: Variable
- Description: The (generated) *_rofs2_fwid.txt* file name.
- Values: (string)

6.2.41 ROFS2_FWIDINFO

- Type: Variable
- Description: The content string for the *fwid2.txt* file.
- Values: (string)

6.2.42 ROFS2_HEADER

- Type: Variable
- Description: This variable can contain a header section for the ROFS2 master OBY.
- Values: (string)

6.2.43 ROFS2_MSTOBY

- Type: Variable
- Description: The (generated) ROFS2 master OBY file name. This file includes the ROFS2_OBY files and other parameters.
- Values: (string)

6.2.44 ROFS2_NAME

- Type: Variable
- Description: The full name of the ROFS2 image.
- Values: (string)

6.2.45 ROFS2_OBY

- Type: Variable
- Description: The OBY file(s) included to the ROFS2 image creation.
- Values: (string)

6.2.46 ROFS2_OPT

- Type: Variable
- Description: The ROFS2 specific buildrom options.
- Values: (string)

6.2.47 ROFS2_ROMVER

- Type: Variable
- Description: The ROFS2 ROM version string.
- Values: (string)

6.2.48 ROFS2_TIME

- Type: Variable
- Description: The time defined to the ROFS2 image.
- Values: (string)

6.2.49 ROFS2_VERIBY

- Type: Variable
- Description: The (generated) version IBY file name for the ROFS2 image. This file includes the version text files and other version parameters.
- Values: (string)

6.2.50 ROFS3_CUSTFILE

- Type: Variable
- Description: The (generated) source file name for *customersw.txt*.
- Values: (string)

6.2.51 ROFS3_CUSTINFO

- Type: Variable

- Description: The content string for *customersw.txt*.
- Values: (string)

6.2.52 ROFS3_DIR

- Type: Variable
- Description: The working directory, when creating the ROFS3 image.
- Values: (string)

6.2.53 ROFS3_FOOTER

- Type: Variable
- Description: This variable can contain a footer section for the ROFS3 master OBY.
- Values: (string)

6.2.54 ROFS3_FWIDFILE

- Type: Variable
- Description: The (generated) *_rofs3_fwid.txt* file name.
- Values: (string)

6.2.55 ROFS3_FWIDINFO

- Type: Variable
- Description: The content string for the *fwid3.txt* file.
- Values: (string)

6.2.56 ROFS3_HEADER

- Type: Variable
- Description: This variable can contain a header section for the ROFS3 master OBY.
- Values: (string)

6.2.57 ROFS3_MSTOBY

- Type: Variable
- Description: The (generated) version IBY file name for the ROFS3 image. This file includes the version text files and other version parameters.
- Values: (string)

6.2.58 ROFS3_NAME

- Type: Variable
- Description: The full name of the ROFS3 image.
- Values: (string)

6.2.59 ROFS3_OBY

- Type: Variable
- Description: The OBY file(s) included to the ROFS3 image creation.
- Values: (string)

6.2.60 ROFS3_OPT

- Type: Variable
- Description: The ROFS3 specific buildrom options.
- Values: (string)

6.2.61 ROFS3_ROMVER

- Type: Variable
- Description: The ROFS3 ROM version string.
- Values: (string)

6.2.62 ROFS3_TIME

- Type: Variable
- Description: The time defined to the ROFS3 image.
- Values: (string)

6.2.63 ROFS3_VERIBY

- Type: Variable
- Description: The (generated) version IBY file name for the ROFS3 image. This file includes the version text files and other version parameters.
- Values: (string)

6.2.64 SOS_VERSION

- Type: Variable
- Description: Symbian OS version number. The value is used in the version information generation (*platform.txt*). See USE_VERGEN.
- Values: ([0-9]+.[0-9]+)

6.2.65 USE_OVERRIDE

- Type: Variable
- Description: Define whether the *override.pm Buildrom.pl* plug-in is used (includes automatically also */epoc32/rom/override.oby*).
- Values: ([0|1])

6.2.66 USE_PAGING

- Type: Variable
- Description: Define the usage of On Demand Paging (ODP). (E.g. 0, rom, code).
- Values: ((0|rom|code[:[(1|2|3)]+]?))

6.2.67 USE_ROFS

- Type: Variable
- Description: Define the ROFS sections in use. A list separated with commas can be given as possible values. (E.g. 1,2,3).
- Values: ([[dummy|]0..6][,[dummy|]0..6]*)

6.2.68 USE_ROMFILE

- Type: Variable
- Description: Define whether */epoc32/rombuild/romfiles.txt* is used. Files in romfiles are automatically moved to ROM, everything else in the core is moved to ROFS1.
- Values: ([0|1])

6.2.69 USE_ROMSYMGEN

- Type: Variable
- Description: Generate the ROM symbol file. 0= Do not generate, 1= Generate.
- Values: ([0|1])

6.2.70 USE_UDEB

- Type: Variable
- Description: Include the usage of the debug binary **.txt* to define the list of binaries that are taken from the UDeb folder instead of URel.
- Values: ([0|1|full])

6.2.71 USE_VERGEN

- Type: Variable
- Description: Use the iMaker version information generation.
- Values: ([0|1])

6.2.72 WORKDIR

- Type: Variable
- Description: The working directory for the image creation.
- Values: (string)

7 iMaker command line usage

This chapter describes briefly how to use iMaker from the command line, as well as the syntax and ways to use the makefile variables and targets inside iMaker.

7.1 General concept

iMaker is called via the *imaker.cmd* file (in *epoc32/tools/imaker.cmd*), which eventually calls MinGW make with the default configuration file included (in */epoc32/tools/rom/imaker.mk*). Any parameters given to *imaker.cmd* are passed through to the actual make command, so actually the command line interface of iMaker is exactly the same as the interface of make (see Running Make). This document does not cover the entire command line functionality of Make, because it is already well documented in the GNU Make manual. Here are described some of the key concepts that are needed when running iMaker.

7.1.1 Defining the configuration (makefiles)

All configuration files in iMaker are makefiles, so when the image creation is done, the configuration is given as a makefile. The makefiles are given with an `-f` or (`--file`) option. E.g. `-f image_conf_product.mk` adds this makefile to the list of included makefiles which are included after the default configuration *imaker.mk* makefile. An example:

```
Give a configuration as a command line parameter. The default target only
prints the version information.
> imaker -f /epoc32/rom/config/<family>/image_conf_<product>.mk
```

7.1.2 Defining the goal(s) from command line

In iMaker the goal defines which steps are being executed in the call in normal makefile manner. An example:

```
Give the execution target as a command line parameter. The flash target should
create flashable images.
> imaker -f /epoc32/rom/config/<family>/image_conf_<product>.mk flash
```

For a list of available targets, see the iMaker API targets on page 20.

7.1.3 Overriding variables from command line

Any iMaker variable can be overridden in the command line. An example:

```
Override a parameter as a command line parameter. The WORKDIR changes the
location where the all output files are stored in the image creation process.
> imaker -f /epoc32/rom/config/<family>/image_conf_<product>.mk flash
WORKDIR=/flash_work/<product>/
```

For a list of available variables, see the iMaker API variables on page 23.

8 Use cases

8.1 Creating a normal image

8.1.1 Preconditions

iMaker is properly installed to the system. A working iMaker configuration exists in the environment.

8.1.2 Steps

Execute iMaker from the command line (command prompt).

- imaker -f <path to makefile> flash

An example:

```
> imaker -f /epoc32/rom/config/<family>/<product>image_conf_product.mk flash
```

8.1.3 Postconditions

All image files are created or copied under working directory. The WORKDIR path in default configuration is */epoc32/rombuild/.*

8.2 Creating MultiROFS image(s)

8.2.1 Preconditions

- iMaker is properly installed to the system.
- A working iMaker configuration exists in the environment.
- The environment uses MultiROFS as the ROM configuration.

8.2.2 Creating a separate Core image

The default core image contains ROM and ROFS1 sections.

8.2.2.1 Steps

Execute iMaker from command line (Command prompt)

- imaker -f <path to makefile> core

An example:

```
> imaker -f /epoc32/rom/config/<family>/<product>/image_conf_product.mk core
```

8.2.2.2 Postconditions

The core image files are created or copied under working directory under core folder. The WORKDIR path in default configuration is */epoc32/rombuild/.*

8.2.3 Creating a separate ROFS2 image

8.2.3.1 Steps

Execute iMaker from command line (Command prompt)

- imaker -f <path to makefile> rofs2

An example:

```
> imaker -f /epoc32/rom/config/<family>/<product>/image_conf_product.mk rofs2
```

8.2.3.2 Postconditions

The ROFS2 image files are created or copied under working directory under the *rofs2* folder.

8.2.4 Creating a separate ROFS3 image

8.2.4.1 Steps

Execute iMaker from command line (Command prompt)

- imaker -f <path to makefile> rofs3

An example:

```
> imaker -f /epoc32/rom/config/<family>/<product>/image_conf_product.mk rofs3
```

8.2.4.2 Postconditions

The ROFS3 image files are created or copied under working directory under *rofs3* folder.

8.3 Overriding variables from command line

This use case uses the *buildrom.pl* option overriding as an example.

8.3.1 Preconditions

iMaker is properly installed to the system. A working iMaker configuration exists in the environment.

8.3.2 Steps to override BLDROPT

1 Query the information about the iMaker variable from command line (optional)
   - imaker -f <path to makefile> help-variable-BLDROPT-all

2 Execute iMaker and override the variable BLDROPT
   - imaker -f <path to makefile> flash BLDROPT=-DTESTING_FLAG

An example that should add operator certificates to ROFS3 image:

```
> imaker -f /epoc32/rom/config/<family>/<product>/image_conf_product.mk rofs3
BLDROPT=-DS60_OPERATOR_CERTIFICATES
```

8.3.3 Postconditions

- The query returns all information about the variable (including the current value).
- All image files are created or copied under working directory.

8.4 Printing help from variables and targets

8.4.1 Precondition

iMaker is properly installed to the system.

8.4.2 Steps to query help on all targets

Query help on targets.

- imaker help-target or imaker help-target-*

8.4.3 Postconditions

The query returns a list of all documented targets.

8.4.4 Steps to query help on all variables

Query help on variables.

- imaker help-variable or imaker help-variable-*

8.4.5 Postconditions

The query returns a list of all documented variables.

8.4.6 Steps to query help on specific variables and their values

Query help on variables by searching.

- imaker help-variable-*ROFS3*-all
- imaker -f <path to configuration> help-variable-*ROFS3*-all. **Note:** The value can be configuration specific, so the -f configuration can show a different value.

An example that would print the ROFS3 related values for a product:

```
> imaker -f /epoc32/rom/config/<family>/<family>/<product>/
image_conf_product.mk help-variable-*ROFS3*-all
```

8.4.7 Postconditions

The query returns a list of all documented variables (with value) that contain the word ROFS3 in it.

8.5 Creating a custom configuration file

To create a custom configuration file in iMaker you need to follow the makefile syntax. The custom configuration file can do all the same things as the platform configuration files or even iMaker core configuration file (*imaker.mk*). There is no difference in user defined makefiles and core makefiles of iMaker. The main rule is that last one prevails, so the include order is the one that defines which settings are set last.

8.5.1 Preconditions

iMaker is properly installed to the system. A working iMaker configuration exists in the environment.

8.5.2 Steps

1 Create a new makefile (e.g. *my_makefile.mk*)

2 Edit *my_makefile.mk*

3 Include the base configuration in *my_makefile.mk*

- *iMaker.mk* is automatically included, so the base configuration refers to product configuration.
- Include can also be done via command line (but is not as convenient).
- */epoc32/rom/config* is by default in the make command's system include paths.

Example:

```
include foo/bar/image_conf_product.mk
```

1 Test that the include works (you can use this at anytime to test that the created makefile is valid).

```
imaker -f my_makefile.mk
iMaker - The Image Maker, version 08.02.01, 09-Jan-2008.
```

2 Define the settings, steps and targets for the makefile.

Example:

```
# Target specific variable setting for rofs3 variant creation
myimage: USE_VERGEN =1
myimage: ROFS3_OBY  = <myvariant.oby>myimage: rofs3
```

3 Use the makefile.

```
imaker -f my_makefile.mk myimage
```

8.5.3 Postconditions

- The defined settings work and are visible in the output
  - In this case a ROFS3 image is created and *myvariant.oby* is included as the content.

8.6 Creating a product specific iMaker configuration file

See the general concept of configuration file creating in Creating a custom iMaker configuration file on page 35

8.6.1 Preconditions

iMaker is properly installed to the system. A working iMaker configuration exists in the environment.

8.6.2 Steps

1 Create a new makefile (e.g. *image_conf_product.mk*, it is suggested to name all product configuration files with the image_conf prefix).

2 Edit *image_conf_product.mk*

3 Include the base configuration in *image_conf_product.mk* if such exists (it is generally a good idea to create some generic hardware settings to a common file).
  - *iMaker.mk* is automatically included, so the base configuration refers to a product configuration.
  - Include can also be done via command line (but is not as convenient).
  - */epoc32/rom/config* is by default in the make command's system include paths.

Example 1:

```
include foo/bar/image_conf_family.mk
```

1 Test that the include works (you can use this anytime to test that the created makefile is valid)

```
imaker -f my_makefile.mk
iMaker - The Image Maker, version 08.02.01, 09-Jan-2008.
```

2 Define the needed settings, steps and targets for the makefile.
  - Product requires at least:
    - Core oby CORE_OBY on page 24
    - Core buildrom options CORE_OPT on page 24
    - ROFS usage (usage of MultiROFS) USE_ROFS on page 30
    - Demand Paging usage (whether or not to use paging) USE_PAGING on page 30, see Configuring the ROM/ROFS files on page 38 for exact instructions.
    - Usage of version generation USE_VERGEN on page 31

With MultiROFS you need to define the same parameters to each used ROFS section.

- ROFS2 oby: ROFS2_OBY on page 28
- ROFS2 buildrom options: ROFS2_OPT on page 28
- ROFS3 oby: ROFS3_OBY on page 29

- ROFS3 buildrom options: ROFS3_OPT on page 30

Example 2:

```
USE_FLOAT    =0
USE_VERGEN   =1
USE_ROFS     =1,2,3
USE_PAGING   =rom
USE_PLATSEC  =1
USE_ROMFILE  =1

COREPLAT_OPT  =
CORE_OBY      = $(E32ROM)/master.oby
CORE_OPT      = $(BLDROM_OPT) -D_EABI=$(ARM_VERSION) -D$(call ucase,$
(PRODUCT_NAME))  -es60ibymacros
```

Use the makefile.

```
imaker -f image_conf_product.mk core
```

8.6.3 Postconditions

The defined settings work and are visible in the output. In this case, a core image is created.

8.7 Changing binary files to UDeb (creating debug images)

8.7.1 Preconditions

iMaker is properly installed to the system. A working iMaker configuration exists in the environment.

The variables that need to be set to a proper value are USE_UDEB on page 31 and CORE_UDEBFILE on page 25.

8.7.2 Steps

1 Get the current values of these values.

```
Example:
> imaker -f image_conf_<product>_ui.mk help-variable-*UDEB*-value
Testing new API
CORE_UDEBFILE = `/epoc32/rombuild/mytraces.txt'
USE_UDEB = `0'
```

- Normally, CORE_UDEBFILE is set to something.

2 Define the binaries to the CORE_UDEBFILE text file.

- Create the file if it does not exist
  - Add the target binary names of the files that need to be in UDeb. See iMaker OBY conversion file format on page 12.

```
Example:
# Debug files
IntegrityCheckClient.dll
sanimengine.dll
RestrictedAudioOutputProxy.dll
```

3 Enable the USE_UDEB either from command line or from a makefile.

- imaker -f <path to makefile> target USE_UDEB=1

```
Nokia specific example:
> imaker -f /epoc32/rom/config/<family>/<product> flash USE_UDEB=1
```

8.7.3 Postconditions

The defined core image files in *my_traces.txt* are taken from UDeb folder instead of URel. This can be checked from the *<config>_rom.oby/<config>_rofs.oby*.

8.8 Configuring the ROM/ROFS1 files

8.8.1 Preconditions

The iMaker is properly installed to the system. A working iMaker configuration exists in the environment.

There are several variables that affect the ROM/ROFS configuration are depending of what type of configuration is in use.

- USE_ROMFILE on page 31 defines whether to use romfiles concept at all. If set to 0 the romfiles are ignored.
- USE_PAGING on page 30 defines what type of memory paging configuration is in use.
  - USE_PAGING=0 means that no paging (Non-demand Paging) is in use. So the old memory mechanism is in use where an entire executable binary is loaded to RAM for execution.
  - USE_PAGING=rom means that ROM paging (ROM Demand Paging) is in use. Executable binaries from ROM are paged.
  - USE_PAGING=code means that all partitions can use paging (Code Demand Paging). Executable binaries from all sections can basically be paged.
- CORE_NDPROMFILE on page 24 is used when non demand paging configuration is in use. Defines the name of the root romfile that is used to define files for ROM.
- CORE_ODPROMFILE on page 24 is used when ROM demand paging configuration is in use, to define the root romfile.
- CORE_CDPROMFILE on page 23 is used when code demand paging configuration is in use, to define the root romfile.

Even though the variables seem complex, the main concept is very simple. The romfiles define content for the ROM, which rofsfiles can move back to ROFS1 (rofsfiles override romfiles). The reason for this type of behavior is to enable using of wild cards in romfiles and override them in rofsfiles.

8.8.2 Steps

1 Get the current values of these values.

```
Example:
> imaker -f image_conf_foo_ui.mk help-variable-*ROMFILE*-value
Testing new API
CORE_CDPROMFILE = `/epoc32/rombuild/odpcoderomfiles.txt'
CORE_NDPROMFILE = `/epoc32/rombuild/romfiles.txt'
CORE_ODPROMFILE = `/epoc32/rombuild/odpromfiles.txt'
USE_ROMFILE = `1'

> imaker -f image_conf_<product>_ui.mk help-variable-*ROFSFILE-value
Testing new API
CORE_ROFSFILE = `/epoc32/rombuild/odprofsfiles.txt'

> imaker -f image_conf_foo_ui.mk help-variable-*PAGING*-value
Testing new API
USE_PAGING = `rom'
```

This configuration uses ROM paging, the USE_ROMFILE=1 and the files to be configured are therefore */epoc32/rombuild/odpromfiles.txt* and */epoc32/rombuild/odprofsfiles.txt*.

2 Define the binaries to the text file.

- Create the file if it does not exist.
  - Add the target binary names of the files that need to be in ROM. See iMaker OBY conversion file format on page 12

```
Example:
# Rom files
*.ldd
*.dll
Aaa.exe
```

3 Define the exceptions to rofsfiles (if needed)

- Create the file if it does not exist
  - Add the target binary names of the files the same way as in romfiles. See iMaker OBY conversion file format on page 12

```
Example:
# Rofs files
bb.dll
```

8.8.3 Postconditions

The defined romfiles are in ROM and the exceptions in ROFS1. This can be checked from *<config>_rom.oby/<config>_rofs.oby*.

8.9 Creating an operator variant image (ROFS3)

It is possible to create ROFS3 or operator variant image by defining parameters on command line or by defining the configuration to makefile (e.g. *my_makefile.mk*).

8.9.1 Preconditions

iMaker is properly installed to the system. A working iMaker configuration exists in the environment.

8.9.2 Steps

1 Define OBY (content) to include in ROFS3_OBY on page 29.

2 Define buildrom options to ROFS3_OPT on page 30.

Example on *makefile operatorvariant.mk*:

```
# Expected that these oby files are in system include paths (NOTE: In this
example the ROFS3_OPT adds system includes in the buildrom options)
ROFS3_OBY = <s60Cenrep_rofs3.oby> <my_variant.oby>
ROFS3_OPT = -I<PATH_TO_MY_VARIANT>
```

3 Use the makefile.

```
imaker -f operatorvariant.mk rofs3
```

8.9.3 Postconditions

The defined settings work and are visible in the output. In this case, a ROFS3 image is created and selected content is included to the image.

8.10 MultiROFS eclipsing

The Symbian composite file system offers possibility to eclipse/hide files. This means that a ROFS image can define a file not to be shown even though it is present on an earlier ROFS or ROM image.

8.10.1 Rombuild configuration

The Symbian *buildrom.pl* accepts a hide parameter as an IBY file definition. There the hiding parameter requires the target file as a given input. It is not recommended to hide files in Core, it is also required to define the image section where the hiding is done.

8.10.1.1 Syntax highlight

```
hide=targetfile
```

8.10.1.2 Example syntax use

Here */sys/bin/test.exe* is hidden in the ROFS3 image section.

```
ROM_IMAGE[3] {
  hide=/sys/bin/test.exe
}
```

8.10.2 Example: Hiding PoC

The PoC application is included to the normal terminal image, which means that the EXE is on Core and the resources are on Language variant part (ROFS2 in MultiROFS case). The PoC application can be hidden from operator variant by using the image creation time hide= parameter. So by defining the following to an IBY file, you end up with a structure illustrated in the image. As a consequence, the file system returns `KErrPathHidden` when trying to open any of these files.

```
ROM_IMAGE[3] {
hide=/resource/apps/PoC.r01
hide=/resource/apps/PoC.r02
hide=/sys/bin/Poc.exe
hide=/resource/apps/PoC_aif.mif
hide=/Private/10003a3f/apps/PoC_reg.rsc
}
```

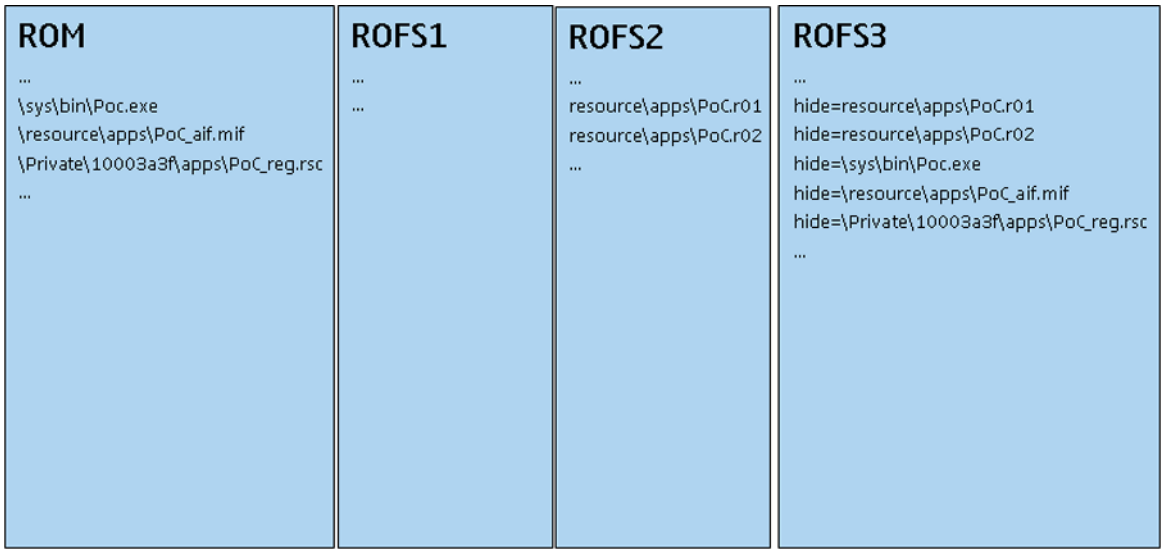

Figure 2: MultiROFS eclipsing

Note: When hiding applications like PoC, it must be checked that the userdisk is erased or blank, so that the application grid links are created correctly on the first boot-up. Otherwise the application grid start-up links can be completely corrupted.

9 MultiROFS configuration guideline

This chapter provides a guideline for MultiROFS configuration usage.

9.1 Technical background

For further information, see:

- Symbian OS Library for Device Creators.chm : Enabling faster ROM reflashing for description of the MultiROFS concept.
- MultiROFS eclipsing on page 39
- MultiROFS overriding on page 43

9.2 Use cases

9.2.1 MultiROFS configuration with ROM demand paging

The ROM demand paging can page the executable binaries on the ROM file system, which gives a quite big performance and memory usage improvement. So therefore it is recommended that the all possible CORE binaries are located on ROM. However, this means that these executable binaries should not be overridden with the MultiROFS system (see MultiROFS overriding on page 43 for reasoning). The eclipsing of files on ROM can still be done.

9.2.1.1 File system sections

- ROM: all core binaries
- ROFS1: all core data files
- ROFS2: all language and product default settings (vanilla variant)
- ROFS3: empty (except for the version information file)

9.2.1.2 Rules of thumb

1 All possible CORE binaries on ROM (Performance & Memory).

2 One should not override binaries on ROM.

3 Hiding of binaries/files on ROM is possible (operator variant application hiding).

9.2.2 MultiROFS configuration with CODE Demand Paging

CODE Demand Paging can page the executable binaries on any file system (ROM/ROFS/FAT (userdisk)), which provides a relatively high performance and memory usage improvement. The main difference to ROM Demand Paging is that the binaries can be placed on any section and can still be paged. Therefore, it is recommended to place every possible binary on CORE to ROFS1 so that there is the possibility of overriding it with MultiROFS.

9.2.2.1 File system sections

- ROM: Only the minimal kernel binaries
- ROFS1: all the rest of the core binaries and data files
- ROFS2: all language and product default settings (vanilla variant)
- ROFS3: empty (except for the version information file)

9.2.2.2 Rules of thumb

1 All possible CORE binaries are on ROFS1 (Performance & Memory).

2 It is not possible to override binaries on ROM.
3 All files on ROFS1 can be overridden and hidden.

10 Terms and abbreviations

Term or abbreviation	Definition
Flash image	Flash image refers to a file that is flashed to the terminal device via some data interface (USB, Serial interface). In Nokia, this means *.fpsx* files that can contain several SymbianImage files (**.img*).
MultiROFS	MultiROFS stands for Multiple Read Only Filesystem(s). This is a feature of Symbian's Composite file system that can mount several file systems as a single file system. So the devices *Z:* drive which is shown as single file system can actually contain several ROM/ROFS sections.
MultiROFS overriding	The MultiROFS system allows also overriding of any file in the previous ROMimages. This is due to the fact that the Symbian composite file system is read from biggest ROFS downwards. Example of reading order ROFS3, ROFS2, ROFS1 and lastly the ROM. So any file in ROM can be overridden just by placing a same named target in some ROFS section. 1 Any file can be overridden or hidden with MultiROFS composite file system. 2 Static dependencies are loaded directly inside ROM. • The XIP file system which the ROM is actually has links directly on a function level. E.g. *A.exe* links to function at address 0x12345678, instead of linking to *B.dll* at 0x10000000. This means that dependencies inside ROM are such that they never hit the composite file system (MultiROFS) or file level at all. • Basically, override can be done on an EXE and its dependencies with MultiROFS, because at the starting of the EXE the file is searched from MultiROFS (the launching of an EXE searches the file from composite file system). However, it is not possible to update e.g. *euser.dll* because all binaries are linked to the euser functions in that function level. • The overriding of EXE and its dependencies can be a bit risky, because it may lead to a situation where same DLL functions are run from ROFS / ROM. This can occur when the executable *a.dll* from ROFS and some other *b.dll* (in ROM) use the same *a.dll*, but it will execute its functions from the ROM version of *a.dll* (because of the XIP file system).