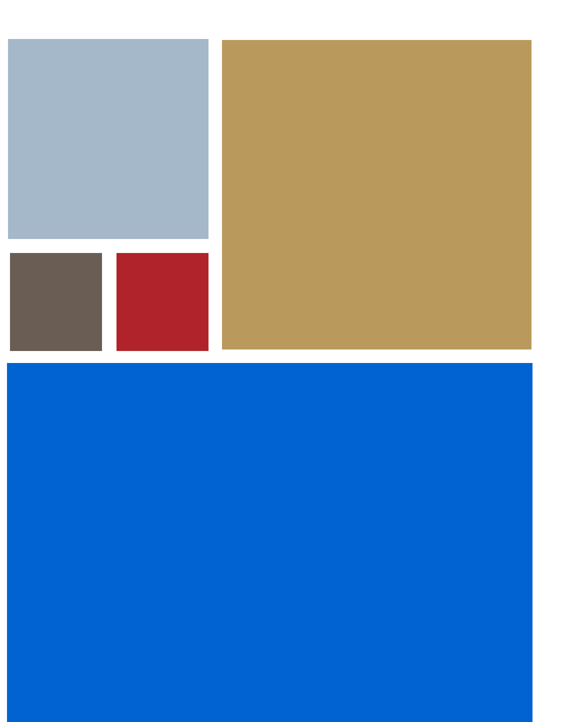


Using TrueFFS for OS-9®

Version 1.3

www.radisys.com

Revision B • July 2006

Copyright and publication information

This manual reflects version 1.3 of TrueFFS for Microware OS-9. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

July 2006
Copyright ©2006 by RadiSys Corporation
All rights reserved.

EPC and RadiSys are registered trademarks of RadiSys Corporation. ASM, Brahma, DAI, DAQ, MultiPro, SAIB, Spirit, and ValuePro are trademarks of RadiSys Corporation.

DAVID, MAUI, OS-9, OS-9000, and SoftStax are registered trademarks of RadiSys Corporation. FasTrak, Hawk, and UpLink are trademarks of RadiSys Corporation.

† All other trademarks, registered trademarks, service marks, and trade names are the property of their respective owners.

Contents

Chapter 1: Installing TrueFFS for OS-9®

Installing FTL on OS-9 6
Method 1: Creating a Bootfile with os9make 6
Method 2: Creating a Bootfile with the Configuration Wizard 7
Installing FTL on OS-9 for 68K 7
Adding FTL to a Running System 8

Chapter 2: Using and Testing TrueFFS

Formatting the Flash Device 10
Testing TrueFFS and the FTL 12

Chapter 3: Technical Overview

The OS-9 Flash File System 14
FTL Device Driver and Descriptors 14
FTL Device Driver Limitations 16
FTL Device Descriptors 16
FTL and the PC Card Program 17

Chapter 4: Programming Reference

System Calls 20
I_SETSTAT, SS_DEFLSH 21
I_SETSTAT, SS_DEFRAG 22
I_SETSTAT, SS_DELBLK 23
I_SETSTAT, SS_LLFRMT 24
Utilities 26
filltest 27
ftcheck 28
ftdefrag 29
ftformat 30
Configuration Fields 31
ds_flash_base 32
ds_flash_size 33
ds_flash_source 34
Errors 35

Chapter 5: Porting Guide

Common Code Files 38
Hardware-Specific Functions 38
Communication Structure 38
Required Fields 39
Optional Fields 39
Fields Specified in the Device Descriptor 39
Example Source Code 40

Driver and Descriptor 40

1 Installing TrueFFS for OS-9®

This chapter contains installation instructions for including the TrueFFS for OS-9® Flash Translation Layer (FTL) in an OS-9 boot image configuration or adding FTL to an already running OS-9 system. FTL is a standard for flash file systems which provides full disk emulation for standard flash devices. TrueFFS for OS-9 allows programs to have disk-like access to flash memory.

The following sections are included in this chapter:

- Installing FTL on OS-9
- Installing FTL on OS-9 for 68K
- Adding FTL to a Running System

Installing FTL on OS-9

Use one of the following methods to configure the OS-9 boot files to include FTL. If you do not want to reboot your OS-9 system to add FTL, refer to the Adding FTL to a Running System section.

Method 1: Creating a Bootfile with os9make

To create a bootfile using `os9make`, complete the steps below:

Step 1. Edit `MWOS/OS9000/<CPU Family>/PORTS/<Port>/BOOTS/SYSTEMS/PORTBOOT/bootfile.ml` to ensure that the following lines are uncommented. (Remove the leading asterisk.)

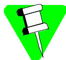
```
../../../../../CMDS/BOOTOBJS/rbf
../../../../../CMDS/format
```

Step 2. Add the following lines to the `bootfile.ml file`:

```
../../../CMDS/BOOTOBJS/rbftl
../../../CMDS/BOOTOBJS/DESC/RBFTL/rrf0
../../../CMDS/ftformat
```

If space in your image permits, uncomment the following lines in the `bootfile.ml` file. These utilities are optional, but help to test the installation.

```
../../../../../CMDS/dcheck
../../../../../CMDS/dir
../../../../../CMDS/kermit
../../../../../CMDS/mdir
../../../../../CMDS/save
../../../../../CMDS/filltest
../../../../../CMDS/ftcheck
../../../../../CMDS/ftdefrag
```



The descriptor name may differ on some OS-9 systems.

Step 3. Rebuild the bootfile using `os9make` and the makefile that resides in `MWOS/OS9000/<CPU Family>/PORTS/<port>/BOOTS/SYSTEMS/PORTBOOT`.

Step 4. Follow the instructions in your OS-9 for <target> Board Guide for placing the bootfile onto the target system.

Method 2: Creating a Bootfile with the Configuration Wizard

Complete the following steps to create a bootfile using the Configuration Wizard:

Step 1. Edit the `bootfile.ml` file in `MWOS/OS9000/<CPU Family>/PORTS/<Port>/BOOTS/INSTALL/PORTBOOT` to ensure that the following lines are uncommented. (Removing the leading asterisk.)

```
../../../../../CMDS/BOOTOBJS/rbf
../../../../../CMDS/format
```

If space in your image permits, uncomment the following utilities. These utilities are optional, but help to test the installation.

```
../../../../../CMDS/dir
../../../../../CMDS/mdir
../../../../../CMDS/save
```

Step 2. Add the following lines to the `user.ml` file in `MWOS/OS9000/<CPU Family>/PORTS/<Port>/BOOTS/INSTALL/PORTBOOT`:

```
../../../CMDS/BOOTOBJS/rbftl
../../../CMDS/BOOTOBJS/DESC/RBFTL/rrf0
../../../../../CMDS/ftformat
```

If space in your image permits, the following optional utilities can be added to test the installation:

```
../../../../../CMDS/dcheck
../../../../../CMDS/kermit
../../../../../CMDS/filltest
../../../../../CMDS/ftcheck
../../../../../CMDS/ftdefrag
```

Step 3. From the Configuration Wizard's Master Builder window, check the User Commands box, then rebuild the bootfile.

Step 4. Follow the instructions in your OS-9 for <target> Board Guide for placing the bootfile onto the target system.

Installing FTL on OS-9 for 68K

Use these instructions to configure the OS-9 for 68K boot files to include FTL. If you do not want to reboot your OS-9 for 68K system to add FTL, see Adding FTL to a Running System on page 8.

Step 1. Edit the applicable bootlist file (example: `d0.bl`, `h0.bl`) in `MWOS/OS9/<CPU Family>/PORTS/<Port>/BOOTLISTS` to ensure that the following lines are uncommented by removing the leading asterisk:

```
../../../68000/CMDS/BOOTOBJS/rbf
../../../68000/CMDS/format
```

If space permits, uncomment the following lines in the bootlist file by removing the leading asterisk. These utilities are optional, but help to test the installation.

```
../../../68000/CMDS/dcheck
../../../68000/CMDS/dir
../../../68000/CMDS/kermit
../../../68000/CMDS/mdir
../../../68000/CMDS/save
```

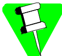
Add the following lines to the bootlist `file`:

```
CMDS/BOOTOBJS/RBFTL/rbftl
CMDS/BOOTOBJS/RBFTL/rrf0
CMDS/BOOTOBJS/RBFTL/rrf0fmt
```

The `rrf0fmt` is a format-enabled descriptor. The descriptor name may differ on some OS-9 for 68K systems.

Step 2. Rebuild the boot image.

Step 3. Follow the boot image download instructions included in your Microware manuals to place your new boot into flash on the system.



Refer to your *OS-9 for 68K Processors BLS Reference* for OS-9 for 68K installation and configuration information.

Adding FTL to a Running System

Complete the following steps to add FTL to a currently running OS-9 system. These steps work for both OS-9 and OS-9 for 68k systems.

Step 1. Use a program such as `kermit -ri` to transfer the following modules from your host system to the target:

- `rbftl`
- `rrf1`
- `rrf1fmt` (for OS-9 for 68K)

Step 2. Load the driver, `rbftl`. To load `rbftl` from your execution directory, use the following command at an OS-9 shell prompt:

```
load -d rbftl
```

Step 3. Load the descriptors. To load `rrf1` from your execution directory, use the following command from an OS-9 shell prompt:

```
load -d rrf1
```

The descriptor name may differ on some OS-9 systems.

2 Using and Testing TrueFFS

This chapter explains how to use and test TrueFFS and the FTL for OS-9/OS-9 for 68K once it has been installed.

OS-9 for 68K Users:
When using TrueFFS for the first time, the bootfile must be in the EPROM device; some flash chips require the entire flash area to be seen by the driver. You also need another media (hard disk, floppy disk) for the high-level system. After the flash disk is initialized, you can copy the high-level system from the hard disk or floppy disk to the flash disk, and make a script to run the modules in the flash disk.

OS-9 for 68K Users:
Do not want to put a bootfile on an EPROM that is bigger than the EPROM size allows.

The following sections are included in this chapter:

- Formatting the Flash Device
- Testing TrueFFS and the FTL

Formatting the Flash Device

Before you can perform any file operations on a flash device, you must perform a low-level format and a high-level format. Be sure the target does not have jumpers or switches set to boot from FLASH before trying to format FLASH. For more information on FLASH, refer to your OS-9 board guide.

On some systems (for example the StrongARM ThinClient and GraphicsClient) it is necessary to run the `pflash -u -ew` command prior to performing the steps below. This is necessary if `pflash` has been used previously to program the Flash device.

Step 1. Use the included `ftformat` utility to perform the low-level FTL format of the flash device. The command below uses the default parameters:

```
[1]$ ftformat /rrf0

This utility will perform a low level format of a flash volume /rrf0@
for use with the rbftl TrueFFS flash file system driver.

This operation can run for up to 35 seconds for each megabyte of
flash being formatted.

This formatting will destroy all file and low-level state info
including wear-leveling info, if any, stored in the flash.

Do you wish to continue (y or n)? : y

The low level format of /rrf0@ has completed successfully.

Be sure to perform a format of /rrf0 before attempting file
operations (answer n to requests for physical format and physical
verify in format).
```

The `ftformat` utility uses the `I_SETSTAT`, `SS_LLFRMT` call to perform a low-level format of the flash volume.

The parameters passed with this call include:

- boot image length
- percent of the device to reserve for when the volume becomes full
- number of spare erase units
- size of RAM-resident FTL virtual map
- additional FAT-16 parameters

The low-level format needs to be done only once unless the format becomes corrupted.

The only operations you can perform on a flash volume before low-level formatting are: `I_ATTACH`, `I_OPEN`, `I_SETSTAT`, and `I_CLOSE`.

Step 2. Use any OS-9 format utility to perform the high-level, file system formatting of the volume. This example uses the `format` utility with default parameters:

```
[2]$ format /rrf0
```

```
disk format utility
    OS-9000 for the ARM   PS7111 - 7111
   ---------------- format parameters ---------------
disk type: hard disk - auto sizing
variable parameters:
  block size: 512
  logical block offset: 0
  block interleave: 3
  block address of bitmap: 1242
  total blocks on disk: 3726

ready to format /rrf0 (y/n/q) ? y

do physical format (y/n/q) ? n

disk name (32 bytes max.): <flash>

performing the logical format

do physical verify (y/n/q) ? n

quantity good           3726    (    1907712 bytes )
quantity bad               0    (          0 bytes )
quantity on disk        3726    (    1907712 bytes )
quantity verified          0    (          0 bytes )
```

The high-level format utility typically obtains the number of total blocks to format using an `I_GETSTAT, SS_DSIZE` call to `rbftl`. Since this call is supported by the driver, a high-level format operation can use the automatic parameters selected by the `format` utility.

If the low-level format was not performed, the `I_GETSTAT, SS_DIZE` command from the high-level format utility fails.

You can use the formatted flash device with standard RBF or PCF commands and utilities. The following example shows the `dir` utility using an RBF descriptor.

```
[3]$ dir -e /rrf0

Directory of /rrf0 00:21:05
Owner  Last modified Attributes Block  Bytecount Name
-----  ------------- ---------- -----  --------- ----
```

To mark blocks of flash as deleted, use the `I_SETSTAT, SS_DELBLK` call. This allows the flash memory to be re-used after an RBF or PCF file is deleted. The `SS_DELBLK` call is not performed by RBF or PCF, but the capability may be added.

To defragment and recycle non-writeable areas of flash, use the `ftdefrag` utility. The following example shows a typical use of the `ftdefrag` command:

```
[4]$ ftdefrag /rrf0@

Specify (in dec) max number of free sectors desired: 400
```

```
Actual number of free sectors available (in dec) is 438
```

The `ftdefrag` utility uses the `I_SETSTAT`, `SS_DEFRAG` call. This call is typically performed by an application after an `I_SETSTAT`, `SS_DELBLK` call.

Testing TrueFFS and the FTL

Step 1. On the target board, change to the directory where the descriptor is located using the following command: `chd /rrf0`

Step 2. Save all modules from memory into the flash to test the write function. Use the following command from the target board shell prompt:

```
mdir -u ! save -z &
```

The system should display a shell prompt while performing the write operations.

Step 3. Enter commands to test the system while a save operation occurs. Below is an example:

```
save rbf
dir -e
```

Step 4. Enter commands to delete files while saving. Below is an example:

```
del activ
del build
del dir
```

3 Technical Overview

This chapter explains how the `rbftl` device driver works with the OS-9 RBF and PCF file managers to make flash memory accessible like disk devices.

The following sections are included in this chapter:

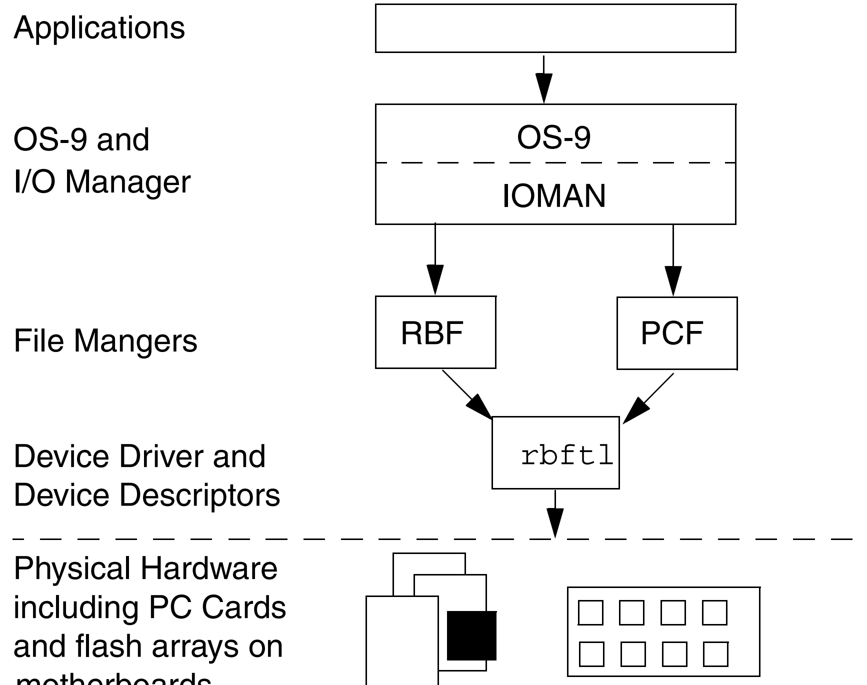
- The OS-9 Flash File System
- FTL Device Driver and Descriptors
- FTL and the PC Card Program

The OS-9 Flash File System

FTL (flash translation layer) is a standard for flash file systems which provides full disk emulation for standard flash devices. TrueFFS for OS-9 uses existing OS-9 file managers with a new device driver, `rbftl`, to enable access to flash memory that emulates disk access. Flash memory is non-volatile, block-accessible memory. FTL supports board-resident flash and PC Card flash devices. See your Release Notes to determine the cards and flash devices that are supported for your board.

Like other OS-9 I/O systems, FTL separates the applications and operating system from the physical devices. This enables the `rbftl` device driver to handle multiple types of devices through the common RBF and PCF file manager interfaces as shown in Figure 3-1..

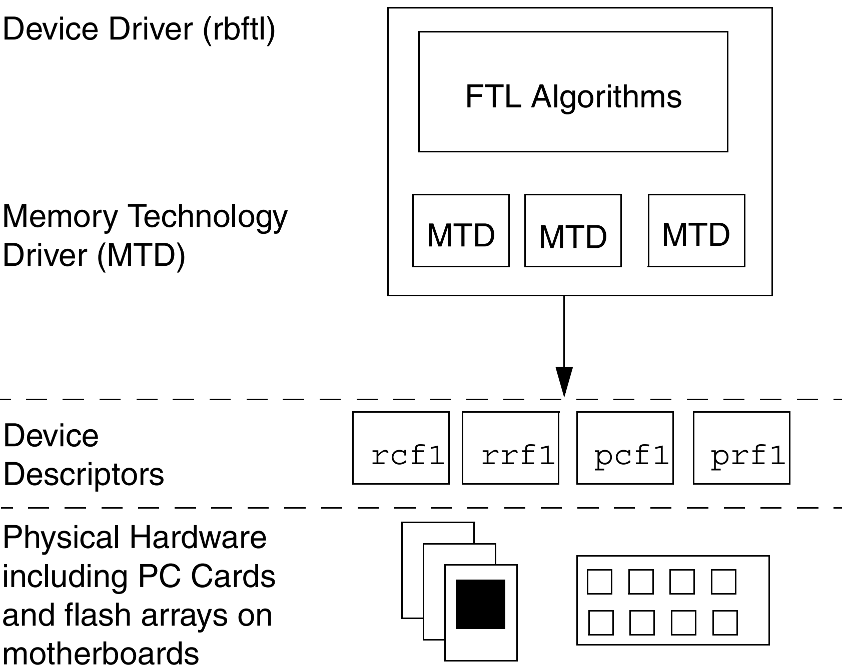
Figure 3-1. OS-9 I/O System with FTL



FTL Device Driver and Descriptors

FTL for OS-9 includes the `rbftl` device driver and RBF and PCF device descriptors. `rbftl` includes translation algorithms common to all devices and device-specific Memory Technology Device (MTD) components which are board-specific.

Figure 3-2. FTL rbftl Structure



The `rbftl` driver includes a single "block manager" and set of hardware driver routines which share a relationship similar to that of RBF and a typical device driver.

The "block manager," shown as the FTL Algorithms in Figure 3-2., implements the media wear-leveling intelligence.

Media wear-leveling is accomplished through a metadata structure on the media which provides `rbftl` with the following:

- a block address translation map
- block wear-leveling information
- block-in-use information

The FTL algorithms use this information for block read requests and to determine a flash block range for memory write requests.

To maintain flash memory device independence, the MTD (memory technology driver) intelligence is not embedded into the FTL algorithms. Separate routines, though still internal to the driver, are used by the FTL algorithms to manage flash hardware.

The thin layer that binds the FTL algorithms to a given MTD (memory technology driver) is a standard set of call entry points. These are:

- Write - writes a block of bytes to flash.
- Erase - erases one or more contiguous flash erasable blocks.

- Identify - probes the flash hardware to determine if it is a type this MTD supports. If so, the MTDs `write` and `erase` routines are set up for the FTL algorithms to call.

A read routine is not included because all types of flash that can be memory mapped may be read with simple processor memory reads.

FTL Device Driver Limitations

- `rbftl` cannot operate with flash cards that are write protected. To read the flash JEDEC ID, the FTL code puts the flash into command mode which can only be done when the flash is write-enabled.
- `rbftl` supports unpartitioned volumes only. This prevents `rbftl` from reading flash cards that were partitioned on another system.
- FTL for OS-9 supports only one device per LUN (logical unit number) to be open under a single instantiation of the driver. Multiple flash volumes require multiple driver static storage allocation. This is guaranteed by having different `dd_port` values in each `rbftl` descriptor.

FTL Device Descriptors

FTL for OS-9 includes the following device descriptors:

- `rcf1` for PC Card flash under RBF
- `rcf2` for PC Card flash under RBF
- `rrf1` for resident flash under RBF
- `pcf1` for PC Card flash under PCF
- `pcf2` for PC Card flash under PCF
- `prf1` for resident flash under PCF

These descriptors are standard RBF or PCF descriptors which include the following:

- additional device-specific logical-unit static storage initializers to define a flash base address
- a flash or array size
- a flag to indicate if the flash is PC Card based or resident
- a flash type JEDEC identification specifier

The standard descriptor header, `dd_port` field, is set to the flash base address value. The LUN initializer is always zero (0).

The PC Card flash descriptors are not PCMCIA socket specific. `rcf1, rcf2, pcf1`, and `pcf2` work with a flash card inserted into any supported socket.

`rbftl` requires that all of the flash accessed as a single volume is directly byte accessible. The base address and the size of the flash to be accessed are specified in the device descriptors. The base address must point to the beginning of a single flash device so `rbftl` can access the flash command registers. For interleave flash, the base address must point to the first flash device.

The flash size does not have to span the entire flash device but must terminate on an erase block boundary.

The flash location flag is not used. `rbftl` accesses both resident and PC Card-based flash without regard for the additional considerations that removable flash provides. For PC Card flash, this requires that another program has already ensured the following:

- A single memory window large enough to access the entire flash array is enabled.
- Flash accesses through this window are always word-wide.
- Flash `Vcc` and `Vpp` are always available and enabled.
- Flash is always available. The PC Card is in the slot.

The flash type identification specifier, JEDEC ID, is not used. FTL auto-detects the flash type. If it is necessary to manually set the JEDEC ID, this number can be set to a non-zero value.

FTL and the PC Card Program

A separate PC Card program is required to use FTL with flash memory on PC Cards. FTL requires the PC Card program to:

- initialize the flash socket with `I_ATTACH`
- open a single memory window into the flash array on the card with the base address specified in the `rbftl` device descriptor
- issue an `I_DETACH` call when the card is removed

The flash length value specified in the `rbftl` descriptor is an upper bound. If the PC Card program determines that the flash array is larger than the descriptor value, `rbftl` uses the descriptor value when operating on the flash volume. If the PC Card program determines the flash array is smaller than the descriptor value, the PC Card program needs to issue an `SS_DEFLSH` call to the flash device to modify the value used by the `rbftl` descriptor. For `rbftl` to obtain the modified values in the driver static storage, the flash device must be attached at the time of the change. For `rbftl` to retain the modified values, the flash device must remain attached. The values are cleared from the driver static storage when a new `SS_DEFLSH` call is issued or when the card is removed using `I_DETACH` calls.

Using the CardSoft daemon, `csfd`, as the PC Card program allows `rbftl` to access the PCMCIA socket controller. `csfd` passes the path identification for the opened socket to `rbftl` through an `I_SETSTAT` call. `rbftl` can perform `csf` operations in system state in a device-independent manner. `csfd` can pass the device name of the socket containing the flash card to `rbftl`. This enables a process performing I/O operations on flash files to obtain the socket device name through a special `I_GETSTAT` call. The process can register as a notification client through the CardSoft application programming interface (API) like `csfd` registers itself. Any process for which flash card removal is a critical event can receive notification of the event.

4 Programming Reference

This chapter contains the reference pages for the FTL system calls, utilities, configuration fields, and errors.

The following sections are included in this chapter:

- System Calls
- Utilities
- Configuration Fields
- Errors

System Calls

The following OS-9 system calls are included for TrueFFS and the FTL:

- `I_SETSTAT, SS_DEFLSH`
- `I_SETSTAT, SS_DEFRAG`
- `I_SETSTAT, SS_DELBLK`
- `I_SETSTAT, SS_LLFRMT`

I_SETSTAT, SS_DEFLSH

Function Code

`I_SETSTAT, SS_DEFLSH`

Syntax

```
#include <IO/ftlsrvcb.h>
```

Parameter Block Structure

```
typedef struct ss_deflsh_pb
{
    u_int32 flash_base;
    u_int32 flash_size;
    u_int16 bus_width;
}
ss_deflsh_pb, *Ss_deflsh_pb;
```

Attributes

State: User, System, and I\O

Description

`csfd` calls `I_SETSTAT, SS_DEFLSH` to notify `rbftl` that the flash array in the system is smaller than specified in the descriptor.

`flash_base`

is the starting memory address of the flash array.

`flash_size`

is the actual size of the flash array.

`bus_width`

specifies in bits the width of array memory accesses.

I_SETSTAT, SS_DEFRAG

Function Code

`I_SETSTAT, SS_DEFRAG`

Syntax

```
#include <IO/ftlsrvcb.h>
```

Parameter Block Structure

```
typedef struct ss_defrag_pb
{
    u_int32 irLength;
}
ss_defrag_pb, *Ss_defrag_pb;
```

Attributes

State: User, System, and I\O

Description

The `ftdefrag` utility calls `I_SETSTAT, SS_DEFRAG` to defragment a flash device.

`irLength` is the maximum number of free sectors to make available. Depending on the amount of recoverable space on the device, this number may not be reached.

Possible Errors

EOS_BAD_FORMAT
EOS_BTYP
EOS_GENERAL_FAILURE
EOS_WRITE_FAULT

I_SETSTAT, SS_DELBLK

Function Code

`I_SETSTAT, SS_DELBLK`

Syntax

```
#include <IO/ftlsrvcb.h>
```

Parameter Block Structure

```
typedef struct ss_delblk_pb
{
    u_int32 irSectorNo;
    u_int32 irSectorCount;
}
ss_delblk_pb, *Ss_delblk_pb;
```

Attributes

State: User, System, and I\O

Description

`I_SETSTAT, SS_DELBLK` marks blocks of flash as deleted. This allows flash memory to be re-used more efficiently after files have been deleted.

`irSectorNo` is the starting sector number of blocks to delete.

`irSectorCount` is number of sectors to delete.

Possible Errors

EOS_BAD_FORMAT
EOS_BTYP
EOS_GENERAL_FAILURE
EOS_SECTOR_NOT_FOUND
EOS_WRITE_FAULT

I_SETSTAT, SS_LLFRMT

Function Code

`I_SETSTAT, SS_LLFRMT`

Syntax

```
#include <IO/ftlsrvcb.h>
```

Parameter Block Structure

```
typedef struct ss_llfrmt_pb
{
    FormatParams* formatParams;
}
ss_llfrmt_pb, *Ss_llfrmt_pb;
```

Attributes

State: User, System, and I\O

Description

The `ftformat` utility calls `I_SETSTAT, SS_LLFRMT` to perform a low-level format of the flash volume.

`formatParams` are a set of device parameters including:

- `int32 bootImageLen;`
  Space to reserve for a boot-image at the start of the medium. The FLite volume begins at the next higher erase unit boundary.
- `u_int16  percentUse;`
  FTL performance depends on how full the flash media is. Performance slows as the media is close to 100% full. To increase performace, format the media to less than 100% capacity, which guarantees some free space at all times by sacrificing some capacity.
- `u_int16 noOfSpareUnits;`
  FTL needs at least one spare erase unit to function as a read/write media. (It is possible to specify zero (0) to achieve WORM functionality). You can specify more than one spare unit. This takes more media space, but if one of the flash erase units becomes bad and un-erasable in the future, one of the spare units needs to replace it. A second spare unit can continue `read/write` functionality. Without the spare unit, the media enters read-only mode. The standard value to use is 1.

- `u_int32 vmAddressingLimit;`
  Part of the FTL virtual map always resides in RAM. The RAM-resident portion is used to address the media below the VM addressing limit. Reading and writing to this part is usually faster. The larger the limit, the more RAM size that is required. To get the extra RAM requirement in bytes, divide the limit by 512. The minimum VM limit is 0. The standard value to use is 0x10000, the first 64KB.

- ```
  int (*progressCallback)
       ( u_int16 totalUnitsToFormat,
           u_int16 totalUnitsFormattedSoFar );
  ```
  The progress callback routine is called if not NULL. The callback routine is called after erasing each unit, and its parameters are the total number of erase units to format and the number erased so far. The callback routine returns a status value. A value of OK (0) allows formatting to continue. Any other value will abort the formatting with the returned status code.

DOS Formatting Section

- `char volumeId[4];`

  The volume identification number is four characters.

- `char *volumeLabel;`

  The volume label string is used if entered. If NULL, no label is used.

- `u_int16  noOfFATcopies;`

  DOS media is usually formatted with two FAT copies. The first copy is always used, but more copies make it possible to recover if the FAT becomes corrupted, which is a rare occurrence. Having multiple copies slows down performance and uses media space. The standard value to use is 2.

- `u_int16 embeddedCISlength;`

  This is the length in bytes of CIS to embed after the unit header.

- `u_int8 *embeddedCIS;`

  The unit header is structured as a beginning of a PCMCIA 'tuple' chain (a CIS). The unit header contains a data organization tuple, which points past the end of the unit header to a location which usually contains hexadecimal ff's which mark an 'end-of-tuple-chain'. It is possible to embed an entire CIS chain at this location. If so, '`embeddedCISlength`' marks the length of the chain in bytes.

Possible Errors

`EOS_BAD_FORMAT`. `SS_LLFRMT` returns this code when the verify pass following the low-level format failed, indicating a hardware failure.

`EOS_<callback>`. These are callback procedure errors. One field in the `SS_LLFRMT` structure is for a callback procedure to take advantage of format progress status.

`EOS_VOLUME_TOO_SMALL`

`EOS_WRITE_FAULT`
  ```

Utilities

The following FTL utilities are included:

- `filltest`
- ftcheck
- `ftdefrag`
- `ftformat`

filltest

Syntax

```
filltest [<opts>] {<path> [<opts>]}
```

Attributes

Operating System:OS-9 and OS-9 for 68K

Options and Parameters

`<path>`

pathname of the flash device to do the test

`-h[=]<count>`

Performs the “Hammer” test count times after writing the files. The default is 5.

`-s[=]<size>`

Specifies the maximum number of files to write. The default is 9999.

Description

The `filltest` utility completely fills any RBF-formatted device with files that are several blocks in length and performs the stress testing.

Examples

```
filltest -s=100 /rrf1
```

Errors

EOS_BAD_FORMAT
EOS_BTYP
EOS_GENERAL_FAILURE
EOS_UNKNOWN_MEDIA

EOS_WRITE_FAULT

ftcheck

Syntax

```
ftcheck <mem addr> [size]
```

Attributes

Operating System:OS-9 and OS-9 for 68K

Parameters

`<mem addr>`

The starting address of the flash disk

`[size]`

The size of the flash disk

Description

The `ftcheck` utility checks the integrity of an FTL media by collecting the information, such as Erase Units, Sectors/Unit, Direct Memory from it. It can also verify erased media.

Examples

```
ftcheck 0xFFA00000
```

Errors

EOS_BTYP
EOS_GENERAL_FAILURE
EOS_UNKNOWN_MEDIA
EOS_WRITE_FAULT

ftdefrag

Syntax

```
ftdefrag [<opts>] {<path> [<opts>]}
```

Attributes

Operating System:OS-9 and OS-9 for 68K

Options and Parameters

`<path>`

pathname of the flash device to defragment.

`-q`

Operates in Quiet mode, and return the number of actual sectors available.

`-s[=]<size>`

Specifies the maximum number of physical sectors to recover. The default is 16.

Description

The `ftdefrag` utility optimizes performance of FTL media by reorganizing media data blocks. This reorganization defragments and recycles non-writeable flash areas to achieve optimal writing speed. This operation would typically be performed after deleting one or more large files.

Examples

```
ftdefrag /rrf1@
```

Errors

EOS_BAD_FORMAT
EOS_BTYP
EOS_GENERAL_FAILURE
EOS_WRITE_FAULT

ftformat

Syntax

```
ftformat [<opts>] {<path> [<opts>]}
```

Attributes

Operating System:OS-9 and OS-9 for 68K

Options and Parameters

`<path>`

pathname of the flash device to format

`-p[=]<percent>`

Specifies the percentage of media space to reserve for slack. The default is 1%.

`-q`

Operates in quiet mode and do not prompt.

`-r[=]<size>`

Specifies the size of area to reserve at the start of Flash. The default is 0h.

`-s[=]<size>`

Specifies the Flash array. The default is the value in device descriptor.

`-u[=]<units>`

Specifies the number of erase units to reserve for swapping. The default is 1.

`-v[=]<size>`

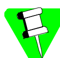
Specifies the Virtual Memory Direct Addressing Limit. Default is 1.

`-y`

Does not ask whether to continue; format immediately.

Description

The ftformat utility performs a low-level format of a flash media device. During a low-level format operation, the rbftl driver erases all units of the flash media and lays down a metadata structure which is used for wear leveling and data block location. Once the media has been low-level formatted, a high-level format can be done by the file manager. The high-level format of the FTL media creates the directory structure and file allocation information. The low-level format is normally a one-time operation, unless the media becomes corrupted through an event such as software or hardware failure.



OS-9 for 68K Users:
On OS-9 for 68K, there is a format-enabled descriptor (rrf1fmt) that can be used if you want to format the media.

Example

```
ftformat /rrf1
```

Errors

EOS_BAD_FORMAT
EOS_<callback>
EOS_VOLUME_TOO_SMALL
EOS_WRITE_FAULT

See Also

`format` in the Utilities Reference

Configuration Fields

In addition to the configuration fields included in RBF, the following FTL fields are included:

- `ds_flash_base`
- `ds_flash_size`
- `ds_flash_source`

ds_flash_base

Default Description Macro

FLASH_BASE

EditMod Labels

1-module header

2-device descriptor data definitions

4-RBF logical unit static storage

5-rbftl specific information

1-flash base address

Description

This is the base address of the flash memory.

Default Value

None

Available Values

Value ranges are hardware dependent. Refer to your OS-9 board guide to see what types and sizes of flash your board has available.

ds_flash_size

Default Description Macro

```
FLASH_SIZE
```

EditMod Labels

```
1-module header
2-device descriptor data definitions
4-RBF logical unit static storage
5-rbftl specific information
2-size of flash to be used
```

Description

This is the size of flash to be used in bytes.

Default Value

None

Available Values

Value ranges are hardware dependent. Refer to your OS-9 board guide to see what types and sizes of flash your board has available.

ds_flash_source

Default Description Macro

`FLASH_SOURCE`

EditMod Labels

```
1-module header
2-device descriptor data definitions
4-RBF logical unit static storage
5-rbftl specific information
3-flash source location
```

Description

Use this parameter to set the flash location, on the motherboard or on a PC Card.

Default Value

None

Available Values

Available values are listed in <links>Table 4-1. ds_flash_source Available Values.

Table 4-1. ds_flash_source Available Values

Description	Macro	EditMod
Board resident-hosted flash	None	0
PC Card-hosted flash	None	1

Errors

The FTL errors described in <links>Table 4-2. FTL Errors are defined in `ftlsrvcb.h`.

Table 4-2. FTL Errors

Number	Name	Description
000:249	EOS_BTYP	Bad type (incompatible media). A read operation was attempted on incompatible media. For example, a read attempt was made on a flash device that was not low-level formatted.
012:019	EOS_WRITE_PROTECT	Write protect. A write operation was attempted on a write-protected device.
012:023	EOS_BAD_FORMAT	Mount error. An error was detected in the FTL structure on the device while mounting the format units.
012:026	EOS_UNKNOWN_MEDIA	Media error. The flash driver MTDs did not recognize the flash hardware type.
012:027	EOS_SECTOR_NOT_FOUND	Sector out of bounds. The specified sector is not on the available media.
012:029	EOS_WRITE_FAULT	Write error. Error occurred during write operation, often a hardware error such as write timeout.
012:031	EOS_GENERAL_FAILURE	Device error. Error was detected while operating with the device structures.
012:063	EOS_NOT_ENOUGH_MEMORY	Mount error. The paramenters specified in the metadata structure of the flash media exceed the memory requirement specified in the same structure. This indicates the FTL structure is not intact.
012:064	EOS_VOLUME_TOO_SMALL	Format error. The descriptor specified the number of units to be less than or equal to the descriptor's first effective unit plus the format-specified number of spare erase units. This indicates a bad descriptor.

5 Porting Guide

This chapter explains how to add support for a new flash part into a rbftl flash driver.

Rbftl drivers consists of two parts: Common code that is already written for your driver, and hardware specific code that you write. The common code includes the wear-balancing algorithm and RBF driver interface, and generally makes up most of the driver. In the hardware specific code, you will need to write functions for identifying, erasing, and writing the flash part you choose.

Common Code Files

Below is a list of files you will need to build your driver:

`rbftl_start.r`	Wear-balancing and RBF driver interface, normally found in the `LIB` directory for your processor.
`identify.c`	Called by the common code when requesting identification from the flash part.
`rbftl_defs.h`	Describes the interface between your hardware specific code and the common code.
`rbftl.des`	File used for making rbftl descriptors.

Hardware-Specific Functions

The hardware-specific functions you need are for identification, erasing, and writing the flash part. Each has a specific prototype:

```
error_code flash_write( u_int32 address, u_int8* buff, u_int32
length, u_int16 overwrite);
```

`address`	Flash part relative address to write data. You must call `flash.map()` to make this a system address.
`buff`	Buffer containing data to write to flash part.
`length`	Number of bytes to write to flash part.
`overwrite`	Used to specify whether or not this area of flash has been erased. No longer used.

```
error_code flash_erase( u_int32 first_block, u_int32 num_of_blocks);
```

`first_block`	This specifies the first block to erase on the flash part. Normally, flash blocks are of a fixed size, such as 64K. To compute the address to erase, multiply `first_block` by the flash block size.
`num_of_blocks`	Number of flash blocks to erase.

```
error_code flash_identify();
```

This routine is responsible for identifying your flash device as well as filling out fields in the global structure called `flash`.

Communication Structure

Communication between the hardware specific and common code happens via a global data structure that is defined in `rbftl_defs.h`. This structure is declared in the common code and is named `flash`. Your flash identification routine is responsible for setting the required fields properly so that the common code can make use of them.

Required Fields

Required fields are referenced by common code.

erasable_block_size	Number of bytes in an erase block.
write	Function pointer to your flash write routine.
erase	Function pointer to your flash erase routine.

Optional Fields

Optional fields are used by other existing hardware specific code.

type	JEDEC ID of flash part.
chip_size	Byes in a single chip in flash array.
num_chips	Number of flash chips.
interleaving	Address difference between two successive words on a chip (for example 2 for two 8 bit parts, 1 for one 16bit part).
parallel_limit	Effective flash byte bus width.
flags	Not used currently.
hw_data	Place you can store data.
read	Function pointer that reads memory off the flash device. The default value is memcpy. You can set this to another value if your flash device has special access needs.
map	Function that converts flash device relative addresses to real memory addresses. Default is returning (address+flash.flash_base).

Fields Specified in the Device Descriptor

The following fields are values specified in the device descriptor and should not be modified.

flash_cache	Address of cached flash memory.
flash_base	Address of uncached flash memory.
flash_size	Number of bytes in the flash device.
flash_source	0—resident, 1—PC Card.
flash_width	8, 16, 32.
flash_mfg_id	Manufacturer's id (optional).
flash_dev_id	Flash device id (optional).

Example Source Code

Sample source code is provided for your reference. This includes skeleton code for writing your hardware specific section, and makefiles to build the driver and descriptor.

The skeleton code is located in the `MWOS/SRC/IO/RBFTL/SAMPLE` directory. It contains three files: `id.c`, `erase.c`, and `write.c`. These files contain function declarations for the hardware specific functions you need to write, as well as useful comments about the interface to the common code. Make a new directory in `MWOS/SRC/IO/RBFTL` and copy these three files there.

As for the ports themselves, the sample systems can be found in the following directory:

- `MWOS/OS9000/SAMPLES/PORTS/RBF/RBFTL` for OS-9
- `MWOS/OS9/68000/PORTS/SAMPLES/RBF/RBFTL` for OS-9 for 68K

You can modify the configuration files and makefiles in the directory and move them to the desired new port.

These sample systems are for example purpose only. They will not work correctly without modifications from you. Also, the real RBFTL driver may have different file name(s).

Driver and Descriptor

The driver is called rbftl, and is in the `CMDS/BOOTOBJS` directory in the `PORT` tree. The descriptors will be generated in the `CMDS/BOOTOBJS/DESC/RBFTL` directory in the `PORT` tree. Makefiles for the driver and descriptor are located in the `RBF/RBFTL` directory of the `PORT` tree.

On an OS-9 system, the `config.des` file specifies descriptor values where you set the flash base address, size, and bus width for your device. Below is an example section that shows how to set these values for a descriptor called `rrf0`. In this case, the base address starts at 0x09000000, the flash size is 16 MB, and the bus width is 32 bits.

```
#if defined (RRF0) /* Rrf0 descriptor */
/* Module header macros */
#define MH_NAME_OVERRIDE "rrf0"
/* Device descriptor common macros */
/* rbf macros */
/* rbftl macros */
#define FLASH_BASE_OVERRIDE      0x09000000
#define FLASH_SIZE_OVERRIDE      0x1000000
#define FLASH_SOURCE_OVERRIDE    0
#define FLASH_WIDTH_OVERRIDE     32
#define FLASH_MFGID_OVERRIDE     0
#define FLASH_DEVID_OVERRIDE     0
#define LUN_OVERRIDE             0
#endif /* RRF0 descriptor */
```

On an OS9 for 68k system, the base address, size, and bus width are specified in the `systype.d` file.

```
********************
* Flash System Memory Definitions
*
* These are used to make descriptors for the flash file system.

FlashDBase      equ $FF800000    Base address of FLASH when
                                 coldstarting from ROM
FlashDSize      equ $200000      size of FLASH memory
FlashSecSize    equ 512          Size of sector size

********************
* Flash disk descriptor definitions
*
* FLASHDesc Port,Size,Source,Width,MFGID,DEVID,Driver name,MTD
name
*
FlashRRF0 macro
 FLASHDesc FlashDBase,FlashDSize,0,8,0,0,rbftl,mtd
 endm
```