TargetFFS™

FLASH FILE SYSTEM

HIGHLIGHTS

- **Royalty Free**
- **Reliable, Re-entrant Embedded File System**
- **Provides POSIX and Standard C API**
- **Use of Flash Memory is Invisible to Applications**
- **Supports Dynamic Creation and Deletion of Files, Directories, and Links**
- **Complete Wear-Leveling**
- **Guaranteed Integrity Across Unexpected Resets**
- **Supports Multiple Volumes**
- **Includes Shell Application**
- **Works with TargetOS**
- **Integrated with CodeWarrior**

Blunk Microsystems provides system software, device drivers, and board support packages to the embedded systems market, both off-the-shelf products and custom work done under contract.

BLUNK
Microsystems

TargetFFS is a reliable, re-entrant embedded file system with a POSIX and ANSI C compliant application program interface. Supports dynamic creation and deletion of files, directories, and links with read and write capability. Not a static ROM-image file system. Use of flash media for the backing store is invisible to the application layer.

Supports the large erasable block sizes typical of flash memory, usually 64KB or larger. Memory is logically divided into 512 byte sectors that are assigned to individual files as needed. Before a block is erased, the sectors in use are copied to another block.

Implements complete wear-leveling to prevent premature failure of the flash media. Erase cycles are spread evenly across all sectors. A wear count is maintained starting with the first time a flash volume is formatted. The current wear count is available to applications via the `vstat()` call.

File system integrity is guaranteed across unexpected shutdowns. Only data written since the last synchronizing operation (`fclose()`, `fflush()`, etc.) can be lost. Directory structures, closed files, and files open for reading are never at risk.

"Thin" hardware dependent layer for maximum ease in porting to new platforms. Supports linearly addressable flash with a 32, 16, or 8-bit interface to the CPU. Wider interfaces and interleaved devices provide higher performance.

Includes an example driver for a 4MB flash memory that uses four 8-bit devices to implement a 32-bit wide interface with erasable blocks of size 256KB.

Supports concurrent use of multiple volumes. Flash volumes can be permanently installed at startup, or added and deleted as removable devices come and go.

Flash memory can be shared between a boot program and a TargetFFS volume. Because flash memory is not accessible while being erased or programmed, the boot program can read files, change directories, etc., but cannot create files, directories, or links. Alternatively, the boot program can copy itself to RAM and jump there before making calls to TargetFFS.

Low priority background task does garbage collection to ensure minimal use of RAM.

Shipped with two sample applications: a shell that supports creating directories, listing directories, etc. and a binary search application. The shell may be extended with user commands.

Developed using TargetOS™, Blunk Microsystems' real-time operating system, the source code is 100% ANSI C and is easily ported to other real-time kernels.

Integrated with CodeWarrior™, the development environment from Metrowerks with an integrated source level debugger, compiler, assembler, linker, editor, and GUI make tool.

Royalty free. Includes full source code, user's manuals, sample applications, and one year of technical support. ▶

CONTACT INFORMATION

- Visit our web site:
  www.blunkmicro.com
- Customer Support:
  (408) 323-9833
- Technical Support:
  (408) 323-1758
- Fax:
  (408) 323-1757
- E-mail:
  sales@blunkmicro.com
- Address:
  6576 Leyland Park Drive
  San Jose, CA 95120
  USA

AVAILABLE COMPONENTS

TargetOS
Real-time, deterministic, multi-tasking, priority-based, preemptive kernel. Includes Standard C library.

TargetTCP
RFC compliant TCP/IP protocol stack providing the standard Berkeley Sockets API.

TargetLAPB
ISO/IEC 7776 protocol stack. Supports exchanging data on point-to-point networks. Provides automatic flow control and data reliability.

The TargetFFS Application Program Interface:

```
int chdir(const char *path);
void clearerr(FILE* stream);
int close(int fid);
int closedir(DIR *dirp);
int fclose(FILE *stream);
FILE *fdopen(int fid, const char
 *mode);
int feof(FILE *stream);
int ferror(FILE *stream);
int fflush(FILE *stream);
int fgetc(FILE *stream);
int fgetpos(FILE *stream, fpos_t
 *pos);
char *fgets(char *s, int n, FILE
 *stream);
int fileno(FILE *stream);
FILE *fopen(const char *filename,
 const char *mode);
int format(char *path);
int fprintf(FILE *stream, const
 char *format, ...);
int fputc(int c, FILE *stream);
int fputs(const char *string, FILE
 *stream);
size_t fread(void *ptr,  size_t
 size, size_t nmemb, FILE *stream);
FILE *freopen(const char *filename,
 const char *mode, FILE *stream);
int fscanf(FILE *stream, const char
 *format, ...);
int fseek(FILE *stream, long offset,
 int mode);
int fsetpos(FILE *stream, const
 fpos_t *pos);
int fstat(int fid, struct stat
 *buf);
long ftell(FILE *stream);
size_t fwrite(const void *ptr, size_t
 size, size_t nmemb, FILE *stream);
int getc(FILE *stream);
int getchar(void);
char *getcwd(char *buf, size_t size);
char *gets(char *s);
int link(const char *existing,
 const char *new);
off_t lseek(int fid, off_t offset,
 int whence);
int mkdir(const char *path, mode_t
 mode);
int mount(char *path);
int open(const char *path, int
 oflag, ...);
DIR *opendir(const char *dirname);
void perror(const char *s);
int printf(const char *format, ...);
int putc(int c, FILE *stream);
int putchar(int c);
int puts(const char *s);
int read(int fid, void *buf,
 unsigned int nbyte);
struct dirent *readdir(DIR *dirp);
int remove(const char *filename);
int rename(const char *old, const
 char *new);
void rewind(FILE *stream);
void rewinddir(DIR *dirp);
int rmdir(const char *path);
int scanf(const char *format, ...);
void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char
 *buf, int mode, size_t size);
int stat(const char *path, struct
 stat *buf);
FILE *tmpfile(void);
char *tmpnam(char *s);
int ungetc(int c, FILE *stream);
int unlink(const char *path);
int unmount(char *path);
int utime(const char *path, const
 struct utimbuf *times);
int vfprintf(FILE *stream, const
 char *format, va_list arg);
int vprintf(const char *format,
 va_list arg);
int vstat(const char *path, union
 vstat *buf);
int write(int fid, const void *buf,
 unsigned int nbyte);
```

■

Price and Licensing Terms

TargetFFS™ is royalty free. Purchasers are granted a non-exclusive license to use the provided source code at a single site. Licensees have the right to disseminate or resell the software in executable format only. The source code and derived object code may not be redistributed or resold.

Pricing is $5,000 for a one-year term license and $15,000 for a perpetual license. The term license allows development use of the software for one year from the date of purchase. License renewal is not required to fix bugs or to duplicate executable code.