USFiles Utility
Users Guide

MAN-USF-FILEUTIL
Version 1.03
February 2002

Copyright & Trademark

© 1998-2001 United States Software Corporation, a Lantronix Company. All rights reserved.

U S Software®, USFiles®, SuperTask!® and TronTask!®, are registered trademarks of United States Software Corporation. MultiTask!™, and EmTalk™ are trademarks of United States Software Corporation. Other brands and names are marked with an asterisk (*) and are the property of their respective owners.

No part of this publication may be reproduced, translated into another language, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of United States Software Corporation.

United States Software Corporation
7175 NW Evergreen Parkway, Suite 100
Hillsboro, OR 97124 USA
Phone: 503-844-6614
Fax: 503-844-6480
On-line: www.ussw.com

Disclaimer & Revisions

United States Software Corporation makes no warranty of any kind with regard to this material, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. United States Software Corporation assumes no responsibility for any errors that may appear in this document. United States Software Corporation makes no commitment to update or to keep current the information contained in this document.

Date	Rev.	Part No.	Comments
09/21/00	1.00	NA	First build
10/03/00	1.01	NA	Modified chkdsk function routine Routine is executed without usin file system.
10/19/00	1.02	NA	Modified the data obtained from **show_partition** function. Modified to parse MBR information directly to the partision information. Modified the data obtained from **show_logical_partition** function. Modified to parse EPBR information directly to the partision information. Modified the way to specify the link executable file in **tp\ makefile**. Modified to use a macro **[$(LNK)]** to specify the file, instead of to specify the link directly.
2/02	1.03	MAN-USF-FILEUTIL	Reformated, minor edits.

Contents

1: Overview

The USFiles Utility (hereinafter referred to as “Utility”) formats disk, creates and deletes partitions, and accesses information.

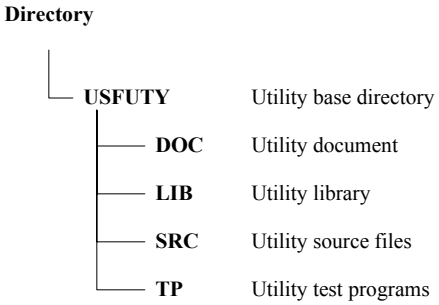
Installation

Specify the directory into which you will copy all files from the media that was provided with this Utility.

Directory Structure

After copying all Utility files, the directory structure will look like this.

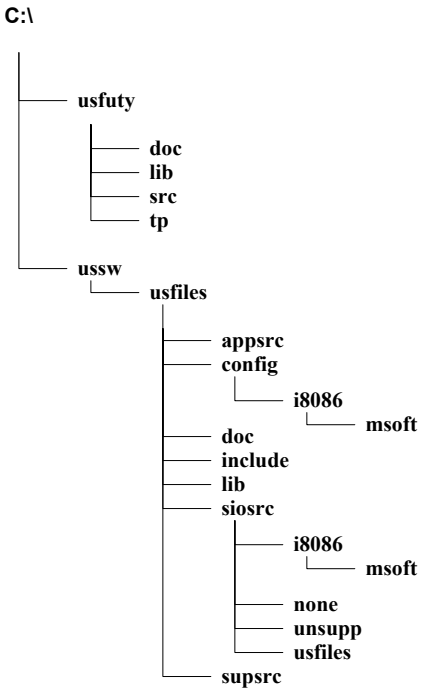
Figure 1-1. Utility Directory Structure, Closeup



- Since the Utility is referenced to the header of USFiles, you must install the Utility software in the same directory in which USFiles is installed.

- The following directory structure assumes that you have specified "C:" as the target directory where the Utility is to be installed.
- It further assumes that USFiles is already installed in "C:\USSW\USFILES.

Figure 1-2. Directory Structure, Zoomed Out



Directory Descriptions

Note: *After compiling the Utility, a library called* ***usf_uty.lib*** *will be created.*

Table 1-1. All Directories

Directory Name	Description
DOC	Contains document files for the Utility
LIB	Library will be stored here after it compiles
SRC	Contains source files of the Utility
TP	Contains sample codes of the test program

Table 1-2. DOC Directory

Document Filename	Description
usf_utility_1_01.pdf	Users guide
verlog.txt	Version information and history

Table 1-3. SRC Directory

Source Filename	Description
fileutls.h	The Utility's header file
utl_ext.h	The Utility's external declaration header file
usf_uty.mak	Sample .**mak** file for creating compile environment
makefile	Sample .**mak** file
chkdsk.c	Sample code for the disk check program
del_lpar.c	Logical partition deletion program
del_par.c	Partition deletion program
format.c	Format program
get_lpar.c	Logical partition data collection program
get_par.c	Partition data collection program
mk_lpar.c	Logical partition creation program
mk_par.c	Partition creation program
usr_func.c	Sample codes for functions to obtain media information and function table
utls_wrk.c	Global work declaration used in the Utility

Table 1-4. TP Directory, Sample Codes

Source Filename	Description
makefile	.**mak** file that creates an executable file for the test program
all_tp.c	Test program that includes Create/Delete partition and Format.
chk_tp.c	Example of disk check function
dl_lp_tp.c	Example of logical partition deleting function
dl_p_tp.c	Example of partition deleting function
fmt_tp.c	Example of formatting function
gt_lp_tp.c	Example of logical partition data collection function
gt_p_tp.c	Example of partition data collection function
mk_lp_tp.c	Example of logical partition creating function
mk_p_tp.c	Example of partition creating function

2: Customize

You need to customize two items, the .**mak** file and the user-defined functions.

MAKE (.mak) file

You can configure compiler settings. Specify the directory for the Utility and USFiles.

- The **MAKE** file that needs customizing is located in **[<USFiles Utility installed directory>\USFUTY\SRC\usf_uty.mak**].
- MS-DOS is used for compiling.
- OPUS MAKE version 6.12 (from OPUS Software) is used to create MAKE executable file.
- Microsoft ® C/C++ Optimizing Compiler Version 8.03 is used for compiling.

Table 2-1. Items Needing Customizing

Default Value	Description
UTY_DIR =c:\ usfuty	Specify the base directory for the Utility. **MAKE** file will always reference to this directory.
CC_DIR =c:\ compiler\ msvc15	Specify directory for C compiler.
CC =$(CC_DIR)\ bin\ cl	Specify C compiler.
LNK =$(CC_DIR)\ bin\ link	Specify linker.
LIBR =$(CC_DIR)\ bin\ lib	Specify librarian.
CC_INC_DIR =$(CC_DIR)\ include	Specify directory for the compiler's **Include** files.
CC_LIB_DIR =$(CC_DIR)\ lib	Specify directory for the compiler's **standard** library.

Default Value	Description
USF_DIR =c:\ ussw\ usfiles\ siosrc	Specify path to **siosrc** directory of USFiles.
USF_LIB_DIR =c:\ ussw\ usfiles\ lib	Specify path to directory for storing library created by USFiles.
USE_RTOS =none	Specify the same value for RTOS that was assigned in **config.mak** file of USFiles.
USE_CPU =i8086	Specify same value for the CPU assigned in **config.mak** file of USFiles.
COMPILER =MSOFT	Specify same value for the compiler assigned in **config.mak** file of USFiles.
USF_INC_DIR =$(USF_DIR); USF_INC_DIR+=$(USF_DIR)\ usfiles; USF_INC_DIR+=$(USF_DIR)\ $(USE_RTOS); USF_INC_DIR+=$(USF_DIR)\ $(USE_CPU); USF_INC_DIR+=$(USF_DIR)\ $(USE_CPU)\ $(COMPILER)	Specify all directories that contain **Include** files for USFiles. This directory structure only applies, to USFiles version 3.01.02. Modify if necessary.
%setenv INCLUDE=	Specify the **Include** directory for the environment variables, since there are too many command lines for Include and it is difficult to compile.
USFLIB =ussw.lib	Specify the name of USFiles library.
DEBUG =1	Select **1**, if a debug code is required during compiling.
CFLAGS	Compile options for a C compiler. Modify as needed for each compiler.
LFLAGS	Linker options. Modify as needed for each linker.

User-defined functions

Since the Utility requires the parameter of the hard disk in which it is installed, you must define functions to collect media information for each device.

Media Information

The following values must be defined for the media information.

- Maximum head number
- Maximum cylinder number
- Number of sectors per track

The method to obtain these values varies, depending on the media and development environment. In the sample, provided with the Utility, these values are obtained with MS-DOS using INT13H.

- Since this is Standard INT13H (not INT13H Extensions), the maximum hard disk size that it supports is about 8 GB.
- You may enter a value in the BIOS menu under HDD information. However, as for the head and cylinder, the values might be indicated in "count". In such a case, the maximum number would be the value indicated on the screen, minus 1. However, if you enter values to set these parameters, you may need to configure again, when the hard disk media is changed.

The values, obtained for the media, then needs to be defined in the following structure in the global area:
MEDIAINFO MediaInfo;/*Structure containing media information*/

```
struct _MEDIAINFO {
unsigned short cylinder;/*maximum cylinder number*/
unsigned short head;/*maximum head number*/
unsigned short sect_p_track;/*sectors per track count*/
}  MEDIAINFO;
```

The total number of sectors can be obtained by [maximum cylinder number + 1] x [maximum head number + 1] x [sectors per track count].

Create the Media Information Function

Create the Media Information Retrieval function using the following format.

Format
#include "pcfmhdr.h ”
#include “fileutls.h ”
#include "utl_ext.h"
short UserMakeFunc(unsigned short unit_no)

Argument
unsigned short unit_no :device number (=unit_no in device_tab)

Returned value
Positive value: normal
Negative value: Error code (ERR_MEDIAINFO)

If you have more than one device, you must create the function for each device.
You can assign any name for the function.

The following is a sample function to obtain media information via MS-DOS, using INT13H.

```
#include <dos.h>

#include "pcfmhdr.h"
#include "fileutls.h"
#include "utl_ext.h"

short BiosDrvInfo(char unit_no)
{
    union REGS inregs,outregs;
    short ret;
```

```
    ret =SUCCESS;
    inregs.h.dl =unit_no;/*Set the device number to DL register
    */
    inregs.h.ah =BIOS_GET_INFO;/*Set function number (8) to
    AH */
    int86(0x13,&inregs,&outregs);/*Use INT13H */
    if(outregs.x.cflag)/*If CF is not equal to 0, then Error */
    {

    ret =ERR_MEDIAINFO;/*Set the returned value for Error */
    }

    /*If successful, then set the obtained data to Global work */
    /*The last cylinder number */
    MediaInfo.cylinder =(outregs.h.ch |((outregs.h.cl
    &0xc0)<<2));

    /*The last head number */
    MediaInfo.head =outregs.h.dh;

    /*Sectors per track count */
    MediaInfo.sect_p_track =outregs.h.cl &0x3f;

    return(ret);

}
```

Registering to Function Table

When the Utility calls the media information retrieval function, it calls from a function table. Therefore, you must first register the function to the function table.

You must register functions for all drives that are registered in **device_tab** of USFiles application.

- You must register the functions in the same order, as the drives, which are registered in **device_tab**.

- When the function is called, the index value for **device_tab** in USFiles will be used as the index value for **func_tbl**.
- If a removable drive, such as a floppy drive, is not being used, set the value to NULL.
- You must register for all devices that are registered in **device_tab** of USFiles.

Format

```
typedef struct func_member
{
short (*MediaInfoGet)(char );
}FUNC_MEMBER;

FUNC_MEMBER func_tbl[ ] ={
    Media information retrieval function for row 0 in device_tab,
    Media information retrieval function for row 1 in device_tab,
    Media information retrieval function for row 2 in device_tab,
     •
    Media information retrieval function for row n in device_tab,
    };
```

The following example shows how to register BiosDrvInfo function to funt_tbl. Since all devices use same manner to obtain the information, one function is used for all devices.

```
FUNC_MEMBER func_tbl[ ] ={
NULL ,          /* For device_tab[ 0] *FD not used      */
BiosDrvInfo ,   /*      [ 1] D drive                    */
BiosDrvInfo ,   /*      [ 2] E                          */
BiosDrvInfo ,   /*      [ 3] F                          */
BiosDrvInfo ,   /*      [ 4] G                          */
BiosDrvInfo ,   /*      [ 5] H                          */
BiosDrvInfo ,   /*      [ 6] I                          */
BiosDrvInfo ,   /*      [ 7] J                          */
BiosDrvInfo ,   /*      [ 8] K                          */
BiosDrvInfo ,   /*      [ 9] L                          */
};
```

3: Function Interface

This Chapter provides detailed descriptions of each function.

Format

Description

This function formats the specified drive with FAT type, indicated in the partition information.

Function call format

```
#include "pcfmhdr.h ”
#include “fileutls.h ”
int format(char *drivename,char *options );
```

Argument

char *drivename	name of drive to be formatted
char *options	option flag string Use default setting for NULL pointer.

Table 3-1. Option Flag String Descriptions

String	Description
/V<name>	Specify Volume name, up to 11 characters long Default setting: Do not set NO NAME for BPB setting
/C	Search for bad cluster during formatting and set bad cluster mark in FAT. Search for bad cluster is initiated when an error occurred during Read/Write to a sector. Default setting: Do not search for bad cluster
/S	Clear all sectors with 0 Default setting: Do not clear sector

String	Description
/Q	Run Quick Format. Create FAT and root directory entry only. Default setting: Create BPB.
/Z64	Set the number of sectors per allocation unit to 64. Number can be set if the drive capacity is more than 512 MB. One allocation unit is 32kb. Default setting: as described below.

Returned Value:
Positive value: Normal
Negative value: Error code (refer to "Returned Value Index")

> **Note:** *Partition information is used to distinguish between FAT32 FAT16 and FAT12. The number of FATs to be created will be 2 (fixed). The number of root directory entries to be created for FAT 12/16 will be 512 (fixed).*

*For FAT32, one cluster is reserved to create an area for root directory entry. The first cluster of root directory entry is cluster number 2 (fixed).

Searching for the partition table is based on USFiles format. For Quick Format, the existing BPB setting values is used.

Table 3-2. Error Conditions

Order	Description
1	Sector size is not 512 bytes
2	Medium descriptor is not 0xF8
3	Heads and sector per track information obtained from media information is different

If Quick Format is selected in Options, an error will occur, if "/S", "/V", or "/Z64" is specified.

Table 3-3. Default Formatting Allocation Unit Size

Disk Capacity	Sectors	Clusters	FAT Bits[1]	Remarks
0~15 MB	8	4K	12	FAT12
16~127 MB	4	2K	16	FAT16
128~255 MB	8	4K	16	FAT16
256~511 MB	16	8K	16	FAT16
512~1023 MB	32	16K	16	FAT16
1024~2048 MB	64	32K	16	FAT16
512~1023 MB	8	4K	18	FAT32
1024 MB~2047 MB	8	4K	19	FAT32
2048 MB~4095 MB	8	4K	20	FAT32
4096 MB~8191 MB	8	4K	21	FAT32
8192 MB~16384 MB	16	8K	21	FAT32
16384 MB~32767 MB	32	16K	21	FAT32
32 GB or more	64	32K	26	FAT32
1 TB~2 TB	64	32K	26	FAT32

Creating Partition

Description

This function creates a partition in Master Boot Record of the specified device.

Call Format

```
#include "pcfmhdr.h ”
#include “fileutls.h ”
int make_partition (
unsigned short      unit_no,
unsigned short      partition_no,
unsigned long       start_lsect,
unsigned long       lsect_size,
char                *options );
```

Argument

unsigned short unit_no :Device number (=Unit No.

[1] FAT bits are effective bits used by FAT

on device tab)
unsigned short partition_no :Partition number (0~3)
unsigned long start_lsect, :Logical sector start number
unsigned long lsect_size, :Sector size
char *options :Option flag string
Use default setting for NULL pointer

Table 3-4. Option Flag String Descriptions

String	Description
/E	Creates an extended partition. Default setting: Create DOS FAT partition.
/X	Creates FAT16 partition, if the disk capacity is between 512 MB and 2048 MB. Default setting: FAT 32, if more than 512 MB. * If other than the effective capacity is specified, the default setting is applied.

Returned Value
Positive value: Normal
Negative value: Error code (refer to "Returned Value Index")

Note: *If the partition is already in use, an error will occur. System type settings in partition information for Master Boot Record are set with following condition. Following are the partitions that can be created.*

Table 3-5. Partitions

Order	Description
1	DOS FAT partition only
2	Extended partition only
3	DOS FAT partition, followed by Extended partition

* Unused partition (partition type: 0x00) will be ignored.
* Unknown partition type is considered as DOS FAT partition (except for the partition type described in next section).

If the partition is already in use, an error will occur.

Note that when a partition is created, the drive structure in Device Table may change.

When creating an extended partition, starting logical sector number must be above 16065 (approximately 8 MB), and sector size to be above 16002 (approximately 8 MB). FDISK may not run properly, when the starting logical sector number of an extended partition is below 16065. FDISK can recognize an extended space smaller than 8 MB. However, the minimum sector size is set at 16002, since creating logical drive may corrupt the subsequent space, due to 8 MB minimum size limitation.

When creating DOS FAT partition, starting logical sector number must be above 1, and sector size to be above 2048 (1 MB).

Table 3-6. FAT System Type Setting Requirements

Disk Capacity	FAT Bits	Remarks
0~15 MB	12	FAT12
16~127 MB	16	FAT16
128~255 MB	16	FAT16
256~511 MB	16	FAT16
512~1023 MB	16	FAT16
1024~2048 MB	16	FAT16
512 MB~1023 MB	18	FAT32
1024~2047 MB	19	FAT32
2048~4095 MB	20	FAT32
4096~8191 MB	21	FAT32
8192~16384 MB	21	FAT32
16384~32767 MB	21	FAT32
> 32 GB	26	FAT32
1~2 TB	26	FAT32

Table 3-7. Partition Type Descriptions

Partition	Description
00H	Unused partition

Partition	Description
01H	DOS FAT12
04H	DOS FAT16
05H	Extended (Extended partition)
06H	FAT16B (16 bit FAT+ >32 MB)
0BH	FAT32
0CH	FAT32X
0EH	FAT16X
0FH	Extended X

*FAT16X, FAT32X, and ExtendedX are partition types to enable FAT partitions to exist beyond 1024 cylinders

Deleting Partitions

Description
This function deletes all the information on a specific partition in Master Boot record of specified device.

Call format
#include "pcfmhdr.h ”
#include “fileutls.h ”
int purge_partition (unsigned short unit_no,unsigned short partition_no);

Argument

unsigned short unit_no	:device number (=Unit No. in device_tab)
unsigned short partition_no	:Partition number to be deleted (0~3)

Return Value
Positive value: Normal
Negative value: Error code (refer to “Returned Value Index”)

Note: *The function sets the value [0x00] in a corresponding partition table. When deleting an extended partition, all partition, including logical*

partitions, will be deleted. When deleting a partition, the starting sector of corresponding partition is cleared with [0x00]. When a partition is created, the drive structure in Device Table may change.

Retrieving Partition Information

Description
This function retrieves specific partition information from master Boot Record of a specified device.

Call Format
#include "pcfmhdr.h ”
#include “fileutls.h ”
int show_partition (unsigned short unit_no,
unsigned short partition_no,
PARTITION *partition_info);

Argument
unsigned short unit_no :device number (unit_no in device_tab)
unsigned short partition_no :partition number (0~3)
PARTITION *partition_info :pointer to the partition information storage

* PARTITION type is defined in mtio.h in USFILE.
typedef struct partition{
byte boot_flag;
byte start_head;
byte start_sect;
byte start_cyl;
byte system_id;
byte ending_head;
byte ending_sect;
byte ending_cyl;
unsigned long partition_start_sect,

```
    partition_size;
}PARTITION;
```

Returned Value
Positive value: Normal
Negative value: Error code (refer to "Returned Value Index")

> **Note:** *Information retrieve in the above manner is the same information in MBR.*

Creating a Logical Partition

Description
This function creates a logical partition within an Extended DOS partition for the specified device.

If there is already a logical partition created, a new logical partition is created at the end of the first logical partition.

Call Format

```
#include "pcfmhdr.h "
#include "fileutls.h "
unsigned long make_logical_partition (
unsigned short unit_no,
unsigned long lsect_size,
char *options );
```

Arguments

unsigned short unit_no	:device number (unit_no in device_tab)
unsigned long lsect_size	:Sector size
char *options	:Option flag string Use default setting for NULL pointer

Table 3-8. Option Flag String Descriptions

String	Description
/X	Creates FAT16 partition, if the disk capacity is between 512 MB and 2048 MB. Default setting: FAT 32, if more than 512 MB. If other than the effective capacity is specified, the default setting is applied.

Returned Value
Positive value: Normal
Negative value: Error code (refer to "Returned Value Index")

> **Note:** *Logical number of sector per track is set as a relative sector number. Note that when a logical partition is created, the drive structure in Device Table may change. Make sure that the sector size is above 16002 (approximately 8 MB). You will need free sector, derived by the formula [(sector size) + (number of sectors per track)].*

Retrieving Logical Partition Information

Description
This function retrieves the specified logical partition information (EPBR) in an extended DOS partition.

Call Format
```
#include "pcfmhdr.h "
#include "fileutls.h "
int show_logical_partition (
unsigned short unit_no,
unsigned short partition_no,
PARTITION *partition_info,
unsigned long *free_sectors );
```

Arguments
unsigned short unit_no :device number (=unit_no in device_tab)

unsigned short partition_no	:specified logical partition number (0~n)
PARTITION *partition_info	:logical partition information
unsigned long *free_sectors	:number of free sectors in Extended DOS partition

Returned Value
Positive value: Number of logical partitions
Negative value: Error code (refer to “Returned Value Index”)

> **Note:** *Information retrieve in above manner is the same information in EPBR.*

Deleting the Logical Partition

Description
This function deletes only the logical partition (EPBR) at the end of Extended DOS partition of a specified device.

Call Format
#include "pcfmhdr.h ”
#include “fileutls.h ”
unsigned long purge_logical_partition (
unsigned short unit_no);

Arguments

unsigned short unit_no	:device number (=unit_no in device_tab)

Returned Value
Positive value: Normal
Negative value: Error code (refer to “Returned Value Index”)

> **Note:** *After deleting a partition, partition type is set with 0x00. The starting sector is cleared with 0x00. Note that when a logical partition is deleted, the drive structure in Device Table may change.*

Checking Disk (Sample Code)

Description
This function checks the FAT consistency among FAT12, FAT16, and FAT32.

Call Format
#include "pcfmhdr.h ”
#include “fileutls.h ”
int chkdsk (char *drivename);

Arguments
char *drivename :drive to be checked

Returned Value
Positive value: Normal
Negative value: Error code (refer to “Returned Value Index”)

> **Note:** *FAT consistency check is done, only when there are more than two FATs. If two FAT contents are not consistent, an error will occur.*

Common Items

The Utility uses “Driver Function Table” in USFiles’s “device_tab” to perform Read and Write to a specific device or a drive.

4: Returned Value Index

The Utility returns the following values.

Table 4-1. Returned Values

Code	Value	Description
SUCCESS	0	Normal termination
ERR_SYS_LOCK	-1	System lock failed
ERR_NO_MEM	-2	Failed to obtain memory
ERR_NO_DRIVE	-3	Can't find specified drive name
ERR_NO_HDD	-4	Device is not HDD
ERR_CAPABILITY	-5	Hardware does not have sufficient capacity
ERR_USE_FILESYSTEM	-6	File system is in use
ERR_OPTIONS	-7	Invalid option string
ERR_DISKINIT	-8	Failed to initialize device
ERR_MEDIAINFO	-9	Failed to retrieve media data
ERR_SEC_READ	-10	Failed to read a sector
ERR_MBR_ABNORMAL	-11	Invalid MBR contents
ERR_PART_NOTFOUND	-12	Specified partition not found
ERR_NOMUCH_FAT_TYPE	-13	Invalid FAT type for partition
ERR_BPB_ABNORMAL	-14	Invalid BPB contents
ERR_SEC_WRITE	-15	Writing to sector failed
ERR_FAT_READ	-16	Reading from FAT failed
ERR_FAT_WRITE	-17	Writing to FAT failed
ERR_DIR_WRITE	-18	Root directory writing failure
ERR_NO_UNIT	-19	Unit number does not exist
ERR_PART_NUM	-20	Partition number is invalid
ERR_SEC_SIZE	-21	Abnormal sector size
ERR_EXTPART_EXIST	-22	Extended partition already exists
ERR_PRIPART_EXIST	-23	Primary partition already exists

Code	Value	Description
ERR_UNSUPP_PART	-24	Unsupported partition structure
ERR_USE_PART	-25	Partition is already in use
ERR_NOUSE_PART	-26	Unused partition
ERR_START_SEC	-27	Incorrect starting sector
ERR_EPBR_ABNORMAL	-28	Invalid EPBR
ERR_CLUST_SIZE	-29	Abnormal cluster size
ERR_NO_USER_FUNC	-30	User-defined function is not registered
ERR_NO_DEVICE	-31	Specified device not found
ERR_UNDER_FAT_NUM	-32	Insufficient FAT number
ERR_FAT_UNMUCH	-33	Mismatched FAT contents between FAT1 and FAT2
ERR_FILE_ABNORMAL	-34	Abnormal file was found
ERR_FAT_ABNORMAL	-35	Abnormal FAT chain
ERR_FAT_FLOAT	-36	Float exists in FAT

5: Test Programs

This section describes various test programs, which are included in USFiles Utility.

System Requirements

The test programs are designed to run under the following requirements.

- MS-DOS command prompt in Windows98, or MS-DOS launched from Windows98 startup disk.
- Use the same compiler that was used to create the library files.
- OPUS MAKE version 6.12 (from OPUS Software) is to be used to create MAKE executable file.
- Use values in the table below for device table settings.
- Test results are displayed on a screen using "printf".

Table 5-1. Device Driver Settings

Name	Description
Device number	0x81
Partition	0 ~ 8 (0: primary partition area, Others: extended partition area)
Driver	biosdrv_s （biosdrv.c is used
Drive name	[D] ~ [L]

```
extern struct driver_p const biosdrv_s;
#define HARDDRV biosdrv_s
```

```
/*  [D] drive                   */
     FM_PCFM,0xf,0,0,
     0x81,/*unit number,(128 =first hard drive,129 =second,..*/
     0,/*partition number */
     0,NULL,0,(DRIVER *)&HARDDRV,&pcfm,"D",
     &pcparmD,NULL
,
```

Drives [E] to [J] omitted.

```
/*  [K] drive           */
     FM_PCFM,0xf,0,0,
          0x81,/*unit number,(128 =first hard drive,129
          =second,..*/
          7,/*partition number */
          0,NULL,0,(DRIVER *)&HARDDRV,&pcfm,"K",
          &pcparmK,NULL
,

/*  [L] drive           */
     FM_PCFM,0xf,0,0,
          0x81,/*unit number,(128 =first hard drive,129
          =second,..*/
          8,/*partition number */
          0,NULL,0,(DRIVER *)&HARDDRV,&pcfm,"L",
          &pcparmL,NULL
```

Program Descriptions

This section describes each test program, included in USFiles Utility.

Each test program executes normal mode scripts, followed by abnormal mode (invalid parameter, etc.), and displays the test results on a screen using “printf”.

Table 5-2. Test Program Descriptions

Filename	Description
fmt_tp.c	Format the specified drive. **Results** normal format(D,NULL)ret:0 format(E,/C)ret:0 format(F,/S)ret:0 format(G,/Vgdrive)ret:0 format(H,/C/S)ret:0 format(I,/S/Vidrive)ret:0 format(J,/Z64)ret:0 format(K,/C/S/Vkdrive)ret:0 abnormal format(L,/Q)ret:-13 format(D,/Z32)ret:-28 format(D,////SSS)ret:-6 format(D,/Q/S)ret:-6 format(D,/Q/Vddrive)ret:-6 format(D,/Q/Z64)ret:-6 format(ZZ,NULL)ret:-2 format(A,NULL)ret:-3
mk_p_tp.c	Create a partition in a specific region of the specified device. **Results** normal make_partition(0x81,0,0x3f,0x3e82,NULL)ret:0 make_partition(0x81,1,0x3ec1,0xdae73d,/E)ret:0 abnormal make_partition(0x81,0,0x3f,0x3e82,NULL)ret:-21 make_partition(0x81,1,0x3ec1,0xdae73d,/E)ret:-21 make_partition(0x81,3,0x3f,0x3e82,NULL)ret:-21 make_partition(0x81,9,0x3f,0x3e82,NULL)ret:-20 make_partition(0x81,3,0x3f,0x3e82,/T)ret:-7 make_partition(0x81,3,0x3f,0x3,NULL)ret:-21 make_partition(0x89,0,0x3f,0x3e82,NULL)ret:-19 make_partition(0x01,0,0x3f,0x3e82,NULL)ret:-19
gt_p_tp.c	Retrieve information of specified partition of the device. **Results** normal show_partition(0x81,0,&par[0])ret:0 boot_flag:80 start_head:1 start_sect:1 start_cyl:0

Filename	Description
	system_id:1 ending_head:fe ending_sect:3f ending_cyl:0 partition_start_sect:3f partition_size:3e82 show_partition(0x81,1,&par[1])ret:0 boot_flag:0 start_head:0 start_sect:1 start_cyl:1 system_id:5 ending_head:fe ending_sect:3f ending_cyl:7d partition_start_sect:3ec1 partition_size:dae73d show_partition(0x81,2,&par[2])ret:0 boot_flag:0 start_head:0 start_sect:0 start_cyl:0 system_id:0 ending_head:0 ending_sect:0 ending_cyl:0 partition_start_sect:0 partition_size:0 show_partition(0x81,3,&par[3])ret:0 boot_flag:0 start_head:0 start_sect:0 start_cyl:0 system_id:0 ending_head:0 ending_sect:0 ending_cyl:0 partition_start_sect:0 partition_size:0 abnormal show_partition(0x81,4,&par[4])ret:-20 show_partition(0x89,1,&par[4])ret:-19 show_partition(0x01,1,&par[4])ret:-19
dl_p_tp	Delete a specified partition of the device. **Results** normal purge_partition(0x81,0)ret:0 purge_partition(0x81,1)ret:0 abnormal purge_partition(0x81,2)ret:-26 purge_partition(0x81,3)ret:-26 purge_partition(0x81,4)ret:-20 purge_partition(0x89,1)ret:-19 purge_partition(0x01,0)ret:-19
mk_lp_tp.c	Create a logical drive in the extended partition of the specified device. **Results** normal make_logical_partition(0x81,0x3e82,NULL)ret:0 make_logical_partition(0x81,0xbc04,NULL)ret:0 make_logical_partition(0x81,0x42a92,NULL)ret:0 make_logical_partition(0x81,0x816a2,NULL)ret:0

Filename	Description
	make_logical_partition(0x81,0x102d83,/X)ret:0 make_logical_partition(0x81,0x201c84,/X)ret:0 make_logical_partition(0x81,0x403947,NULL)ret:0 make_logical_partition(0x81,0x5d263d,NULL)ret:0 abnormal make_logical_partition(0x81,0x3e82,NULL)ret:-21 make_logical_partition(0x89,0x3e82,NULL)ret:-19 make_logical_partition(0x81,0x3e82,/T)ret:-7 make_logical_partition(0x81,0x3,NULL)ret:-21 make_logical_partition(0x01,0x3e82,NULL)ret:-19
gt_lp_tp.c	Retrieve logical partition information from the extended partition of specified device. **Results** normal show_logical_partition(0x81,0,&par[0] ,&rem_size)ret:8 boot_flag:0 start_head:1 start_sect:1 start_cyl:1 system_id:1 ending_head:fe ending_sect:3f ending_cyl:1 partition_start_sect:3f partition_size:3e82 rem_size:1000 show_logical_partition(0x81,1,&par[1] ,&rem_size)ret:8 boot_flag:0 start_head:1 start_sect:1 start_cyl:2 system_id:4 ending_head:fe ending_sect:3f ending_cyl:4 partition_start_sect:3f partition_size:bc04 rem_size:1000 show_logical_partition(0x81,2,&par[2] ,&rem_size)ret:8 boot_flag:0 start_head:1 start_sect:1 start_cyl:5 system_id:6 ending_head:fe ending_sect:3f ending_cyl:15 partition_start_sect:3f partition_size:42a92 rem_size:1000 show_logical_partition(0x81,3,&par[3] ,&rem_size)ret:8 boot_flag:0 start_head:1 start_sect:1 start_cyl:16 system_id:6 ending_head:bd ending_sect:3e ending_cyl:36 partition_start_sect:3f partition_size:806a2 rem_size:1000 show_logical_partition(0x81,4,&par[4] ,&rem_size)ret:8 boot_flag:0 start_head:be start_sect:3f start_cyl:36 system_id:6 ending_head:bd ending_sect:3e ending_cyl:78

Filename	Description
	partition_start_sect:3f partition_size:102d83 rem_size:1000 show_logical_partition(0x81,5,&par[5] ,&rem_size)ret:8 boot_flag:0 start_head:be start_sect:3f start_cyl:78 system_id:6 ending_head:bd ending_sect:3e ending_cyl:fb partition_start_sect:3f partition_size:201c84 rem_size:1000 show_logical_partition(0x81,6,&par[6] ,&rem_size)ret:8 boot_flag:0 start_head:be start_sect:3f start_cyl:fb system_id:b ending_head:bd ending_sect:3e ending_cyl:1 partition_start_sect:3f partition_size:403947 rem_size:1000 show_logical_partition(0x81,7,&par[7] ,&rem_size)ret:8 boot_flag:0 start_head:be start_sect:3f start_cyl:1 system_id:b ending_head:bd ending_sect:3e ending_cyl:7d partition_start_sect:3f partition_size:5d263d rem_size:1000 abnormal show_logical_partition(0x81,8,&par[0] ,&rem_size)ret:-12 show_logical_partition(0x89,0,&par[0] ,&rem_size)ret:-19 show_logical_partition(0x01,0,&par[0] ,&rem_size)ret:-19
dl_lp_tp.c	Delete specified logical drive in the extended partition of specified device. **Results** normal purge_logical_partition(0x81)ret:0 purge_logical_partition(0x81)ret:0 purge_logical_partition(0x81)ret:0 purge_logical_partition(0x81)ret:0 purge_logical_partition(0x81)ret:0 purge_logical_partition(0x81)ret:0 purge_logical_partition(0x81)ret:0 purge_logical_partition(0x81)ret:0 abnormal purge_logical_partition(0x81)ret:-12 purge_logical_partition(0x89)ret:-19 purge_logical_partition(0x01)ret:-19

Filename	Description
all_tp.c	Create partitions, delete partitions, retrieve information and format the specified drive. **Results** normal No.1 purge_logical_partition(0x81)ret:-12 purge_partition(0x81,0)ret:0 purge_partition(0x81,1)ret:-26 purge_partition(0x81,2)ret:-26 purge_partition(0x81,3)ret:-26 make_partition(0x81,3,0x3ec1,15466689,NULL)ret:0 show_partition(0x81,0,&par[0])ret:0 boot_flag:0 start_head:0 start_sect:1 start_cyl:1 system_id:5 ending_head:c0 ending_sect:3f ending_cyl:c3 partition_start_sect:3ec1 partition_size:ec00c1 make_logical_partition(0x81,15466626,NULL)ret:0 show_logical_partition(0x81,0,&par[0] ,&rem_size)ret:1 boot_flag:0 start_head:1 start_sect:1 start_cyl:1 system_id:b ending_head:c0 ending_sect:3f ending_cyl:c3 partition_start_sect:3f partition_size:ec0082 rem_size:0 make_partition(0x81,0,0x3f,0x3e82,NULL)ret:0 show_partition(0x81,0,&par[0])ret:0 boot_flag:80 start_head:1 start_sect:1 start_cyl:0 system_id:1 ending_head:fe ending_sect:3f ending_cyl:0 partition_start_sect:3f partition_size:3e82 format(D,/S)ret:0 purge_partition(0x81,0)ret:0 purge_partition(0x81,3)ret:0 show_logical_partition(0x81,0,&par[0] ,&rem_size)ret:-12 boot_flag:0 start_head:0 start_sect:0 start_cyl:0 system_id:0 ending_head:0 ending_sect:0 ending_cyl:0 partition_start_sect:0 partition_size:0 rem_size:0 make_partition(0x81,0,0x3f,15466689,NULL)ret:0 show_partition(0x81,0,&par[0])ret:0

Filename	Description
	boot_flag:80 start_head:1 start_sect:1 start_cyl:0 system_id:b ending_head:c1 ending_sect:3f ending_cyl:c2 partition_start_sect:3f partition_size:ec00c1 format(D,/S)ret:0
chk_tp.c	Check FAT chain of the specified drive. **Results** normal chkdsk c:ret:0 chkdsk e:ret:0 chkdsk f:ret:0 chkdsk g:ret:0 chkdsk h:ret:0 chkdsk i:ret:0 chkdsk j:ret:0 chkdsk k:ret:0 abnormal chkdsk A:ret:-4 chkdsk Z:ret:-3 chkdsk m:ret:-3

Compile methods

From Command line, enter “make [test program name]”.

Specify the test program name without “.c” extension.

Example: “Format” test program

From command line, enter following commands.

c:\ usfuty\ tp>make fmt_tp

del fmt_tp.exe
del fmt_tp.obj
c:\ compiler\ msvc15\ bin\ cl /Od /c /AL /Gs /nologo /YX /DFARDEF=far /D_BIOS_IBMPC

```
/I.;c:\ ussw\ usfiles\ siosrc;c:\ ussw\ usfiles\ siosrc\ usfiles;c:\
ussw\ usfiles\ siosr
c\ none;c:\ ussw\ usfiles\ siosrc\ i8086;c:\ ussw\ usfiles\ siosrc\
i8086\ MSOFT /Zi fmt_t
p.c
fmt_tp.c
c:\ compiler\ msvc15\ bin\ link /M /LI /NOD /NOE /NOL /CO
fmt_tp.obj ..\ src\ utls_wr
k.obj,fmt_tp.exe,fmt_tp.map,c:\ compiler\ msvc15\ lib\ llibce c:\
compiler\ msvc15\
lib\ oldnames c:\ ussw\ usfiles\ lib\ ussw.lib c:\ usfuty\ lib\
usf_uty.lib,nul
```

After these lines are displayed, an executable file called "fmt_tp.exe" will be created.