MF1362-01

EPSON

CMOS 32-BIT SINGLE CHIP MICROCOMPUTER **E0C33 Family**

CF33 MIDDLEWARE MANUAL

ENERGY SAVING EPSON

SEIKO EPSON CORPORATION

PREFACE

This manual describes the configution and functions of CompactFlash Middleware CF33 for the E0C33 Family, and explains methods for using this middleware. It is targeted to developers of applications for the E0C33 Family of microcomputers.

CONTENTS

1 Outline of CF33 Middleware

CF33 is CompactFlash middleware for the E0C33 Family of microcomputers. It is designed to perform access to the FAT file system on the E0C33 Family chip. Function to access to FAT file system supplied as library functions, which can be used by linking with the target program.
CF33 middleware is ideally suited to the development of applications such as Digital Camera, PDAs, electronic, and stationery.

The main features of the CF33 middleware are given below:

- Interface with memory cards is based on the PCMCIA Ver.2.0 socket service and card service..
- The ATA driver enables you to use the CompactFlash, ATA flash memory card, and memory card or drive with ATA interface as MicroDrive.
- The FAT file system driver enables you to exchange files compatible with MS-DOS Ver.6.x. Also, FAT format drivers are provided for the CompactFlash and PCMCIA ATA cards.

CAUTION

- Be sure to fully evaluate the operation of your application system before shipping. Seiko Epson assumes no responsibility for problems arising from use of this middleware in your commercial products.
- Rights to sell this middleware are owned solely by Seiko Epson. Resale rights are not transferred to any third party.
- All program files included in this package, except sample programs, are copyrighted by Seiko Epson.
  These files may not be reproduced, distributed, modified, or reverse-engineered without the written consent of Seiko Epson.

1.1 Components of the CF33 Package

The contents of the CF33 package are listed below. When unpacking, check to see that all of the following items are included.

(1) Tool disk (CD-ROM)	1 pc.
(2) E0C33 Family CF33 Middleware Manual (this manual)	1 pc. each in English and Japanese
(3) Warranty card	1 pc. each in English and Japanese

1.2 Basic Configuration of CF33 System

Hardware configuration

The basic hardware configuration of the CF33 system is composed of the core E0C33, external memory, and the PCMCIA controller.
For the PCMCIA controller, the -MR-SHPC-01 V1 from Marubun Corporation is used.

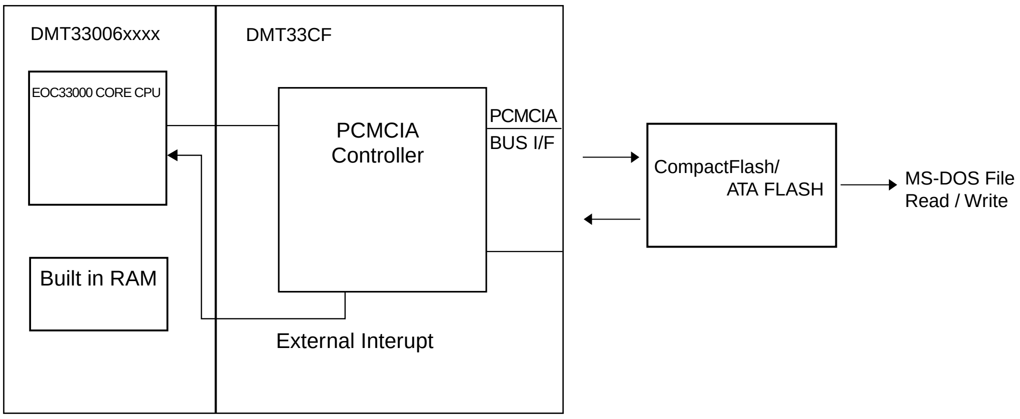


Figure 1.2.1 Hardware Configuration of CF33 System

Software configuration

The CF33 library is middleware, positioned between the E0C33 hardware and user applications, performing hardware control associated with CompactFlash and file access to the FAT file system. By including or linking the top-level functions supplied as C source files into user applications, you can easily perform to call CF33 library functions directly from the user program, and access the CompactFlash and PCMCIA ATA card.

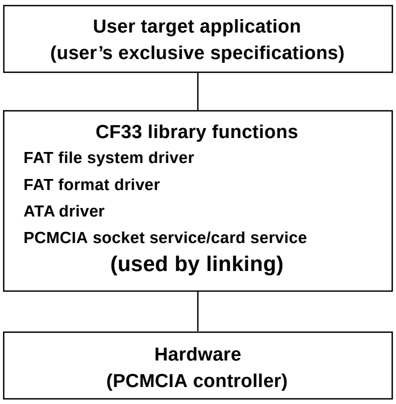


Figure 1.2.2 Software Configuration for Speech Input/Output Unit

For detailed information on CF33 library functions, see Section 4, "CF33 Library Reference".

2 Installation

This section describes the operating environment for the CF33 tools and explains how to install the CF33 middleware.

2.1 Developing Environment

Personal computer

An IBM PC/AT or fully-compatible machine. We recommend a Pentium 90 MHz or higher CPU, and 32 MB or more of RAM. A CD-ROM is required to install tools from the CD-ROM.

Display

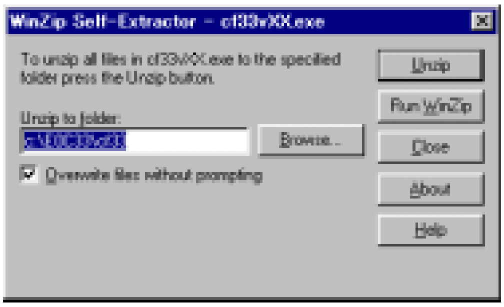
An SVGA (800 × 600) monitor or better. In the Windows Control Panel, choose "Small font" among the display options.

Other

The "E0C33 Family C Compiler Package" is required for software development.

2.2 Method of Installation

The CF33 library is supplied on CD-ROM. Open the self-extracting file on the CD-ROM named "cf33vXX.exe" to install the CF33 library in your computer. (The XX in this file name denotes a version number. For Version 1.0, for example, the file is named "cf33v10.exe".)



WinZip Self-Extractor - cf33vXX.exe

To unzip all files in cf33vXX.exe to the specified folder press the Unzip button.

Unzip to folder: c:\E0C33\cf33

[✓] Overwrite files without prompting

Double-click on "cf33vXX.exe" to start installation. The dialog box shown below appears.
Enter the path and folder name under which you want to install the files in the text box and click on the [Unzip] button. The specified folder is created and all files are copied into it.
If the specified folder already exists in the specified path and [Overwrite Files Without Prompting] is checked (turned on), the files in the folder are overwritten without asking for your confirmation.

The following shows the directories and file configuration after the program files have been copied:

(root)\	
readme.txt	Supplementary explanation, etc. (in English)
readmeja.txt	Supplementary explanation, etc. (in Japanese)
cf33	
lib ..... CF library	
cf.lib	..... CF33 library (socket service, card service, ATA driver, FAT file system driver, and FAT format driver)
include	..... CF include file
cf.h	..... CF header file
src ..... CF33 library's open C source part	
cs_irq.c	..... Card service interrupt functions' C source file
fatchchk.c	..... FAT file system letters check functions' C source file
ss.c	..... Socket service's C source file
demoata ..... ATA card demonstration program	
33208_1.par	..... Parameter file for 33208
bcu.h	..... BCU definition header file
demoata.bat	..... ATA card demonstration program's startup batch file
demoata.c	..... ATA card demonstration program's C source file
demoata.cm	..... ATA card demonstration program's link file
demoata.cmd	..... ATA card demonstration program's command file
demoata.cmx	..... ATA card demonstration program's optimized file
demoata.mak	..... ATA card demonstration program's make file
demoata.srf	..... ATA card demonstration program's SRF file
idma.h	..... DMA header file
init.c	..... Initialization's C source file
int.h	..... Interrupt header file
io.h	..... I/O header file
sys.c	..... Input/Output system functions' C source file
vector.c	..... Vector functions' C source file
demofat ..... FAT file system demonstration program	
33208_1.par	..... Parameter file for 33208
bcu.h	..... BCU definition header file
clock.c	..... Real time clock manager's C source file for file date
ct.h	..... Real time clock's header file
demofat.bat	..... FAT file system demonstration program's startup batch file
demofat.c	..... FAT file system demonstration program's C source file
demofat.cm	..... FAT file system demonstration program's link file
demofat.cmd	..... FAT file system demonstration program's command file
demofat.cmx	..... FAT file system demonstration program's optimized file
demofat.mak	..... FAT file system demonstration program's make file
demofat.srf	..... FAT file system demonstration program's SRF file
idma.h	..... DMA header file
init.c	..... Initialization's C source file
int.h	..... Interrupt header file
io.h	..... I/O header file
sys.c	..... Input/Output system functions' C source file
vector.c	..... Vector functions' C source file

demofmt\ ..... FAT format demonstration program

33208_1.par	..... Parameter file for 33208
bcu.h	..... BCU definition header file
demofmt.bat	..... FAT format demonstration program's startup batch file
demofmt.c	..... FAT format demonstration program's C source file
demofmt.cm	..... FAT format demonstration program's link file
demofmt.cmd	..... FAT format demonstration program's command file
demofmt.cmx	..... FAT format demonstration program's optimized file
demofmt.mak	..... FAT format demonstration program's make file
demofmt.srf	..... FAT format demonstration program's SRF file
idma.h	..... DMA header file
init.c	..... Initialization's C source file
int.h	..... Interrupt header file
io.h	..... I/O header file
sys.c	..... Input/Output system functions' C source file
vector.c	..... Vector functions' C source file

Although you can select a different directory structure and file organization, the discussions in the following pages will assume that the files have been copied from the CD-ROM according to the directory structure given above.

For detailed information on the configuration of sample programs(demoata, demofat, demofmt) and how to use them, see “readme.txt” or “readmeja.txt” .

3 Software Development

This section describes how to develop software. Shown below is the basic flow of development:

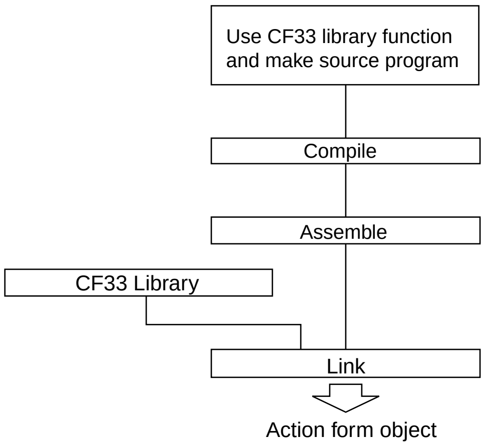


Figure 3.1 Procedure for Developing E0C33 CompactFlash Software

1) Create a user application. Call CF33 library functions from the user program.
2) Compile and assemble the source program.
3) Link the objects generated in Step 2 with the CF33 library. This generates an executable object file.

3.1 Creating an Application and Linking to the CF33 Library

You can implement access to the CompactFlash and PCMCIA ATA card on the E0C33 chip by calls to CF33 library functions. For detailed information on CF33 library functions, see Section 4, "CF33 Library Reference". Sample programs are provided in the "demoxxxx" directory for reference

When creating and linking a program, watch the following:

- Use the 16-bit memory module for CF33 library program code.
- Configure the memory space as 16 bits for the CF33 library's PCMCIA controller.

4 CF33 Library Reference

This section gives a detailed explanation of CF33 library functions, as well as precautions regarding their use.

4.1 Outline of CF33 Library

Functional outline

The CF33 library is a set of CompactFlash functions in srf33 library format, used after linking functions to the target application. The following functions can be executed by calling the required function from the target application:

- Access using the FAT file system on CompactFlash and PCMCIA ATA cards
- FAT file system formatting of CompactFlash and PCMCIA ATA cards

4.2 Hardware Resources and Initialization

Operating clock

CF33 library assumes a value of 40 MHz (typ.) and 20 MHz for Bus Clock for high-speed (OSC3) clock frequency. To change the clock, modify the required parts of the CF33 header file(cf.h).

Memory

The following describes precautions concerning memory:

ROM	Approximately 50KB (card service, socket service, ATA driver, FAT file system driver, and FAT format driver)
RAM	Approximately 15KB (internal variable for card configuration, buffer and internal variable for FAT file system)
STACK	Approximately 1.2 KB

4.3 CF33 Driver and Library Specifications

The specifications of the CF33 driver and library are described below.

- Interrupt functions of the socket service and the card service are open.
- Socket service and card service support only one PCMCIA slot. In the same way, ATA driver, FAT file driver and FAT format driver also support only one driver.
- FAT file system driver support FAT12 and FAT16. It does not support compressed driver and long file name as DoubleSpace and Stacker.
  Also, while ATA card can perform partition configuration, this FAT file system driver supports only operation in the basic DOS area of partition 1.
- FAT format driver performs logical formatting of the FAT file system. It does not perform physical formatting.

4.4 DMT33006LV Board and DMT33CF Board

Jumper and DIP switch configuration

The supplied CF33 library places the memory space for the PCMCIA card controller in 0x2000000..0x23fffff, and the demonstration program in 0xc00000..0xcfffff, by default.
Interrupt signal uses port K62.
Also, the CPU clock runs at 40MHz, and the bus clock at 20 MHz.
Based on the above, the jumper switch configuration is listed as follows:

DMT33006LV :

DSW1	1:ON, 2:OFF	...	Use PLL(x2 40MHz)
	3:ON	...	CPU Bus Clock(2:1)
	4-5:OFF	...	External ROM
	6-7:OFF, 8:ON	...	OSC3 x 1/3
DSW2	1:ON	...	Little endian
	2-8:OFF		
JP5	2-3	...	Area 8 for Flash (*1)
JP6	2-3	...	Area 10 for SRAM (*1)

Other jumpers are set to their default values.

DMT33FCF :

JP1	#CE7/13	...	Set the memory space for CE7/13 PCMCIA controller as area 13. (*1)
S1	SA24-25	...	Adjust to 25-24 bit of memory space address for PCMCIA controller(*1)
	SA24:ON, SA25:OFF		
	RA22-25	...	Decide 22-25 bit of memory space address for PCMCIA controller
	RA22:ON, RA23:ON, RA24:ON, RA25:OFF		

(*1) Change according to the memory space address for PCMCIA controller.

For configuration of the jumper and the DIP switch of the DMT33006LV board, refer to the DMT33006LV manual.
For configuration of the jumper and the DIP switch of the DMT33CF board, refer to the DMT33CF manual.

Memory space for the PCMCIA controller

For the DMT33CF board, the MR-SHPC-01 chip is used as the PCMCIA controller.
The memory allotted for the PCMCIA controller is 0x2000000..0x23fffff. 0x2000000..0x23fffff is used only for the register controlling the PCMCIA controller.
PCMCIA controller specification has bit 21-8 of the system address fixed at 0x3fff.
Also, the specification allocate 0x2000000..0x23fffff to memory window and I/O window in units of 1MB.
Therefore, in the map setting of the ICD33 startup parameter file, add the following address as a RAM area.

Example)

```
; Map allocation (max 31 areas, 256bytes boundary)

RAM 0       1fff       ; internal RAM area 8KB
IO  40000   4ffff      ; internal IO area 64KB
RAM c00000  cfffff     ; external RAM 1MB
RAM 2000000 23fffff    ; PCMCIA controller area 4MB
```

PCMCIA controller interrupt signal

For the PCMCIA controller interrupt signal, SIRQ0 to SIRQ3 are connected to K92, P32, P33, and P35 respectively .
SIRQ0 -> K62
SIRQ1 -> P32
SIRQ2 -> P33
SIRQ3 -> P35

4.5 Make and Run the CF33 Demonstration Program

Make

Three demonstration programs are provided as follows:

- ATA card demonstration program (demoata)
- FAT file system demonstration program (demofat)
- FAT format demonstration program (demofmt)

Open the MS-DOS window, move to the subdirectory containing the demonstration program, and perform make using the make file of the demonstration program.

```
DOS> make -f demoxxxx.mak
```

Make can be run from Workbench wb33. . (Refer to the E0C33 Family C Compiler Package manual)

Customizing the CF33 library

1. If the interrupt functions of the socket service and the card service need to be modified, do so using the open C source files ss.c and cs_inq.c.
2. Extract the necessary objects from the library. Using the -x option of librarian lib33 enables you to restore them to their object files.

   ```
   DOS> lib33 -x cf.lib cs_irq.o
   ```
3. Compile the modified ss.c and cs_inq.c, then add to the library.

   ```
   DOS> lib33 -a cf.lib cs_irq.o
   ```
4. Link the demonstration program.

Run

Connect the CF33 demonstration board to the ICD33 target connector.
Also, connect the PC's COM1 port and LPT1 port to the ICD, using a serial cable and a parallel cable respectively.
Run the batch files for the respective demonstration program.

```
DOS> demoxxxx.bat
```

Debugger runs db33, and starts demonstration.

Contents of the demonstration program

ATA card demonstration program

For the physical center number given by the CompactFlash and the ATA cards, perform data read/write using the given sector size.
Detect whether the card is inserted; if the card is inserted, run the demonstration program to perform data read/write of 2 sectors from physical sector 1. After the demonstration is finished, pulling the card out ends the program. If an error occurs, the program aborts.
Specify the byte number of read/write as multiples of 512 bytes.
Also, after the card is installed, Vcc set voltage is shown in hexadecimal number as ten times of the actual voltage set.

FAT file system demonstration program

For formatted CompactFlash and ATA cards, the program operate the FAT file system.
The program detects whether the card is inserted, and identifies the ATA card automatically. When the ATA card is inserted, the program starts. After the demonstration is finished, remove the card to end the program. If an error occurs, the program aborts.
Also, this program uses the real time clock for file modification date and time. The file update date and time read function(cfReadDate) in the real time clock's C source file for file date is replaced with the one in CF33 library.
The default value of the file modification date and modification time is set to 21:30:00 January 1 2000 by the real time clock initialization function.
The demonstration program performs tasks in the following order.

1. Set current directory
2. Make directory (make the TEST directory)
3. Open file (open the TEST.DAT file in the TEST directory)
4. Write file (Write the data on TEST.DAT file)
5. Read file (Read the data from TEST.DAT file)
6. Seek file (Seek TEST.DAT file in write mode and read mode.)
7. Close file (close the TEST.DAT file in the TEST directory)
8. Get number of files in the directory (number of files in the TEST directory)
9. Get contents of the directory (show files in the TEST directory)
10. Rename file (TEST.DAT file to DEMO.DAT file)
11. Rename directory (TEST directory to DEMO directory)
12. Set file property (property of the DEMO.DAT file)
13. Get file property (property of the DEMO.DAT file)
14. Get the serial number and the volume label

15 Set the serial number and the volume label (serial number 0x01234567, volume label E0C33)

16. Delete file (delete DEMO.DAT file)
17. Delete directory (delete DEMO directory)
18. Get disk space

FAT format demonstration program

After the CompactFlash and ATA cards are inserted, the program starts to perform FAT formatting.
When the message prompts for performing FAT formatting, input the serial number and the volume label to start the format demonstration. If an error occurs, the program aborts.
The program detects whether a card is inserted; if the ATA card is identified (automatically), it starts the demonstration program. If the card is removed, the programs ends. If an error occurs, the program aborts.

4.6 Socket Service Specification

About definition of the socket service and customization of the variables table

User can modify the values listed below by modifying the C source file and the header file of the socket service.

- CPU wait number
- Addresses of the PCMCIA control register and the memory area
- Priority of interrupts used by the socket service and the card service
- Interrupt port used by the PCMCIA controller
- Bus clock cycle

CPU wait number

Modify the following definition of the CF33 header file (cf.h).

```
#define CF_LOOP10_PER_MICRO 12
```

...Approximately 12us/10loop when the CPU clock is running at 40 MHz

In the default setting, it is set to match the time when the wait loop is 10, with the CPU clock running at 40 MHz.

The case of CPU Clock 20Mhz is below.

```
#define CF_LOOP10_PER_MICRO 6
```

...Approximately 6us/10loop when the CPU clock is running at 20 MHz

Modify the definition comparing with the modification in the Wait function(cfWait) of the CPU clock and the socket service C source file.

Modify the PCMCIA control management register address and the memory area address(*2)

Modify the following definition of the CF33 header file(cf.h)

```
#define CF_MEM_BASE        0x02000000
```

This is a base address definition of memory space. A total of 4 MB of memory space must be used: 1 MB for each of the attributes window area, the common window area, the I/O window area, and the PCMCIA controller management register area.

(*2) With DMT33CF, address of PCMCIA memory space is specified by the DMT33CF board jumper and the DIP switch.

Interrupt priority used by the socket service and the card service

Modify the following configuration of the CF33 header file(cf.h)

```
#define CF_INT_PRI_LEVEL  3
```

It's a interrupt priority level of the socket service and the card service. By default, 3 is defined.

Interrupt port used by the PCMCIA controller

Modify the following definition of the CF33 header file(cf.h).

```
#define CF_SIRQ0_K62
```

It's a definition of interrupt port you use.
To change to SIRQ1, modify it to SIRQ1_P32. To change to SIRQ2, modify it to SIRQ2_P33. The interrupt port specification does not allow use of P35(SIRQ3).
Because modifying this part enables the card service C source file(cs_irq.c) to be modified, customize the CF33 library together with the socket service C source file.

Also, modify the interrupt vector address of the user's program.

SIRQ0(K62)	->	port input interrupt 2 (vector address 0x48)
SIRQ1(P32)	->	port input interrupt 6 (vector address 0x118)
SIRQ2(P33)	->	port input interrupt 7 (vector address 0x11c)

Bus clock cycle

Modify the following CF33 header file (cf.h)

```
#define CF_BUS_CT      50   ... 1/20MHz = 50ns
```

Set the bus clock cycle in units of nano second for speed setting of PCMCIA's window area.
The CF33 ATA driver set the attribute window to 700ns, and the I/O window to 300ns.

The case of CPU Clock 20Mhz is below.

```
#define CF_BUS_CT       25   ... 1/40MHz = 25ns
```

4.7 Card Service Specifications

Get the card service status

Use the following function to get the card service status.

Int cfGetCardStatus(void)	...	Card service status acquisition function

Return values are as follows:

`CF_STATUS_CARD_REMOVAL(0x00)`	:	Card insertion
`CF_STATUS_CARD_INSERTION(0x01)`	:	Card removal

Card insertion/removal processing

unsigned char cfCardInsertion(void)	...	Card insertion processing function
unsigned char cfCardRemoval(void)	...	Card removal processing function

Use the return value to see whether card insertion/removal processing is normal.

Set CF33 callback function

Registering card insertion/removal processing to the CF33 callback function enables callback processing on card insertion/removal. You can register one function for each of the events. If more than one function are registered for the same event, the last function registered function becomes available.

Use the following function to register the CF33 callback function.

`Void cfSetCallback(int iType, void (*pFunc)(void))` ... Register CF33 callback

This function registers the card insertion/removal processing as a CF33 callback function.
The first argument indicates a callback event type, as follows:

`CF_CB_REMOVAL(0x00)`	:	Event of the time of removal
`CF-CB_INSERTION(0x01)`	:	Event in the time of insertion

The second argument specifies callback function pointer..

Call CF33 callback function

Use the following function to call the CF33 callback function.

`unsigned char cfCFCallback(void)` ... CF33 Callback Function

It calls the CF33 callback function. If the CF33 callback function is not registered, a return value of normal termination is always returned.

To see whether the card insertion/removal processing is normal, check the return value of the following global variable in the card service interrupt management function during interrupt processing.

`unsigned char cf_bCBRet`

0 is returned for normal operation, or some other value for error.

Also, if the card is removed during access to the card, callback process of card removal is performed; therefore error processing is performed.

4.8 ATA Driver Specifications

ATA drive geometry information

For the ATA drive's FAT formatting, the ATA drive geometry structure shown below is set as global.

```
AtaDrvGeom_t cf_stAtaDrvGeom
```

AtaDrvGeom_t is the geometry structure definition of the ATA drive.

ATA driver voltage specification

The ATA card has the voltage configuration information stored in CIS information. For CompactFlash with its Vcc supporting both 5V and 3.3V voltage, its voltage is set automatically to either one of them according to the voltage configuration of the CIS information.
Also, the Vcc voltage can be found by checking the following global variable.

```
unsigned char cf_bAtaCardVcc
```

CF_VOLT_33 (33) : Vcc = 3.3V
CF_VOLT_50 (50) : Vcc = 5.0V

ATA driver access method

The ATA card has three access methods: memory mode, I/O card, and true IDE mode. The ATA driver operates using the contiguous I/O map system of the I/O card mode. Also, the ATA driver assumes the CPU clock is running at 20 MHz to 80 MHz.

4.9 FAT File System Specifications

Global variable of FAT file system driver

The following is the global variable of the FAT file system driver.

`unsigned char cf_bCurDrv`	... Current Drive Because the CF33 supports only one drive, it is set to 0 during initialization.
`unsigned char cf_bCurDir[64]`	... Current Directory This variable stores the path name of the current directory. It can be up to 64 characters. During initialization, it is set to '\', indicating the root directory.
`unsigned char cf_bMediaChgFlag`	... Media Change Flag During initialization, the media change flag is set to 0 to indicate no media change. When a removable media is changed, it is set to 1 (to indicate there is a change of media). For the CF33, it is always set to 0 as removal media is not used.
`unsigned char cf_bVerifyFlag`	... Verify Flag Verify Flag is set to 0 (OFF) during initialization. When the flag is set to 1(ON), write is performed using the verify write command.
`FatErr_t stFatErr`	... FAT Error Information Structure FatErr_t is defined as follows:

```
typedef struct {
unsigned char bErrCode    // error code
unsigned char bType       // error type
unsigned char bDrvNum     // drive number
unsigned long ulSecNum    // sector number
} FatErr_t;
```

The following error statuses are used set for the error code of the FAT Error Information Structure.

`#define FAT_OK`	`0x00`	... Normal
`#define FAT_INVALID_FUNCTION`	`0x01`	... Invalid function
`#define FAT_FILE_NOTFOUND`	`0x02`	... File is not found
`#define FAT_PATH_NOTFOUND`	`0x03`	... Path is not found
`#define FAT_TOO_MANYOPEN`	`0x04`	... Too many files are open
`#define FAT_ACCESS_DENIED`	`0x05`	... Access is denied.
`#define FAT_INVALID_HANDLE`	`0x06`	... Invalid handle
`#define FAT_INVALID_ACCESSCODE`	`0x0c`	... Invalid access code
`#define FAT_INVALID_DATA`	`0x0d`	... Invalid data
`#define FAT_INVALID_DRIVE`	`0x0f`	... Invalid drive
`#define FAT_ATTEMPT_REMOVECD`	`0x10`	... Attempt to remove current directory
`#define FAT_NOT_SAMEDEVICE`	`0x11`	... Not same device
`#define FAT_NOMORE_FILES`	`0x12`	... No more files
`#define FAT_DSK_WP`	`0x13`	... Disk is write-protected.
`#define FAT_UNKNOWN_DSU`	`0x14`	... Unknown disk unit
`#define FAT_DRV_NOTREADY`	`0x15`	... Drive is not ready
`#define FAT_INVALID_CMD`	`0x16`	... Invalid command
`#define FAT_CRC_ERROR`	`0x17`	... CRC error
`#define FAT_INVALID_LENGTH`	`0x18`	... Invalid data length
`#define FAT_SEEK_ERROR`	`0x19`	... Seek error

```
#define FAT_NOT_DOSDISK           0x1a ... Disk is not for DOS
#define FAT_SECTOR_NOTFOUND       0x1b ... Sector is not found
#define FAT_WRITE_FAULT           0x1d ... Write fault
#define FAT_READ_FAULT            0x1e ... Read fault
#define FAT_GENERAL_FAILURE       0x1f ... General failure
#define FAT_SHARING_VIOLATION     0x20 ... Sharing violation
#define FAT_LOCK_VIOLATION        0x21 ... Lock violation
#define FAT_INVALID_DISKCHANGE    0x22 ... Invalid disk exchange
#define FAT_EMPTY_CLUSTER         0x23 ... No empty cluster
#define FAT_DRIVE_BUSY            0x24 ... Drive is busy
```

The following are set for the FAT Error Information Structure.
Error Type (bit 3-0)

```
#define FAT_ET_GENE               0x00 ... General failure
#define FAT_ET_CHRDEV             0x01 ... Character device is not available
#define FAT_ET_NOFAT              0x02 ... Not FAT file system
#define FAT_ET_BADFNAME           0x03 ... Invalid file name
#define FAT_ET_BADFAT             0x04 ... Invalid FAT, cluster
#define FAT_ET_PARA               0x05 ... Invalid parameter
#define FAT_ET_BPB_RD             0x08 ... BPB area read error
#define FAT_ET_DSK_RD             0x09 ... Cannot read disk
#define FAT_ET_DSK_WR             0x0a ... Cannot write disk
```

Program abort request(bit 4)

```
#define FAT_ET_ABORT              0x10 ...Program abort request(not used in CF33)
```

Area information(bit 7, 6)

```
#define FAT_ET_SYS                0x00 ... System area
#define FAT_ET_FAT                0x40 ... FAT area
#define FAT_ET_RDIR               0x80 ... Directory area
#define FAT_ET_DATA               0xc0 ... Data area
```

Restrictions on the FAT file system driver specification

Files handled by the CF33 file system must all be in the binary mode.
The maximum number of files allowed to open is 20. Because the maximum number of buffers for open files is 20, this also affects the restriction.
Path name of the current directory can be up to 64 characters.
A file name consists of 8 characters plus the 3-character extension. Long name is not supported.

FAT file system driver function

The following FAT file system driver functions are available.
For details, refer to "FAT file system driver specification".

Table 4.9.1 FAT file system driver function

Function name	Description
int cfFatInit(void)	Initialize FAT file system driver
int cfChdir(const char *)	Change current directory (*2)
FatFile_t *fat_fopen(const char *, const char *)	Open file (*1)
int cfFclose(FatFile_t *)	Close file (*1)
size_t cfFread(void *, size_t, size_t, FatFile_t *)	Read file (*1)
size_t cfFwrite(void *, size_t, size_t, FatFile_t *)	Write file (*1)
int cfFseek(FatFile_t *, long, int)	Seek file (*1)
int cfMkdir(const char *)	Make directory (*2)
int cfRmdir(const char *)	Remove directory (*2)
int cfRemove(const char *)	Remove file (*1)
int cfRename(char *, char *)	Rename file (*1)
int cfFeof(FatFile_t *)	Check for file end (*1)
int cfFerror(FatFile_t *)	Check file error (*1)
void cfClearerr(FatFile_t *)	Clear file error (*1)
unsigned cfDosGetFileAttr(const char *, unsigned *)	Get file attributes (*2)
unsigned cfDosSetFileAttr(const char *, unsigned)	Set file attributes (*2)
int cfDosFindFirst(const char *, unsigned, struct find_t *)	Find first file (*2)
int cfDosFindNext(struct find_t *)	Find next file (*2)
int cfDosGetDiskFree(unsigned char, struct diskfree_t *)	Get disk free space (*2)
int cfGetFn(unsigned char *, unsigned long *)	Get number of files
int cfGetVolume(unsigned char, unsigned long **, unsigned char **)	Get serial number/volume label
int cfSetVolume(unsigned char, unsigned long, unsigned char **)	Set serial number/volume label

(*1) Compatible with ANSI's C function
(*2) Compatible with C function of MS-C Ver 6.0 for MS-DOS

File structure of CF33's FAT file system driver is defined as follows:

```
typedef stuct {
    int _flag    // file mode flag
    int _file;   // file descriptor
}FatFile_t;
```

Character code used by the FAT file system driver

Character code not available for the FAT file system driver
The following character codes are not available because of CF33's FAT file system driver inside processing.
* ? . \ null (00H, '\0') space (20H)

* ?	... Used as the wild card character
.	... Used to separate extension
\ null character	... Used in path name to separate file name from directory name.
space character	Do not use for the file and directory name.

However, the space character(20H) can be used in a volume label.

Customize the character code used by the FAT file system driver

By default, the following JIS X208-compliant character codes are used in file name, directory name, and volume label.

Alphabets	A..Z (A small alphabet character(a...z) is converted into capital letter)
Numbers	0..9
Symbols	$ & # % ` ' ( ) - @ ^ _ { } ~ !
First byte of Shift JIS	81H..9FH, E0H..FCH
Second byte of Shift JIS	40H..7EH, 80H..FCH
Kana, Kana symbols of 1 byte long	A0H..DFH

The following symbols are not available.
" * + , . / : ; < = > ? [\ ] |
The space character(20H) cannot be used in file names and directory names, but can be used in volume labels

To meet the supported character standard, modify the value and string for comparison with the condition of the following functions in the FAT file system character inspection functions' C source file (fatchchk.c). in the CF33 library's open C source code section.

`int cfIs2ByteCode1(unsigned char)`: Function to inspect the first byte of the double byte code
In the following conditional text, change the value for comparison with the character code

```
if ((bChr >= 0x81) && (bChr <= 0x9f)) return 1;
if ((bChr >= 0xe0) && (bChr <= 0xfc)) return 1;
```

If it is in the comparison range, 1 is returned.

`int cfIs2ByteCode2(unsigned char)`: Function to check the second byte of the double byte code
In the following conditional text, change the value for comparison with the character code.

```
if ((bChr >= 0x40) && (bChr <= 0x7e)) return 1;
if ((bChr >= 0x80) && (bChr <= 0xfc)) return 1;
```

If it is in the comparison range, 1 is returned.

`int cfIsFnchr(unsigned char)`: Function to inspect 1 byte code
In the following conditional text, change the value for comparison with the character code.

```
if ((bChr >= 0xa0) && (bChr <= 0xdf)) return 1;
```

If it is in the comparison range, 1 is returned.

Also, when symbols are changed, to use a character, add it to the bFnOK[] string, or remove a character from the string if it is mot required.

Modification date and time of file

In the CF33 library, the file update date read function(cfReadDate) is a dummy function.
The user needs to prepare a function to read the modification date and time.
For API specification and return value of the file update date read function, use the same value and specification of cfReadDate function.

4.10 FAT Format Driver Specification

CF33 FAT driver format specification

- Support ATA card only
- Number of bytes per sector : 512 bytes
- Number of sectors per cluster : expressed as power of 2 (FAT12 is fixed to 8, and FAT16 has a minimum of 4)
- Number of entries in the root directory : 512 entries (32 sectors) fixed
- FAT format driver decides and initializes the sizes of the MBR area, the PBR area, FAT the table area, the root directory area, and the data area, to get the maximum data area in the basic DOS area in partition 1.
- These numbers may not match the numbers of sectors or clusters formatted by the format command of MS-DOS or MS-Windows.

4.11 Sample Programs of CF33 Library and FAT File System Driver

The process of FAT file system driver demo program processing is below.

1. Perform initialization of the real time clock. (Only when to use the real time clock)
2. Perform initialization of the socket service and the card service.
3. Set the card installation/removing processing to the CF33 callback function.
   If the card status changes, an interrupt triggers and the callback function for card insertion/removal is called from within the interrupt processing.
4. Perform initialization of the ATA driver, and register the ATA driver as the event handler for the card service.
5. Confirm the status of whether the card is already inserted.
6. If the card is already inserted, perform callback processing of card insertion at this time. If the card is not inserted, get the status of the card service, then check to see whether the card is inserted.
7. After callback processing is done, check to see whether an error has occurred in the callback processing.
8. Perform demonstration of the FAT file system driver. To perform initialization of the FAT file system in the beginning of the text processing part(fatTest), call the FAT file system driver initialization function(cfFatInit), then perform each of the FAT file system driver functions.
9. Get the card service status, and check to see whether the card is removed.

Main function part of C source file(demofat.c) of the FAT file system demonstration

```
#include <stdio.h>
#include <string.h>
#include "cf.h"

/* Prototype */
int main(void);
int fatTest(void);
extern void write_str(char *);
extern void write_hex(unsigned long);
extern void cfSetClock(unsigned long);


#define DATE_TIME 0x0e44151e // 2000.01.01 21:30
#define TEST_SIZE 0x800


/* Global variable */
unsigned char bData[TEST_SIZE];


/****************************************************************************
* main
*    Type :   int
*    Ret val :     Return code  FAT_OK = Success, Others = Error
*    Argument :    void
*    Function :    FAT demonstration main function.
****************************************************************************/
int main(void)
{
    int     iStatus;
    unsigned char bRet;
    write_str("*** FAT File System Demo Program ***\n");
```

```
        write_str("\n");
        /* Initialize real time clock */
        cfSetClock(DATE_TIME);... 1.   Initialize real time cClock

        /* Initialize socket service and card service */
        cfSSInit();...2.   Initialize socket service and card service
        cfCSInit();

        /* Register CF33 callback function */
        cfSetCallback(CF_CB_REMOVAL, cfCardRemoval);...3.   Register callback function
        cfSetCallback(CF_CB_INSERTION, cfCardInsertion);

        /* Initialize ATA */
        if ((bRet = cfAtaInit()) != CS_SUCCESS) {...4.   Initialize ATA driver
                write_str("ATA initialize error = ");
                write_hex(bRet);
                return 1;
        }

        iStatus = cfGetCardStatus();  // Check card is inserted already
        ...5.   Check card is inserted already
        if (iStatus == CF_STATUS_CARD_INSERTION) {
                cf_bCBRet = cfCardInsertion();  // Card insertion
        } else {
                /* Wait for card insertion */
                while (1) {
                        iStatus = cfGetCardStatus(); ... 6.   Wait for card insertion
                        if (iStatus == CF_STATUS_CARD_INSERTION) break;
                }
        }

        write_str("Card insert!\n");
        if (cf_bCBRet != CS_SUCCESS) {  // Callback return check
        ...7.   Check for error after callback processing
                write_str("Card insertion process error = ");
                write_hex(cf_bCBRet);
                return 1;
        }

        /* FAT test */
        if (fatTest() != FAT_OK) {...8.   FAT file system demonstration program
                write_str("FAT error = ");
                write_hex(cf_stFatErr.bErrCode);
                return 1;
        }

        /* Wait for card removal */
        while(1) {
                iStatus = cfGetCardStatus();...9.   Wait for card removal
                if (iStatus == CF_STATUS_CARD_REMOVAL) break;
        }

        write_str("Card remove!\n");
        if (cf_bCBRet != CS_SUCCESS) {  // Callback return check
        ...7.   Check for error after callback processing
                write_str("Card removal process error = ");
                write_hex(cf_bCBRet);
                return 1;
        }
        write_str("\n");

        return FAT_OK;
}
```

4.12 CF33 Library Reference

CF33 library "cf.lib" includes CompactFlash and socket service/card service driver, the ATA driver, the FAT file system driver, and the FAT format driver for PCMCIA ATA card. When they are linked to the user program, formatting of CompactFlash or PCMCIA ATA card, and file access to the FAT file system become available.
The CF33 library functions are listed in Table 4.12.1.

Table 4.12.1 List of CF33 Library Functions

FAT file system driver functions

Function name	Description
int cfFatInit (void)	Initialize FAT file system driver
int cfChdir (const char *)	Change current directory
FatFile_t *cfFopen (const char *, const char *)	Open file
int cfFclose (FatFile_t*)	Close file
size_t cfFread (void *, size_t, size_t, FatFile_t *)	Read file
size_t cfFwrite (void *, size_t, size_t, FatFile_t *)	Write file
int cfFseek (FatFile_t *, long, int)	Seek file
int cfMkdir (const char *)	Make directory
int cfRemove (const char *)	Remove file
int cfRmdir (const char *)	Remove directory
int cfRename (char * char *)	Rename file
int cfFeof (FatFile_t *)	Check for file end
int cfFerror (FatFile_t *)	Check file error
int cfClearerr (FatFile_t *)	Clear file error
unsigined cfDosGetFileAttr (const char *, unsigned *)	Get file attributes
unsigined cfDosSetFileAttr (const char *, unsigned)	Set file attributes
int cfDosFindFirst (const char *, unsigned, struct find_t *)	Find first file
int cfDosFindNext (struct find_t *)	Find next file
int cfDosGetDiskFree (unsigned char, struct diskfree_t *)	Get disk free space function
int cfGetFn (unsigned char *, unsigned long **, unsigned char **)	Get number of files
int cfSetVolume (unsigned char, unsigned long, unsigned char **)	Get serial number/volume label
int cfSetVolume (unsigned char, unsigned long **, unsigned char **)	Set serial number/volume label
int cfIs2byteCode1 (unsigned char)	Check first byte of double byte code
int cfIs2byteCode2 (unsigned char)	Check second byte of double byte code
int cfIsFnChr (unsigned char)	Check 1 byte code
unsigned ling cfReadDate (void)	Check file update date read

FAT file format

Function name	Description
int cfChkFatFmt (unsigned char)	Check FAT format
int cfFatFormat (unsigned char, AtaDrvGeom_t *, unsigned long, unsigned char **)	FAT format

Card service

Function name	Description
void cfCSInit (void)	Initialize card service
void cfPcicInterruptHandler (void)	PCMCIA controller interrupt handler
void cfManageInterruptHandler (void)	Card service interrupt handler
void cfCardInterruptHandler (void)	Card interrupt handler
void cfActivateInterrupt (void)	Activate interrupt card service
unsigned char cfCardInsertion (void)	Process card insertion
unsigned char cfCardRemoval (void)	Process card removal
int cfParseCardStatus (void)	Parse card status
int cfGetCardStatus (void)	Get card service status
unsigned char cfCFCallback (void)	CF33 callback function
void cfSetCallback (int, unsigned char (*) (void))	Register CF33 callback functio

Socket service

Function name	Description
voidcfSSInit (void)	Initialize socket service
unsigned char cfMrshpcSSEntry (enum SS_ServiceCorde_t, void *)	MRSH-PC's socket service entry function
void cfWait (unsigned long)	CF33's wait function

ATA driver

Function name	Description
unsigned char cfAtaInit (void)	Initialize ATA driver
unsigned char cfAtaOpen (unsigned char)	Open ATA driver
unsigned char cfAtaClose (unsigned char)	Close ATA driver
unsigned char cfAtaRead (unsigned char, unsigned long, unsigned long, unsigned char *)	Read ATA driver
unsigned char cfAtaWrite (unsigned char, unsigned long, unsigned long, unsigned char *, unsigned char)	Write ATA driver
unsigned char cfAtaIdet (unsigned char, AtaDrvGeom_t *)	Identify ATA driver

The specifications for each function are explained below. For usage examples, see the source file for demonstration program.

4.12.1 FAT File System

cfFatInit

Function:	Initialize FAT file system driver
Format:	`int cfFatInit(void)`
Parameters:	None
Return value:	`FAT_OK` : Terminated normally `FAT_ERR` : Error
Description:	This function initializes the FAT file system driver. Call it before processing of the FAT file system or performing FAT formatting. CF33 supports only one driver, and the current drive(bCurDrv) is set to 0.

cfChdir

Function:	Change current directory
Format:	`int cfChdir(const char *pcPathName)`
Parameters:	`const char *pcPathName` `in` : Pointer to the path name of the current directory
Return value:	`FAT_OK` : Terminated normally `FAT_ERR` : Error
Description:	This function sets a given path as the current directory. The current directory is set to the global variable bCurDir. When the FAT file system driver is initialized, '\' is set, indicating use of the root directory. This function is compatible with chdir( ) of MS-C Ver6.0 for MS-DOS.

cfFopen

Function: Open file

Format: `FatFile_t *cfFopen(const char *pcPathName, const char *pcOpenMode)`

Parameters:
`const char *pcPathName in`: Pointer to the path name of a file
`const char *pcOpenMode in`: Open mode of the file.
One of the following strings can be used to specify the file open mode.

"r", "rb"	: Open the file in the read mode.
"w", "wb"	: Create a new file and open it in the write mode. If a file with the same name already exists, its contents are discarded.
"a", "ab"	: Open the file in append mode. If the file does not exist, create a new file.
"r+", "r+b"	: Open the file in the read and write mode.
"w+", "w+b"	: Create a new file, and open it in the read and write mode. If a file with the same name already exists, its contents are discarded.
"a+", "a+b"	: Open the file in the read and write mode. If the file does not exist, create a new file.

Return value:

`(FatFile_t *)0`	: File open error
Others	: File pointer

Description: This function opens a file with the specified file name for input and output.
All of the files are handled as binary file.
This function is compatible with ANSI C function fopen( ).

cfFclose

Function: Close file

Format: `int cfFclose(FatFile_t *fp)`

Parameters: `const char *fp in` : File pointer of the file to close

Return value:

`FAT_OK`	: Terminated normally
`FAT_ERR`	: Error

Description: This function closes the file specified by a file pointer
This function is compatible with ANSI C function fclose( ).

cfFread

Function: Read file

Format: `size_t cfFread(void *pvReadBuf, size_t tSize, size_t tCnt, FatFile_t*fp)`

Parameters:

`void *pvReadBuf`	`in`	: Space for storage data
`size_t tSize`	`in`	: Item size in bytes
`size_t tCnt`	`in`	: Maximum number of the items to read
`FatFile_t *fp`	`in`	: File pointer

Return value: Total number of read items.
If tSize or tCnt is 0, it returns 0.
If this number is smaller than tCnt, there is some possibility that an error's occurring or reaching the end of the file. To distinguish the reading error and the end of the file, it is needed to use cfFeof function or cfFerror function.

Description: This function reads the item size data for the number of items from the specified file pointer to the data storing space.
This function is compatible with ANSI C function fread( ).

cfFwrite

Function: Write file

Format: `size_t cfFwrite(void *pvWriteBuf, size_t tSize, size_t tCnt, FatFile_t *fp)`

Parameters:

`void *pvWriteBuf`	`in`	: Pointer to data to write
`size_t tSize`	`in`	: Item size in bytes
`size_t tCnt`	`in`	: Maximum number of items to write
`FatFile_t *fp`	`in`	: File pointer

Return value: Total number of written items
If tSize or tCnt is 0, it returns 0.
If this number is smaller than tCnt, there is some possibility that an error's occurring or reaching the end of the file.To distinguish the writing error and the end of the file, it is needed to use cfFeof function or1 cfFerror function.

Description: This function writes the item size data for the number of items from the specified file pointer to the data writing pointer.
This function is compatible with ANSI with C function fwrite( ).

cfFseek

Function:	Seek file
Format:	`int cfFseek(FatFile_t *fp, long lOffset, int iOrigin)`

Parameters:

`FatFile_t *fp`	`in/out`	: File pointer of file to seek
`long lOffset`	`in`	: Offset value
`int iOrigin`	`in`	: Position of file pointer
	`SEEK_SET`	: Absolute position from front of file
	`SEEK_CUR`	: Position from front of file.
	`SEEK_END`	: Relative position from end of file

Return value:

`FAT_OK`	: Terminated normally
`FAT_ERR`	: Error

Description: This function changes the next read/write position of the file indicated by file pointer.
The new position is located at lOffset bytes from the front or back of the iOrigin position.
This function is compatible with ANSI C function fseek( ).

cfMkdir

Function: Create directory

Format: `int cfMkdir(const char *pcPathName)`

Parameters: `const char *pcPathName` `in` : Pointer to the path name of the directory to create

Return value:

`FAT_OK`	: Terminated normally
`FAT_ERR`	: Error

Description: This function creates a directory with the given path name.
This function is compatible with mkdir( ) of MS-C Ver6.0 for MS-DOS.

cfRemove

Function: Remove file

Format: `int cfRemove(const char *pcPathName)`

Parameters: `const char *pcPathName` `in` : Pointer to path name of file to remove

Return value:

`FAT_OK`	: Terminated normally
`FAT_ERR`	: Error

Description: This function removes a file specified by the path name.
This function is compatible with ANSI C function remove( ).

cfRmdir

Function: Remove directory

Format: `int cfRmdir(const char *pcPathName)`

Parameters: `const char *pcPathName` `in` : Pointer to path name of directory to remove

Return value:

`FAT_OK`	: Terminated normally
`FAT_ERR`	: Error

Description: This function removes the directory of the given path name.
This function is compatible with rmdir( ) of MS-C Ver6.0 for MS-DOS.

cfRename

Function:	Rename file
Format:	`int cfRename(char *pcOldpath, char *pcNewPath)`
Parameters:	`char *pcOldName` `in` : Pointer to the path name of the old file `char *pcNewName` `in` : Porinter to the path name of the new file
Return value:	`FAT_OK` : Terminated normally `FAT_ERR` : Error
Description:	This function renames a file name to the given name. This function is compatible with ANSI C function rename( ).

cfFeof

Function:	Check for end of file
Format:	`int cfFeof(FatFile_t *fp);`
Parameters:	`FatFile_t *fp` `in` : File pointer of file
Return value:	None 0 : End of file 0 : Not end of file
Description:	This function checks to see whether the file specified by the file pointer after read/write has reached its end. This function is compatible with ANSI C function feof( ).

cfFerror

Function:	Check file error
Format:	`int cfFerror(FatFile_t *fp);`
Parameters:	`FatFile_t *fp` `in` : File pointer of file
Return value:	Except 0 : Error 0 : Normal
Description:	This function checks to see whether an error has occurred on the file specified by the file pointer after read/write process. This function is compatible with ANSI C function ferror( ).

cfClearerr

Function:	Clear file error
Format:	`void cfClearerr(FatFile_t *fp);`
Parameters:	`FatFile_t *fp` `in` : File pointer of the file
Return value:	None
Description:	This function clears the file error or file end state of a file specified by the file pointer. This function is compatible with ANSI C function clearerr( ).

cfDosGetFileAttr

Function: Get file attributes

Format: `unsigned cfDosGetFileAttr(const char *pcPathName, unsigned *puiAttributes)`

Parameters:

`const char *pcPathNam`	`in`	: Pointer to path name of a file to get file attributes
`unsigned *puiAttributes`	`out`	: Pointer to file attributes to get

Return value:

`FAT_OK`	: Terminated normally
`FAT_ERR`	: Error

Description: Get file attributes of a specified file.
File attributes are defined as follows:

_A_NORMAL (0x00)	: Normal file
_A_RDONLY (0x01)	: Read-only
_A_HIDDEN (0x02)	: Hidden file
_A_SYSTEM (0x04)	: System file
_A_VOLID (0x08)	: Volume ID
_A_SUBDIR (0x10)	: Sub directory
_A_ARCH (0x20)	: Archive

This function is compatible with _dos_getfileattr( ) of MS-C Ver 6.0 for MS-DOS.

cfDosSetFileAttr

Function: Set file attributes

Format: `unsigned cfDosSetFileAttr(const char *pcPathName, unsigned uiAttributes)`

Parameters:

`const char *pcPathName`	`in`	: Pointer to the path name of the file which sets the file attributes
`unsigned uiAttributes`	`in`	: File attributes to set

Return value:

`FAT_OK`	: Terminated normally
`FAT_ERR`	: Error

Description: This function sets the file attributes of the specified file.
For details on file attributes, refer to the cfDosGetFileAttr function.
This function is compatible with _dos_setfileattr( ) of MS-C Ver 6.0 for MS-DOS.

cfDosFindFirst

Function: Find first file

Format:
```
int cfDosFindFirst(const char *pcPathName, unsigned uiAttributes,
struct find_t *pstFind)
```

Parameters:

`const char *pcPathName`	`in`	: Pointer to path name of file to search
`unsigned uiAttributes`	`in`	: File attributes of file to serch
`struct find_t *pstFind`	`in`	: Pointer to File search structure

find_t structure is defined as follows:
```
struct find_t {
char reserved[FAT_FIND_REVLEN];// Area reserved for use by this function
char attrib;                   // Attributes
unsigned wr_date;              // Modification date
long size;                     // File size
char name[FAT_FIND_LEN];       // File name
}
```
For the format of modification date and time, refer to the cfReadDate function.

Return value:

`FAT_OK` :	Terminated normally
`FAT_ERR` :	Error

Description: This function finds the first file matching the given file name and file attributes.
For the file name, wild card(*,?) can be used.
For details on file attributes, refer to the cfDosGetFileAttr function.
This function is compatible with _dos_findfirst( ) of MS-C Ver 6.0 for MS-DOS.

cfDosFindNext

Function: Find next file

Format: int cfDosFindNext(struct find_t *pstFind)

Parameters:

`struct find_t *pstFind`	`in`	: Pointer to File search structure

Return value:

`FAT_OK` :	Terminated normally
`FAT_ERR` :	Error

Description: This functions finds the next file matching the find_t structure set by the cfDosFndFirst function.
This function is compatible with _dos_findnext( ) of MS-C Ver 6.0 for MS-DOS.

cfDosGetDiskFree

Function: Get disk free space

Format:
```
int cfDosGetDiskFree(unsigned char bDrvNum, struct diskfree_t
*pstdiskfree)
```

Parameters:

`cunsigned char bDrvNum`	`in`	: Drive number (Set current drive)
`struct diskfree_t *pstdiskfree`	`out`	: Pointer to Get disk free space structure

diskfree_t structure is defined as follows:
```
struct diskfree_t {
unsigned total_clusters;          // Total number of clusters
unsigned avail_clusters;          // Number of available clusters
unsigned sectors_per_cluster;     // Number of sectors per cluster
unsigned bytes_per_sector;        // Numbers of bytes per sector
};
```

Return value:

`FAT_OK` :	Terminated normally
`FAT_ERR` :	Error

Description: This function gets the disk free space of the specified file.
This function is compatible with _dos_getdiskfree( ) of MS-C Ver 6.0 for MS-DOS.

cfGetFn

Function: Get number of files

Format:
```
int cfGetFn(unsigned char *pbPath, unsigned long *pulTotalEnt)
```

Parameters:

`unsigned char *pbPath`	`in`	: Pointer to the specified directory string
`unsigned long *pulTotalEnt`	`out`	: Variable pointer of the number of all directory entries that includes volume label and removed entry

Return value:

`FAT_ERR`	: Error
Value over 0	: Number of the files in the specified directory

Description: This function gets the total number of the directories and files in the directory of the specified path name.
In the sub-directory, "." or ".." are also counted as directories.

cfGetVolume

Function:	Get serial number/volume label
Format:	`int cfGetVolume(unsigned char bDrvNum, unsigned long **pulSerID, unsignedchar **pbVolume)`

Parameters:

`unsigned char bDrvNum`	`in`	: Drive number (Set the current drive)
`unsigned long **pulSerID`	`in/out`	: Pointer to serial number
`unsigned char **pbVolume`	`in/out`	: Pointer to Volume label string.

Return value:

`FAT_OK` :	Terminated normally
`FAT_ERR` :	Error

Description: This function gets the serial number and volume label of the specified drive.
If the variable pointer of the acquired serial number is 0, then this indicates there is no serial number.
To the variable pointer of the acquired volume label, a string set in the volume label is returned.
If the string is not set in the volume label, then the variable pointer of the volume label becomes 0.

cfSetVolume

Function:	Set serial number/volume label
Format:	`int cfSetVolume(unsigned char bDrvNum, unsigned long ulSerID, unsigned char **pbVolume)`

Parameters:

`unsigned char bDrvNum`	`in`	: Drive number (set current drive)
`unsigned long ulSerID`	`in`	: Serial number
`unsigned char **pbVolume`	`in/out`	: Pointer to string of volume label(in)/Pointer to string of volume label

Return value:

`FAT_OK` :	Terminated normally
`FAT_ERR` :	Error

Description: This function sets the serial number and volume label to the specified drive.
Also, if you specify the volume label and set the pointer of the volume label to 0, the character string of "NO NAME   " goes into the volume label.
You can use characters up to 11 for the volume label; extra characters are discarded.

cfIs2ByteCode1

Function:	Check first byte of double byte code.
Format:	`int cfIs2ByteCode1(unsigned char bChr)`
Parameters:	`unsigned char bChr` `in` : Characters to check

Return value:

`FAT_OK` :	Terminated normally
`FAT_ERR` :	Error

Description: This function check the first byte of the double byte code.

cfIs2ByteCode2

Function: Check second byte of double byte code

Format: `int cfIs2ByteCode2(unsigned char bChr)`

Parameters: `unsigned char bCh` `in` : Character to check

Return value: `FAT_OK` : Terminated normally
`FAT_ERR` : Error

Description: This function check the second byte of the double byte code.

cfIsFnChr

Function: Check 1 byte code

Format: `int cfIsFnChr(unsigned char bChr)`

Parameters: `unsigned char bChr` `in` : Character to check

Return value: `FAT_OK` : Terminated normally
`FAT_ERR` : Error

Description: This function check the 1 byte character code.

cfReadDate

Function: Read file update date and time

Format: `unsigned long cfReadDate(void)`

Parameters: None

Return value: unsigned long File update's date and time for a file of the FAT file system
top 6 bits : Modification date
6 bits : Modification time

The format of the modification date and time are as follows:
Modification date:

15 9	8 5	4 0
Year	Month	Date

Year : 0...127(value obtained by subtracing 1980)
Month : 1...12
Date : 1...31

Modification time:

15 11	10 5	4 0
Hour	Minute	Second

Hour : 0...23
Minute : 0...59
Second : 0...29 (Half of the actual seconds)

Description: This function reads the date and time of a file which has been updated.
In the CF33 library, this function is a dummy function which returns 00:00:00 January 1 2000.
The user needs to prepare a function to read the data and time a file is updated.
Make sure to meet Data format of Modified date and modifying time to the Return Value's specification above.

4.12.2 FAT Format

cfChkFatFmt

Function:	Check FAT format
Format:	`int cfChkFatFmt(unsigned char bDrvNum)`
Parameters:	`unsigned char bDrvNum` `in` : Drive number (Set the current drive)
Return value:	`FAT_OK` : Terminated normally `FAT_NOT_DOSDISK` : Not FAT format
Description:	This function check whether format of a disk is FAT.

cfFatFormat

Function: FAT format

Format:

```
int cfFatFormat(unsigned char bDrvNum, AtaDrvGeom_t
*pstAtaDrvGeom, unsigned long ulSerID, unsigned char **pbVolume)
```

Parameters:

`unsigned char bDrvNum`	`in`	: Drive number (Set current drive)
`AtaDrvGeom_t *pstAtaDrvGeom`	`in`	: Pointer to ATA drive geometry structure
`unsigned long ulSerID`	`in`	: Serial number
`unsigned char **pbVolume`	`in/out`	: Pointer to volume label string (in)/Pointer to the volume label character string (out)

AtaDrvGeom_t structure is defined as follows:

```
typedef struct {
   unsigned short wBytesPerSector;        // Number of bytes per sector
   unsigned short wCylinders;             // Number of cylinders
   unsigned short wHeads;                 // Number of heads
   unsigned short wSectorsPerCylinder;    // Number of sectors per cylinder
   unsigned long ulTotalSectors;          // Total number of sectors
} AtaDrvGeom_t;
```

Return value:

`FAT_OK`	: Terminated normally
`FAT_ERR`	: Error

Description: This function performs logical formatting of the ATA card, using the FAT file system specified by the serial number and volume label.
Also, if you specify the volume label and set the pointer of the volume label to 0, then the string of "NO NAME   " goes into the volume label.
You can use up to 11 characters for the volume label; extra characters are discarded.

4.12.3 Card Service

cfCSInit

Function: Initialize card service

Format: `void cfCSInit(void)`

Parameters: None

Return value: None

Description: This function initializes the card service, call the card service interrupt active function(cfActiveInterrupt), and enables the interrupt for the card service. It initializes the callback function pointer of CF33 to 0.
Also, it checks whether the card is inserted already.
Be sure to call this function after the socket service initialization function(cfSSInit).

cfPcicInterruptHandler

Function: PCMCIA controller interrupt process

Format: `void cfPcicInterruptHandler(void)`

Parameters: None

Return value: None

Description: This function performs interrupt of the PCMCIA controller.
In the CF33, the interrupt where the card status of the PCMCIA controller changes and the card interrupt share the same interrupt number.

cfManageInterruptHandler

Function: Manage card service interrupt handler

Format: `int cfManageInterruptHandler(void)`

Parameters: None

Return value: 1 : Card service interrupt exists(Card status change exists)
0 : No card service interrupt(No card status change)

Description: This function manages the card service interrupt handler.
Card service interrupt occurs when the card status changes.

cfCardInterruptHandler

Function: Manage card interrupt handler

Format: `void cfCardInterruptHandler(void)`

Parameters: None

Return value: None

Description: This function performs interrupt of the card's driver.

cfActivateInterrupt

Function:	Activate card service interrupt
Format:	`void cfActivateInterrupt(void)`
Parameters:	None
Return value:	None
Description:	This function enables the card service's interrupt.

cfCardInsertion

Function:	Process card insertion
Format:	`unsigned char cfCardInsertion(void)`
Parameters:	None
Return value:	`CS_SUCCESS` : Terminated normally Others : Error of callback event handler
Description:	This function processes the callback event when the card is inserted.

cfCardRemoval

Function:	Process card removal
Format:	`unsigned char cfCardRemoval(void)`
Parameters:	None
Return value:	`CS_SUCCESS` : Terminated normally Others: Error of callback event handler
Description:	This function processes the callback event when the card is removed.

cfParseCardStatus

Function:	Parse card status
Format:	`int cfParseCardStatus(void)`
Parameters:	None
Return value:	0 : No card status change 1 : Card status has changed
Description:	This function checks to see whether the card status has changed after the card service interrupt process.

cfGetCardStatus

Function: Get card service status

Format: `int cfGetCardStatus(void)`

Parameters: None

Return value: `CF_STATUS_CARD_REMOVAL(0x00)` : Card removal
`CF_STATUS_CARD_INSERTION(0x01)` : Card insertion

Description: This function gets the card service status.

cfCFCallback

Function: CF33 callback

Format: `unsigned char cfCFCallback(void)`

Parameters: None

Return value: `CS_SUCCESS` : Terminated normally
Others : Error of callback event handler

Description: This function performs the CF33 callback function. If the CF33 callback function is not regis tered, the return value for normal termination is returned.

cfSetCallback

Function: Entry CF33 callback function

Format: `void cfSetCallback(int iType, unsigned char (*pFunc)(void))`

Parameters: `int i Type in` : Callback event type
`CF_CB_REMOVAL(0x00)` : Event when card is removed
`CF_CB_INSERTION(0x01)` : Event when card is inserted
`void (*pFunc)() in` : Address of callback function

Return value: None

Description: This function registers the callback function of the CF33. You can register one function for each of the events. If more than one function are registered for the same event, the last function registered becomes available.
As the CF33 callback function pointer is initialized to 0 during the card service initialization(cfCSInit function) process, be sure to register the callback function after the card service initialization.

4.12.4 Socket Service

cfSSInit

Function: Initialize socket service

Format: `void cfSSInit(void)`

Parameters: None

Return value: None

Description: This function initializes the socket service.

cfMrshpcSSEntry

Function: Entry socket service for MRSH-PC

Format: `unsigned char cfMrshpcSSEntry(enum SS_ServiceCode_t enCode, void *pvArg)`

Parameters:

`enum SS_ServiceCode_t enCode`	`in`	: Socket service number
`void *pvArg`	`in`	: Pointer to argument of socket service function

Return value:

`SS_SUCCESS`	: Terminated normally
`SS_BAD_SERVICE`	: Unsupported
Others	: Error of socket service

Description: This is the socket service function for MRSH-PC .
The socket service is turned into a table in the array SS_ServiceTable[]. The socket service is called from this table.

cfWait

Function: CF33 wait

Format: `void cfWait(unsigned long ulMicro)`

Parameters: `unsigned long ulMicro   in : Micro-second`

Return value: None

Description: This is a wait function for CF33.
The time unit is 1 micro-second.
This function waits for the specified amount of time, in micro-seconds.
This function is implemented by loop. If you implement a clock, be sure to set in the unit of 1 micro-second.
The source is in src\ss.c.

4.12.5 ATA Driver

cfAtaInit

Function: Initialize ATA driver

Format: `unsigned char cfAtaInit(void)`

Parameters: None

Return value: `CS_SUCCESS` : Terminated normally
Other values indicate error codes of the socket service or the card service.

Description: This function initializes the ATA driver.

cfAtaOpen

Function: Open ATA driver

Format: `unsigned char cfAtaOpen(unsigned char bDrive)`

Parameters:

`unsigned char bDrive in`	: Drive number of ATA card
`0xff`	: Drive is not used
Other than 0xff	: Drive number

Return value: `CS_SUCCESS` : Terminated normally
Other values indicate error codes of the socket service or the card service.

Description: This function opens the ATA driver.

cfAtaClose

Function: Close ATA driver

Format: `unsigned char cfAtaClose(unsigned char bDrive)`

Parameters:

`unsigned char bDrive in`	: Drive number of ATA card
`0xff`	: Drive is not used
Other than 0xff	: Drive number

Return value: `CS_SUCCESS` : Terminated normally
values indicate error codes of the socket service or the card service.

Description: This function closes the ATA driver.

cfAtaRead

Function:	Read ATA driver
Format:	`unsigned char cfAtaRead(unsigned char bDrive, unsigned long ulSecNum,unsigned long ulCount, unsigned char *pbDest)`

Parameters:

`unsigned char bDrive`	`in`	: Drive number of ATA card
`0xff`		: Drive is not used
Other than 0xff		: Drive number
`unsigned long ulSecNum`	`in`	: Physical sector number
`unsigned long ulCount`	`in`	: Number of bytes to read
`unsigned char *pbDest`	`in`	: Data buffer pointer to read

Return value: `CS_SUCCESS` : Terminated normally
Others values indicate error codes of the socket service or the card service.

Description: This function reads the specified size from the physical number of specified drive number. Set the reading bytes in the multiple number of 512 bytes. The maximum bytes number that cfAtaRead function can read in a time is 0x50000 bytes(256 sectors). The transmitting counts that is set in ATA register is 0x01..0xff (from 1 sector to 255 sectors), and 0x00 means 256 sectors. Only one-byte long transmitting count can be set although you set the value more than 0xff.

cfAtaWrite

Function:	Write ATA driver
Format:	`unsigned char cfAtaWrite(unsigned char bDrive, unsigned long ulSecNum, unsigned long ulCount, unsigned char *pbSrc, unsigned char bVerify)`

Parameters:

`unsigned char bDrive`	`in`	: Drive number of ATA drive
`0xff`		: Drive is not used
Other than 0xff		: Drive number
`unsigned long ulSecNum`	`in`	: Physical sector number
`unsigned long ulCount`	`in`	: Number of bytes to write
`unsigned char *pbSrc`	`in`	: Data buffer pointer to write
`unsigned char bVerify`	`in`	: Verify flag 0:Off, 1:On

Return value: `CS_SUCCESS` : Terminated normally
Others values indicate error codes of the socket service or the card service.

Description: This function writes the given size to the physical number of the given drive number. If the Verify lag is On, performs the Verify Write. Set the writing bytes in the multiple number of 512 bytes. The maximum bytes number that cfAtaRead function can write in a time is 0x50000 bytes(256 sectors).
The transmitting counts that is set in ATA register is 0x01..0xff (from 1 sector to 255 sectors), and 0x00 means 256 sectors. Only one-byte long transmitting count can be set although you set the value more than 0xff.

cfAtaIdent

Function: Identify ATA driver

Format: unsigned char cfAtaIdent(unsigned char bDrive, AtaDrvGeom_t *pstAtaDrvGeom)

Parameters:

`unsigned char bDrive`	`in`	: Drive number of ATA card
`0xff`		: Drive is not used
Other than 0xff		: Other than 0xff: Drive number
`AatDrvGeom_t *pstAtaDrvGeom`	`in`	: Pointer to ATA driver geometry information structure

For AarDrvGeom_t structure, refer to the fatFormat function.

Return value: `CS_SUCCESS` : Terminated normally
Others values indicate error codes of the socket service or the card service.

Description: This function identifies the ATA driver, and gets information of the cylinder, header, and sector.

4.13 Library Performance

The explanation assumes here that the following conditions are used when performing file access of the FAT file system.

FAT file system speed based on CF33

Measurement requirements	: CPU clock at 40 MHz, bus clock at 20 MHz, MELCO RCF-A is used as the PCMCIA adapter. Program area BCU output disable is 0.5 PCMCIA memory area BCU output disable is 3.5 and wait 3 Read/write of a 1MB file at 512 bytes per operation is repeated 2048 times

Speed gap caused by difference in memory cards

Program area BCU wait 1, stack interior

- MALCO 8MB CompactFlash RCF-C (Standard value)

Format	0.13s
1MB file read	12.91s
1MB file write	16.30

- EPSON 45MB CompactFlash SECF-A45

Format	0.67s
1MB file read	21.01s
1MB file write	28.96s

- MELCO 340MB Microdrive RMD-CA340M

Format	5.74s
1MB file read	8.23s
1MB file write	33.43s

Access speed of FAT varies depending on the number of clusters and sectors of FAT.
Also, it is greatly influenced by the characteristics of the device .

When the stack is external RAM of wait1

- MELCO 8MB CompactFlash RCF-C

Format	0.14s
1MB file read	16.46s
1MB file write	20.06s

When compared with the standard value, the speed is 1.1 times for format, and 1.2 times for read/write..

External memory is stacked internally (same conditions as ROM and RAM) at wait 2

- MELCO CompactFlash RCF-C

Format	0.16s
1MB file read	18.40s
1MB file write	22.28s

When compared with the standard value, the speed is 1.2 times for format, and 1.4 times for read/write.

Process speed when some objects are is copied to the internal RAM of wait 1

(fatsub.o internal RAM copy program area BCU wait 1, stack internal)

•MELCO 8MB CompactFlash RCF-C

Format	0.13s
1MB file read	7.89s
1MB file write 1	1.29s

When compared with the standard value, the speed for format does not change, but it is 0.6 times for read and 0.7 times for write.

Action check card

Check the action of the following cards on DMT33CF.

EPSON	8MB CompactFlash	SEATA-A08
MELCO	8MB CompactFlash	RCF-C
HAGIWARA	8MB CompactFlash	HPC-CF08X
LEXAR	8MB CompactFlash	CF8MB x4
I/O Data	10MB CompactFlash	PCCF-10MS
SanDisk	16MB CompactFlash	CF16MB
EPSON	45MB CompactFlash	SECF-A45
MELCO	340MB MicroDrive	RMD-CA340M

4.14 Precautions

Make sure to use 16-bit memory for the program code of the CF33 library. Also, set the PCMCIA controller memory space of the CF33 library to 16 bits.

EPSON **International Sales Operations**

AMERICA

EPSON ELECTRONICS AMERICA, INC.

- HEADQUARTERS -
1960 E. Grand Avenue
El Segundo, CA 90245, U.S.A.
Phone: +1-310-955-5300 Fax: +1-310-955-5400

- SALES OFFICES -

West
150 River Oaks Parkway
San Jose, CA 95134, U.S.A.
Phone: +1-408-922-0200 Fax: +1-408-922-0238

Central
101 Virginia Street, Suite 290
Crystal Lake, IL 60014, U.S.A.
Phone: +1-815-455-7630 Fax: +1-815-455-7633

Northeast
301 Edgewater Place, Suite 120
Wakefield, MA 01880, U.S.A.
Phone: +1-781-246-3600 Fax: +1-781-246-5443

Southeast
3010 Royal Blvd. South, Suite 170
Alpharetta, GA 30005, U.S.A.
Phone: +1-877-EEA-0020 Fax: +1-770-777-2637

EUROPE

EPSON EUROPE ELECTRONICS GmbH

- HEADQUARTERS -
Riesstrasse 15
80992 Munich, GERMANY
Phone: +49-(0)89-14005-0 Fax: +49-(0)89-14005-110

- GERMANY -
SALES OFFICE
Altstadtstrasse 176
51379 Leverkusen, GERMANY
Phone: +49-(0)2171-5045-0 Fax: +49-(0)2171-5045-10

- UNITED KINGDOM -
UK BRANCH OFFICE
Unit 2.4, Doncastle House, Doncastle Road
Bracknell, Berkshire RG12 8PE, ENGLAND
Phone: +44-(0)1344-381700 Fax: +44-(0)1344-381701

- FRANCE -
FRENCH BRANCH OFFICE
1 Avenue de l' Atlantique, LP 915 Les Conquerants
Z.A. de Courtaboeuf 2, F-91976 Les Ulis Cedex, FRANCE
Phone: +33-(0)1-64862350 Fax: +33-(0)1-64862355

ASIA

- CHINA -
EPSON (CHINA) CO., LTD.
28F, Beijing Silver Tower 2# North RD DongSanHuan
ChaoYang District, Beijing, CHINA
Phone: 64106655 Fax: 64107319

SHANGHAI BRANCH
4F, Bldg., 27, No. 69, Gui Jing Road
Caohejing, Shanghai, CHINA
Phone: 21-6485-5552 Fax: 21-6485-0775

- HONG KONG, CHINA -
EPSON HONG KONG LTD.
20/F., Harbour Centre, 25 Harbour Road
Wanchai, HONG KONG
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

- TAIWAN -
EPSON TAIWAN TECHNOLOGY & TRADING LTD.
10F, No. 287, Nanking East Road, Sec. 3
Taipei, TAIWAN
Phone: 02-2717-7360 Fax: 02-2712-9164
Telex: 24444 EPSONTB

HSINCHU OFFICE
13F-3, No. 295, Kuang-Fu Road, Sec. 2
HsinChu 300, TAIWAN
Phone: 03-573-9900 Fax: 03-573-9169

- SINGAPORE -
EPSON SINGAPORE PTE., LTD.
No. 1 Temasek Avenue, #36-00
Millenia Tower, SINGAPORE 039192
Phone: +65-337-7911 Fax: +65-334-2716

- KOREA -
SEIKO EPSON CORPORATION KOREA OFFICE
50F, KLI 63 Bldg., 60 Yoido-dong
Youngdeungpo-Ku, Seoul, 150-763, KOREA
Phone: 02-784-6027 Fax: 02-767-3677

- JAPAN -
SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

Electronic Device Marketing Department
IC Marketing & Engineering Group
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5816 Fax: +81-(0)42-587-5624

ED International Marketing Department Europe & U.S.A.
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5812 Fax: +81-(0)42-587-5564

ED International Marketing Department Asia
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5814 Fax: +81-(0)42-587-5110

In pursuit of **"Saving" Technology**, Epson electronic devices.
Our lineup of semiconductors, liquid crystal displays and quartz devices
assists in creating the products of our customers' dreams.
Epson IS energy savings.

EPSON

SEIKO EPSON CORPORATION

ELECTRONIC DEVICES MARKETING DIVISION

■ EPSON Electronic Devices website

http://www.epson.co.jp/device/

Issue SEPTEMBER 2000, Printed in Japan ⓦ A