



Nucleus® Storage Guide

Release 2013.08

August 2013

© 1993-2013 Mentor Graphics Corporation
All rights reserved.

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS 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.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: www.mentor.com/trademarks.

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form

Table of Contents

Chapter 1

Nucleus Storage Components	7
Build Configurations	7
Required Include File	7
Storage Components	8

Chapter 2

Nucleus FILE System	9
Nucleus FILE System Overview	9
Nucleus FILE System Concepts and Definitions	9
Nucleus File System Operation	10
Tuning and Optimization of Nucleus Storage	11
Initialization	11
Addition of Storage Devices	12
Storage Device Mount/Unmount	12
Common File and Directory Operations	13
Search Operations	14
Nucleus FILE System Internals	14
Nucleus Virtual File System Services	14
Nucleus FAT File System	15
Nucleus SAFE File System	16
Nucleus FILE System Data Structures	17
CPTE_S	17
DSTAT	18
FS_S	19
Nucleus FILE System APIs	21
API Notes	23
NU_Become_File_User (Deprecated)	25
NU_Check_Disk	26
NU_Check_File_User (Deprecated)	28
NU_Close	29
NU_Close_Disk (Deprecated)	31
NU_Create_Partition	32
NU_Current_Dir (Deprecated)	34
NU_Delete	35
NU_Delete_Partition	37
NU_Disk_Abort	38
NU_Done	40
NU_FILE_Cache_Create	42
NU_FILE_Cache_Destroy	44
NU_FILE_Cache_Flush	46
NU_FILE_Cache_Get_Config	48

NU_FILE_Cache_Set_Config	50
NU_Flush	52
NU_Format	54
NU_Free_List	57
NU_Free_Partition_List	58
NU_FreeSpace	59
NU_Get_Attributes	61
NU_Get_Default_Drive (Deprecated)	63
NU_Get_First	64
NU_Get_Format_Info	67
NU_Get_Next	69
NU_Get_Partition_Info	70
NU_List_Device	72
NU_List_File_System	74
NU_List_Mount	76
NU_List_Partitions	78
NU_Make_Dir	80
NU_Mount_File_System	82
NU_Open	84
NU_Open_Disk (Deprecated)	89
NU_Read	91
NU_Register_File_System	93
NU_Release_File_User (Deprecated)	95
NU_Remove_Dir	96
NU_Rename	98
NU_Seek	100
NU_Set_Attributes	102
NU_Set_Current_Dir (Deprecated)	104
NU_Set_Default_Drive (Deprecated)	106
NU_Storage_Device_Wait	107
NU_Truncate	108
NU_Unmount_File_System	110
NU_Unregister_File_System	111
NU_Utime	112
NU_Write	114
FILE APIs Available to a Nucleus Process	115
Nucleus FILE System Shell Commands	117
dir	118
del	119
mkdir	120
rmdir	121
copy	122
Chapter 3	
Nucleus SQLite	125
Nucleus SQLite Features	125
SQLite Design	125
Initializing Nucleus SQLite	126

Table of Contents

Introduction to the SQLite C Interface	127
Function Reference	127
Data Structures Reference	127

Embedded Software and Hardware License Agreement


List of Tables

Table 1-1. Nucleus Storage Components	8
Table 2-1. NU_List_Device Output Descriptions	10
Table 2-2. Registry Options for Mount Information	12
Table 2-3. NU_Format Parameters for the FAT file system	55

Chapter 1

Nucleus Storage Components

Nucleus FILE storage system consists of the following components: VFS (Virtual File System), FAT 16/32 bit (File Allocation Table) and SAFE. These components of the storage package are located at `\os\storage`. Depending on your embedded application, any of these components can be configured for use.

 **Note** Your source code is initially located in the `<install_root>\nucleus` directory. The source from this directory is copied into the project folder you specify.

Additionally, SQLite has been ported to provide a lightweight, compact database application. Nucleus SQLite is a server-independent library and does not require an independent server. It requires zero configuration; it does not require installation, configuration, or administrative management. See the [Nucleus SQLite](#) chapter for additional information on the implementation.

For more information about packages and components, see “Nucleus ReadyStart Configuration” in the *Nucleus ReadyStart Guide* or “Nucleus Source Code Configuration” in the *Nucleus Source Code Guide*.

Build Configurations

All components, including SQLite, are, by default, enabled through the `.metadata` file located at the top level of each component’s installation, for example for the VFS: `\os\storage\file\VFS`. When a component is enabled, it is included in the build. You can create a user configuration file to override the default configuration and exclude a component from the build.

The `.metadata` file for each storage component lists the default required components to be enabled for most middleware components and the specific components required to be enabled for SAFE, FILE and/or VFS components.

For more information, see “Creating a Custom Configuration” in the *Nucleus ReadyStart Guide* or in the *Nucleus Source Code Guide*.


Required Include File

To make all the FILE Storage Package Components APIs visible to an application, include the file `storage\nu_storage.h` in your application using the following statement:

```
#include "storage/nu_storage.h"
```

SQLite does not require additional include files.

Warning

 In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus guides. There is no guarantee of future support or compatibility for any interface that is not documented.


Storage Components

[Table 1-1](#) summarizes each Nucleus storage component and links to additional detailed usage information in this document.

Table 1-1. Nucleus Storage Components

Storage Component	Description
Nucleus FILE System	APIs to operate on mass storage devices using FAT and SAFE file systems.
Nucleus SQLite	Self-contained, server-less, zero-configuration, transactional SQL database engine

Note

 Nucleus POSIX File System is documented in the Nucleus POSIX Guide.

Chapter 2

Nucleus FILE System

This chapter describes the Nucleus FILE system, including how to configure your storage devices and manage your storage.

Nucleus FILE System Overview

The Nucleus FILE System allows for quick and easy implementation of file I/O in an embedded application. The two file system components are optimized for embedded applications, providing the following functionality:

- A Windows®-compatible VFAT file system
- A fault tolerant SAFE file system

Features of the Nucleus Storage components include:

- A Virtual File System interface that is highly extensible and suitable for including multiple file systems under a common programming interface. Both the FAT and SAFE file system components are accessed through this interface.
- Support for FAT12, FAT16, and FAT32 formats on any number of drives with filenames up to 255 characters.
- A fault tolerant SAFE File System ideal for safety-critical applications.
- Highly-configurable source code with many tuning and optimization options. Suitable for projects of any scale: from the most robust to the most constrained systems.
- A file system that can be used out of the box. A RAM disk driver is included with Nucleus OS.
- Support for multiple-language encoding in a single application using the UTF-8 standard.

Nucleus FILE System Concepts and Definitions

The following concepts and definitions provide background information for understanding the workings of the Nucleus FILE System.

Fully qualified paths

The [Nucleus FILE System APIs](#) that use path as a parameter should be called using a fully qualified path. An example of a fully qualified path is:

"C:\foo\bar.txt"

Logical and Physical devices

Physical devices can contain one or more logical devices. Logical devices can be formatted with a file system and subsequently mounted. It is possible to obtain information about all devices registered in the system using the [NU_List_Device](#) API function.

The list generated from the `NU_List_Device` function will contain information about each device. The length of the name will indicate if the device is physical or logical.

Table 2-1. NU_List_Device Output Descriptions

Format of name	Type	Example
XXX	Physical	SDA
XXX##	Logical	SDA00

Device physical names are created by the drivers. Logical names are created by enumerating logical devices found and appending to the physical name.

Mount Points

Mount points are simply drive letter designators ("A:\", "C:\") that associate a logical device with a file system type. Mount points are assigned automatically by the system.

Note



Currently, Nucleus Storage does not support the ability to set a drive letter to a specific mount; however, drive letters can be requested in the driver's device section of a platform file. The ability to set the drive letter is a feature that will be added in a future release.

Nucleus File System Operation

This section provides information on to perform basic file system operations using the Nucleus FILE System. Topics covered include:

- [Tuning and Optimization of Nucleus Storage](#)
- [Initialization](#)
- [Addition of Storage Devices](#)
- [Storage Device Mount/Unmount](#)
- [Common File and Directory Operations](#)
- [Search Operations](#)

Tuning and Optimization of Nucleus Storage

The VFS metadata options provide a simple interface to the most frequently used tuning and optimizations available in the Storage component. These metadata options are noted below:

- `auto_mnt_pt_start_loc`

The starting mount point value used by the system when a device listed in `mnt_options` needs additional mount points.

- `mw_mnt_info`

The default device mount configuration used if a storage device entry in the platform file does not define the mount configuration.

- `include_check_disk`

Includes conditionally compiled support for the check disk API.

- `include_blk_cache`

Includes conditionally compiled support for the device layer block cache.

- `include_blk_cache_reorder`

Include support for block reordering.

- `max_open_files`

The maximum number of files that can be open simultaneously in the system.

- `max_devices`

The maximum number of Storage devices (includes logical and physical).

- `max_mount_points`

The maximum number of logical devices that can be mounted in the system.

- `max_file_systems`

The maximum number of file systems that can be registered.

- `include_cp_support`

Allows code page support for whatever code pages are defined.

Initialization

The Nucleus storage components gets initialized automatically (at Run-level 6) during the initialization sequence of the operating system. For more information on the run-level initialization sequence, see “Application Development Overview”, in the chapter “Developing a Nucleus ReadyStart Application”, in the *Nucleus ReadyStart Guide*.

The following components of the OS must successfully initialize first for the initialization of Nucleus storage component to be successful:

- Kernel (Nucleus PLUS)
- Device Manager
- Registry Service

The device drivers register storage devices with the Device Manager., The Device Manager then passes the driver information to the Nucleus FILE System and the drive is checked for partitions and mounted.

Addition of Storage Devices

Physical devices are added after the Device Manager notifies Nucleus FILE System of a new physical device. The FILE component relies on the Device Manager to supply device drivers. As physical devices are discovered by the VFS, the physical device is searched for a partition table. Logical devices are created for each partition, or a single logical device is created if no partition table exists. This is all handled automatically by VFS.

Storage Device Mount/Unmount

Drivers that register Storage devices with the Device Manager will be mounted automatically. Drivers that support hot-plugging will unmount devices using the Device Manager. Mounting and unmounting can also be accomplished using the following [Nucleus FILE System APIs](#):

- [NU_Mount_File_System](#)
- [NU_Unmount_File_System](#)

Logical devices are mounted according to the registry settings for that device, listed in [Table 2-2](#). The default settings are used if specific settings do not exist. The registry settings are also used to assign a drive letter, and are even capable of formatting devices which do not have a valid format.

The default registry configuration is stored in the VFS metadata file. These defaults can be over-ridden using the configuration options for a device in the platform configuration file. After making changes, the configuration must be re-generated by re-building the library.

Table 2-2. Registry Options for Mount Information

Option Name	Description	Default Value	Defined by Storage Device
fs	String describing the file system type.	“FAT” or “Safe”	YES
pt	Drive letter to associate with the device	“A” - “I”	YES

Table 2-2. Registry Options for Mount Information

Option Name	Description	Default Value	Defined by Storage Device
auto_fmt	Boolean that determines if device is formatted automatically if the mount fails.	true or false	YES

The following an example for registry mount information:

```
group("mw_settings") {  
    enregister true  
    option("fs") {  
        default "FAT"  
    }  
    option("pt") {  
        default "A"  
    }  
    option("auto_fmt") {  
        default false  
    }  
}
```

Common File and Directory Operations

Files and directory manipulation is quite simple using [Nucleus FILE System APIs](#). Files are created and opened using [NU_Open](#). The parameters available when opening a device allow the application to specify read and write access, new file creation, and sharing. The file descriptor returned by [NU_Open](#) is used for further access to the file through the other Nucleus FILE System APIs, such as:

- [NU_Read](#)
- [NU_Write](#)
- [NU_Seek](#)
- [NU_Truncate](#)
- [NU_Flush](#)
- [NU_Close](#)

Additionally, directories are manipulated using:

- [NU_Make_Dir](#)
- [NU_Remove_Dir](#)

Search Operations

There are three [Nucleus FILE System APIs](#) that can be used for searching file storage:

- [NU_Get_First](#)
- [NU_Get_Next](#)
- [NU_Done](#)

[NU_Get_First](#) is called using a path and string pattern to match file and directory names. The pattern may contain wildcard characters. The “*” wildcard will match any string pattern, while the “?” wildcard will only match one character; the results will all be from within the specified directory. After [NU_Get_First](#) has successfully returned a match, [NU_Get_Next](#) can be called to find the next matching object. Information about the matching object is returned in the [DSTAT](#) data structure. Upon successful return by [NU_Get_First](#) or [NU_Get_Next](#), [DSTAT](#) will contain information, including file name, time stamps, and access properties, related to the currently matched item in the directory.

Listing a directory's contents is another common application operation. This can be accomplished by looping through the results of [NU_Get_First](#) and [NU_Get_Next](#).

[NU_Done](#) is called when the application is finished with the current match. This clears and releases resources associated with the [DSTAT](#) structure.

Nucleus FILE System Internals

This section provides additional detail on the internal workings of the Nucleus FILE System components, including:

- [Nucleus Virtual File System Services](#)
- [Nucleus FAT File System](#)
- [Nucleus SAFE File System](#)

Nucleus Virtual File System Services

VFS Design

VFS is designed to be an extensible framework for tying together broad device and file system support. Storage device drivers written to comply with the driver interface are transparent to the application. VFS natively supports two file systems, SAFE and FAT. VFS is also capable of incorporating third party file systems using a compatibility layer.

VFS Operation

VFS is an internally managed library, so initialization and shutdown are all handled by the Registry Service. After initialization, application tasks can utilize the [Nucleus FILE System APIs](#) for all of their Storage requirements.

VFS Features

Following are some of the features of the Virtual File System Services:

Block Cache

Device-layer block caching for supported devices. Currently compatible with FAT devices, the block cache enables write caching that will speed up small, random write use-cases.

Code Page and Unicode Support

VFS supports code pages and Unicode used in file names. There is built in support for CP932, CP036, CP949, and CP950.

Partition support

VFS is capable of parsing a disk partition table for mountable volumes. The partition API can also be used to create and remove partitions.

Nucleus FAT File System

FAT File System Design

The Nucleus FAT File System component is designed to provide an ubiquitous storage solution. It was designed with embedded devices in mind and offers a highly configurable FAT file system support, a rich feature set, and a relatively low memory footprint.

FAT File System Operation

The FAT file system is a sub-component of VFS. As such, it's initialization is handled by VFS.

FAT File System Optimization

The FAT component can be optimized for applications using the metadata options.

FAT File System Features

Check disk

Since power failures can corrupt a FAT volume, a check disk feature is included that is capable of finding and correcting errors and inconsistencies on the volume.

File Allocation Table Buffering

FAT accesses are buffered in order to speed up file I/O operations. This feature is configurable in the metadata.

Nucleus SAFE File System

SAFE File System Design

The SAFE flash file system provides complete protection against power failure. The system may be stopped and restarted at any point with no data lost. After the restart, the previously completed state of the file system will be restored.

SAFE is proprietary, although it can be made available externally using other Nucleus products (ftp, USB-MTP). SAFE is used for flash memory devices such as embedded NAND, NOR, and DataFlash. Raw access to flash is required by the driver.

Nucleus provides a Win application that is useful for SAFE driver design. It is located in *os/storage/file/fs/safe/utls/fsmem.exe*

SAFE File System Operation

When a file is closed, data is automatically flushed from to the disk. Until this close takes place, the state of the file system is preserved. The application may also use the [NU_Flush](#) command to write the current state of the file to the disk and update its fail-safe state.

SAFE File System Features

Bad-Block Management

Blocks that have exceeded their wear-leveling limit cannot be erased and are not available for write access. Bad-blocks are recognized and accounted for in SAFE.

Wear Leveling

Flash devices are usually manufactured to a specification that includes a guaranteed number of write-erase cycles that can be performed on each block before developing a fault. It is important to use the blocks in a device evenly if the device is to be used for its maximum lifetime.

SAFE uses a process called dynamic wear leveling, when a new block is written, to allocate the least-used blocks from those available. However, in systems where there are large areas of static data (the executable binary for the system), these areas may be written only once, thus leaving a relatively small section of the device to handle the much more heavily used files. For this reason a process called static wear leveling is introduced. When the `fs_staticwear` function is called, it searches for blocks that have been used much less than the most used blocks in the system. If this difference is greater than a defined threshold (`FS_STATIC_DISTANCE`), then these two blocks will be exchanged in the system.

When the static wear leveling function is executing, the file system is not accessible. The length of time it takes to run the static wear leveling depends on the specification of the target chips being used and, in particular, the time required to erase a block and copy one block to another.

For static wear leveling to function, an additional driver function, BlockCopy, must also be provided. It is important to provide a highly optimized version of BlockCopy, preferably by using special copy functions that are specific to the target chip, in order to achieve the best system performance and least system disruption. Further guidance on when static wear leveling should be used is provided as follows:

- When does an application need static wear leveling?

In many cases, wear leveling is an unnecessary overhead. To assess its importance look at how the application is to be used and consider the specifications of the target devices. Many devices have up to 1 million guaranteed erase/write cycles per block and, in many applications, this number will not be reached in the lifetime of the product.

- When should an application execute static wear leveling?

Because wear-leveling involves swapping blocks in the file system, all access is excluded for the duration of the process. Thus, if there are time-critical features in the device, then it is preferable to do static wear-leveling during idle periods. The function should be called regularly during idle time for effective management of the system.

Long File Names

SAFE supports efficient handling for file names of almost unlimited length.



Note

Long file names use memory from the descriptor blocks in the file system. The system uses an efficient algorithm for allocating additional blocks in units of FS_MAXLFN. The use of long file names reduces the number of file and directory entries that can be stored.

Nucleus FILE System Data Structures

This chapter describes the following data structures associated with the Nucleus [Nucleus FILE System APIs](#) APIs:

- [CPTE_S](#)
- [DSTAT](#)
- [FS_S](#)

CPTE_S

This structure is used in [NU_Mount_File_System](#) to indicate the codepage with which a particular drive should be mounted.

Structure Definition

```
typedef struct codepage
{
    /* Set this to the codepage that you want. */
    UINT16 cp;
    CP_OP  cp_op;

}CPTE_S;
```

Related Topics

[Nucleus FILE System Data Structures](#)

DSTAT

This structure is used by [NU_Get_Next](#) and [NU_Done](#) to traverse the file system. It contains private information used to keep track of where the next directory entry resides and public information used to report the status of a directory entry back to you. DSTAT structures are created by calling [NU_Get_First](#) and destroyed by calling [NU_Done](#). The size of a DSTAT structure is 813 bytes.

This is a generic structure and all fields do not pertain to data in all file systems.

Structure Definition

```
/* File search structure */
typedef struct dstat_struct
{
    CHAR    sfname[9];           /* Null terminated file and extension*/
    CHAR    fext[4];
    CHAR    lfname[EMAXPATH+1]; /* Null terminated long file name */
    UINT8    fattribute;         /* File attributes */
    UINT8    fcrcmsec;          /* File create centesimal millisecond */
    UINT16    fcrttime;          /* File create time */
    UINT16    fcrdate;          /* File create date */
    UINT16    faccddate;        /* Access date */
    UINT16    fclusterhigh;     /* High cluster for data file */
    UINT16    fuptime;          /* File update time */
    UINT16    fupdate;          /* File update date */
    UINT16    fclusterlow;      /* Low cluster for data file */
    UINT32    fsize;            /* File size */

    struct mte_struct *fs_mte; /* MTE for subsequent calls */
    VOID    *fs_private;      /* For file system specific data*/
    UINT32    drive_id;        /* Unique ID for determining validity */
    UINT16    dh;              /* Disk handle for this object */
} DSTAT;
```

Related Topics

[Nucleus FILE System Data Structures](#)

[NU_Get_First](#)

[NU_Get_Next](#)

[NU_Done](#)

FS_S

This structure is used by the [NU_Register_File_System](#) service to allow provide access to file system operations of the newly registered type.

Structure Definition

```
/* File system operations structure */
typedef struct fs_struct
{
    STATUS (*fs_init)      (VOID);
    STATUS (*fs_uninit)    (VOID);
    STATUS (*fs_check_disk)(UINT16 dh,UINT8 flag,UINT8 mode);

    STATUS (*fs_ftime)     (DSTAT *statobj,
                           UINT16 access_date,
                           UINT16 access_time,
                           UINT16 update_date,
                           UINT16 update_time,
                           UINT16 create_date,
                           UINT16 create_time);

    STATUS (*fs_get_format_info)(UINT16 dh,
                                VOID **params);

    STATUS (*fs_mount)(UINT16 dh,
                       VOID*  config);

    STATUS (*fs_unmount)    (UINT16 dh);
    STATUS (*fs_disk_abort) (UINT16 dh);

    /* NU_Disk_Abort */
    #if (CFG_NU_OS_STOR_FILE_VFS_INCLUDE_CHECK_DISK == 1)
        STATUS (*fs_check_disk) (UINT16 dh,
                                UINT8  flag,
                                UINT8  mode);
    #endif

    /* NU_Check_Disk */
    #endif

    STATUS (*fs_open) (UINT16 dh,
                       CHAR   *name,
                       UINT16 flag,
                       UINT16 mode);

    /* NU_Open */
    INT32 (*fs_read) (INT    fd,
                     CHAR   *buf,
                     INT32 count);
}
```

```
/* NU_Read */
INT32 (*fs_write) (INT    fd,
                  CHAR    *buf,
                  INT32    count);

/* NU_Write */
INT32 (*fs_seek) (INT    fd,
                 INT32    offset,
                 INT16    origin);

/* NU_Seek */
STATUS (*fs_close) (INT fd);

/* NU_Close */
STATUS (*fs_delete) (CHAR *name);

/* NU_Delete */
STATUS (*fs_utime) (DSTAT *statobj,
                  UINT16 access_date,
                  UINT16 access_time,
                  UINT16 update_date,
                  UINT16 update_time,
                  UINT16 create_date,
                  UINT16 create_time);

STATUS (*fs_set_attr) (CHAR *name,
                     UINT8 newattr);

/* NU_Set_Attributes */
STATUS (*fs_get_attr) (UINT8 *attr,
                     CHAR *name);

/* NU_Get_Attributes */
STATUS (*fs_truncate) (INT    fd,
                     INT32    offset);

/* NU_Truncate */
STATUS (*fs_flush) (INT fd);

/* NU_Flush */
STATUS (*fs_mkdir) (CHAR *name);

/* NU_Make_Dir */
STATUS (*fs_rmdir) (CHAR *name);

/* NU_Remove_Dir */
STATUS (*fs_rename) (CHAR *name,
                   CHAR *newname);

/* NU_Rename */
STATUS (*fs_get_first) (DSTAT *statobj,
                     CHAR *pattern);

/* NU_Get_First */
STATUS (*fs_get_next) (DSTAT *statobj);

/* NU_Get_Next */
```

```

STATUS (*fs_done) (DSTAT *statobj);

/* NU_Done */
STATUS (*fs_format) (UINT16 dh,
                    VOID **params);

/* NU_Format */
STATUS (*fs_get_format_info) (UINT16 dh,
                             VOID **params);

/* NU_Get_Format_Info */
STATUS (*fs_freespace) (UINT16 dh,
                       UINT8 *secpcluster,

                       /* NU_Freespace */
                       UINT16 *bytepsec,
                       UINT32 *freecluster,
                       UINT32 *totalcluster);

STATUS (*fs_vnode_allocate) (UINT16 dh,
                             CHAR *path,
                             VOID **fsnode);

STATUS (*fs_vnode_deallocate) (UINT16 dh,
                              VOID *fsnode);

STATUS (*fs_fsnode_to_string) (UINT16 dh,
                              VOID *fsnode,
                              CHAR *string);

VOID (*fs_release) (UINT16 dh);

/* Release FS locks for yielding */
VOID (*fs_reclaim) (UINT16 dh);

/* Reclaim FS locks after yielding */

} FS_S;

```

Related Topics

[Nucleus FILE System Data Structures](#)

[NU_Register_File_System](#)

Nucleus FILE System APIs

The Nucleus FILE system can be accessed using the public services defined by the [Nucleus FILE System APIs](#). Most Nucleus FILE services provide return values for the status of the request. Services can include a description of legacy error codes for compatibility with previous versions of Nucleus FILE.

Some of the Nucleus FILE API functions set `fs_user->p_errno` for compatibility with previous versions of Nucleus FILE. These legacy error codes are noted in the following reference section.

- [NU_Become_File_User](#) (Deprecated)
- [NU_Check_Disk](#)
- [NU_Check_File_User](#) (Deprecated)
- [NU_Close](#)
- [NU_Close_Disk](#) (Deprecated)
- [NU_Create_Partition](#)
- [NU_Current_Dir](#) (Deprecated)
- [NU_Delete](#)
- [NU_Delete_Partition](#)
- [NU_Disk_Abort](#)
- [NU_Done](#)
- [NU_FILE_Cache_Create](#)
- [NU_FILE_Cache_Destroy](#)
- [NU_FILE_Cache_Flush](#)
- [NU_FILE_Cache_Get_Config](#)
- [NU_FILE_Cache_Set_Config](#)
- [NU_Flush](#)
- [NU_Format](#)
- [NU_Free_List](#)
- [NU_Free_Partition_List](#)
- [NU_FreeSpace](#)
- [NU_Get_Attributes](#)
- [NU_Get_Default_Drive](#) (Deprecated)
- [NU_Get_First](#)
- [NU_Get_Format_Info](#)
- [NU_Get_Next](#)
- [NU_Get_Partition_Info](#)
- [NU_List_Device](#)
- [NU_List_File_System](#)

- [NU_List_Mount](#)
- [NU_List_Partitions](#)
- [NU_Make_Dir](#)
- [NU_Mount_File_System](#)
- [NU_Open](#)
- [NU_Open_Disk](#) (Deprecated)
- [NU_Read](#)
- [NU_Register_File_System](#)
- [NU_Release_File_User](#) (Deprecated)
- [NU_Remove_Dir](#)
- [NU_Rename](#)
- [NU_Seek](#)
- [NU_Set_Attributes](#)
- [NU_Set_Current_Dir](#) (Deprecated)
- [NU_Set_Default_Drive](#) (Deprecated)
- [NU_Storage_Device_Wait](#)
- [NU_Truncate](#)
- [NU_Unmount_File_System](#)
- [NU_Unregister_File_System](#)
- [NU_Utime](#)
- [NU_Write](#)

API Notes

- The use of wild cards with Nucleus SAFE is limited to the following APIs at this time:
 - [NU_Get_First](#) can be called with the “*.*” wild card.
 - [NU_Delete](#) can be called with “*”, “*.*”, or “*.[extension]”, such that [extension] is a three character file extension. Currently, these wildcards can be used as the last path entry or as the only strings passed in.
 - [NU_Get_Format_Info](#) sets **params to NU_NULL, because currently there are no file system-specific format parameters needed with Safe. You should still format the

drive following the example provided by the [NU_Format](#) API to ensure future compatibility.

- The Cache Services APIs are not supported by the SAFE file system. Cache services includes the following routines:
 - [NU_FILE_Cache_Create](#)
 - [NU_FILE_Cache_Destroy](#)
 - [NU_FILE_Cache_Flush](#)
 - [NU_FILE_Cache_Get_Config](#)
 - [NU_FILE_Cache_Set_Config](#)
- Because devices might not all be available to the application at start-up, the application can request to suspend waiting for a device using the [NU_Storage_Device_Wait](#) API.
- The user interface and environment functions have been deprecated. They remain functional to provide backward compatibility for legacy applications, however calling them will produce compiler warnings. Previously, tasks requiring file system access needed to call [NU_Become_File_User \(Deprecated\)](#) (to register the task with VFS) and then call [NU_Open_Disk \(Deprecated\)](#) for each drive that was to be used. After a task was finished using the disk, it was expected to call [NU_Close_Disk \(Deprecated\)](#) and [NU_Release_File_User \(Deprecated\)](#). Additionally, this method created a user environment, such that each user had a separate default drive and current working directory.

Now there is a global environment, and all tasks are allowed to access all drives without becoming a file user and opening the disks first. If a non-registered task (a task that has not called [NU_Become_File_User \(Deprecated\)](#)) calls [NU_Set_Default_Drive \(Deprecated\)](#) or [NU_Set_Current_Dir \(Deprecated\)](#), the task will change the settings for all other non-registered tasks. It is recommended that file operations requiring a path use a fully qualified path (drive letter, directory, and file name).

NU_Become_File_User (Deprecated)

Caution



This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Task Operations

Source location: *fs_users.c*

This function registers a task as a file system user.

A task that calls this function will be required to also call [NU_Open_Disk \(Deprecated\)](#) for each drive it requires access to. A task that has become a file user will also be provided a separate user environment, so calls to [NU_Set_Default_Drive \(Deprecated\)](#) and [NU_Get_Default_Drive \(Deprecated\)](#), and [NU_Set_Current_Dir \(Deprecated\)](#) and [NU_Current_Dir \(Deprecated\)](#) will only pertain to the calling task.

Usage

```
STATUS NU_Become_File_User (VOID)
```

Return Values

- **NU_SUCCESS**
Request completed successfully.
- **NUF_INTERNAL**
Failed, too many users.
- **NU_INVALID_MEMORY**
User_heap has not been initialized.

Example

```
#include "storage/nu_storage.h"

if (NU_Become_File_User() != NU_SUCCESS)
    printf("NU_Become_File_User error!");
```

Related Topics

[NU_Check_File_User \(Deprecated\)](#)

[NU_Release_File_User \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Check_Disk

Category: Drive Operations

Source location: *fs_dsk.c*

This function checks the disk for issues determined by the flags passed in and reports and fixes the issues depending on the value of mode. The check disk API is only compatible with the FAT file system.

Note



No other task can access the disk while check disk is being run.

Usage

```
STATUS NU_Check_Disk (CHAR  *path,  
                     UINT8  flag,  
                     UINT8  mode)
```

Arguments

- path
Specifies the drive letter on which to run check disk.
- flag
Specifies the type of check to run:
 - CHK_CHECK_LOST_CL_CHAIN
Checks the disk for lost cluster chains.
 - CHK_CHECK_CROSS_LINKED_CHAIN
Checks the disk for cross-linked chains.
 - CHK_CHECK_FILES_SIZES
Checks the disk's files for invalid file lengths.
 - CHK_CHECK_DIR_RECORDS
Checks the disk's directory records.
 - CHK_CHECK_FAT_TABLE
Checks the disk's FAT tables to make sure they match.
- mode
Indicates if check disk should attempt to fix all issues found and log the results or just log the results.
 - CHK_REPORT_ERRORS_ONLY
Check disk only reports the errors found in the log file.
 - CHK_FIX_ERRORS
Check disk attempts to fix all errors encountered and log them.

Return Values

- NUF_ACCES
Another task has this disk open. Check disk requires exclusive access to the disk in order to run.
- NUF_BADDRIIVE
Invalid drive specified.
- NUF_LOG_FILE_CREATED
Errors were found and a log file was created to report them.
- NUF_FAT_TABLES_DIFFER
FAT tables differ so the log file could not be created.

Example

```
#include "storage/nu_storage.h"
STATUS ret_val;
..
..

ret_val = NU_Check_Disk("C:", CHK_CHECK_LOST_CL_CHAIN |
                        CHK_CHECK_CROSS_LINKED_CHAIN |
                        CHK_CHECK_FILES_SIZES |
                        CHK_CHECK_DIR_RECORDS | CHECK_FAT_TABLES,
                        CHK_REPORT_ERRORS_ONLY);

/* Check the C drive for errors. */
if (ret_val == NUF_FAT_TABLES_DIFFER)
{
    printf("Fat Tables differ so have check disk fix them.!");
    ret_val = NU_Check_Disk("C:",
                            CHECK_FAT_TABLES,
                            CHK_FIX_ERRORS)
}

/* Now report only the other errors */
if (ret_val == NU_SUCCESS)
{
    NU_Check_Disk("C:", CHK_CHECK_LOST_CL_CHAIN |
                  CHK_CHECK_CROSS_LINKED_CHAIN |
                  CHK_CHECK_FILES_SIZES |
                  CHK_CHECK_DIR_RECORDS,
                  CHK_REPORT_ERRORS_ONLY)
}
```

Related Topics

[Nucleus FILE System APIs](#)

NU_Check_File_User (Deprecated)

 **Caution** This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Task Operations

Source location: *fs_users.c*

This function checks if the current task is a file system user.

Usage

```
STATUS NU_Check_File_User (VOID)
```

Return Values

- **NU_SUCCESS**
The task may use the file system.
- **NUF_BAD_USER**
The task is not registered.

Example

```
#include "storage/nu_storage.h"

if (!NU_Check_File_User() != NU_SUCCESS)
    printf("NU_Check_File_User error!");
```

Related Topics

[NU_Become_File_User \(Deprecated\)](#)

[NU_Release_File_User \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Close

Category: Simple File Operations

Source location: *fs_fl.c*

This function closes the file, updates the disk and frees all cores associated with the file descriptor.

Usage

```
STATUS NU_Close (INT vfs_fd)
```

Arguments

- `vfs_fd`
The file descriptor of the file to be closed.

Return Values

- `NU_SUCCESS`
Close request completed successfully.
- `NUF_BAD_USER`
Task not registered as a file user.
- `NUF_BADFILE`
Invalid file descriptor.
- `NUF_IO_ERROR`
Driver I/O function routine returned error.
- `NUF_NO_DISK`
Disk is removed.
- `NUF_INTERNAL`
Nucleus FILE internal error.

Legacy Error Codes

- `PEBADF`
Invalid file descriptor.
- `PENOSPC`
I/O error.

Example

```
#include "storage/nu_storage.h"
INT fd;
if (NU_Close(fd) < 0)
    printf("Error closing file");
```

Related Topics

[NU_Open](#)

[Nucleus FILE System APIs](#)

NU_Close_Disk (Deprecated)

Caution



This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Drive Operations

Source location: *fs_dsk.c*

This function releases the file user's resources allocated for maintaining the environment. This function is only required for tasks that have called [NU_Become_File_User \(Deprecated\)](#).

Usage

```
STATUS NU_Close_Disk (CHAR *path)
```

Arguments

- path
Specifies the drive letter for the disk to be closed.

Return Values

- NU_SUCCESS
Close Disk request completed successfully.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_NOT_OPENED
The disk is not opened yet.
- NUF_BADDRIIVE
Invalid drive specified.

Example

```
#include "storage/nu_storage.h"

STATUS status;

/* Assumes the disk is mounted and currently opened by the user */
status = NU_Close_Disk("A:");
```

Related Topics

[NU_Open_Disk \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Create_Partition

Category: Partition Services

Source location: *fs_part.c*

This function creates the specified partition on a device.

Usage

```
STATUS NU_Create_Partition (CHAR    *dev_name,  
                           UINT16  part_type,  
                           UINT32  size,  
                           UINT32  offset,  
                           UINT8   part_id)
```

Arguments

- **dev_name**
Unique string describing the physical device to be modified.
- **part_type**
Primary (0), extended (1), or logical (2)
- **size**
Unspecified (0), or a value in MiB (2^{20} bytes). The size may be adjusted to fall on a cylinder boundary.
- **offset**
Unspecified (0xFFFFFFFF), or a value in MiB (2^{20} bytes). The offset may be adjusted to fall on a cylinder boundary.
- **part_id**
The partition identifier written into the partition table. This value must correspond to the file system type with which the partition is formatted.

Return Values

- **NUF_INVALID_DEVNAME**
Matching device name was not found.
- **NUF_INTERNAL**
Nucleus FILE internal error.
- **NUF_NO_MEMORY**
Cannot allocate internal buffer.
- **NUF_BADPARAM**
Invalid parameter given.
- **NUF_PART_TABLE_FULL**
Only four primaries, or one extended and three primaries can exist.

- NUF_PART_EXT_EXISTS
Only one extended partition can exist.
- NUF_PART_NO_EXT
Logical partitions can only be created within an extended partition.

Description

If the size is zero, the size of the free area is calculated. If the offset is unspecified (0xFFFFFFFF), the first free area on the disk is used. The values specified in size and offset may be adjusted so the start and end of the partition lie on cylinder boundaries (for example, cylinder snapped).

Note



The Partition API's are NOT compatible with Flash devices. Additionally, partitions cannot be formatted with the SAFE file system.

Example

```
#include "storage/nu_storage.h"

/* Create one FAT32 primary partition that occupies the first free area
   encountered on the disk. */

if (NU_Create_Partition("hda", FPART_PRIMARY, 0, 0xFFFFFFFF, 0x0C) < 0)
    printf("Error creating partition.\n");
```

Related Topics

[NU_Delete](#)

[NU_List_Partitions](#)

[NU_Get_Partition_Info](#)

[Nucleus FILE System APIs](#)

NU_Current_Dir (Deprecated)

 **Caution** This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Drive Operations

Source location: *fs_env.c*

This function writes to a string the full path name of the current working directory.

Usage

```
STATUS NU_Current_Dir (CHAR *drive,  
                      CHAR *path)
```

Arguments

- **drive**
The letter of the drive for which the current directory is requested. If the pointer to the drive structure is null, or an invalid drive is specified, the default drive is used.
- **path**
This is filled in with a string of the full path name of the current working directory.

Return Values

- **NU_SUCCESS**
CurrentDir request completed successfully.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_INVNAME**
Path or filename is invalid.
- **NUF_BADDRIIVE**
Invalid drive specified.

Example

```
UINT8 pwd[MAXPATH];  
if (NU_SUCCESS == NU_Current_Dir("A:", pwd))  
    printf ("Working dir is %s", pwd);  
else  
    printf ("Can't find working dir for A:");
```

Related Topics

[NU_Set_Current_Dir \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Delete

Category: Simple File Operations

Source location: *fs_fl.c*

This function deletes the file specified by name.

Usage

```
STATUS NU_Delete (CHAR *name)
```

Arguments

- name
Specifies the file name to be deleted.

Return Values

- NU_SUCCESS
Delete request completed successfully.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_IO_ERROR
Driver I/O function routine returned error.
- NUF_NOFILE
The specified file not found.
- NUF_ACCES
This file has at least one of the following attributes: RDONLY, HIDDEN, SYSTEM, VOLUME, DIRENT.

Description

The NU_Delete function fails if any of the following conditions are true:

- the file is not a simple file
- the file is open
- the file does not exist
- the file is read only.

Note



This is only a limitation of the SAFE file system. When using the SAFE file system, NU_Delete can be called with “*”, “*.*”, or “*.[extension]”, such that [extension] is a three character file extension. Currently, these wildcards can be used as the last path entry or as the only strings passed in.

Example

```
#include "storage/nu_storage.h"

if (NU_SUCCESS != NU_Delete( "B:\\\\USR\\TEMP\\TMP001.PRN" ))
    printf("Can't delete file \n");
```



Note

When you use wildcards for the filename, if NU_Delete returns NU_SUCCESS, it means at least one file is deleted. Therefore, if NU_Delete("**.*") is issued in a directory that contains four read only files and one other file, NU_SUCCESS is returned and one file is deleted.

Related Topics


[Nucleus FILE System APIs](#)

NU_Delete_Partition

Category: Partition Services

Source location: *fs_part.c*

This function matches the associated partition and removes it from the partition table. Extended partitions must be empty (no logical partitions exist).

 **Note** The Partition API's are NOT compatible with Flash devices. Additionally, partitions cannot be formatted with the SAFE file system.

Usage

```
STATUS NU_Delete_Partition (CHAR *log_dev_name)
```

Arguments

- `log_dev_name`
Unique string describing the partition to be removed.

Return Values

- `NUF_INVALID_DEVNAME`
Matching device name was not found.
- `NUF_INTERNAL`
Nucleus FILE internal error.
- `NUF_PART_LOG_EXISTS`
Extended partition cannot be removed because a logical partition still exists.
- `NU_SUCCESS`
Partition removed.

Example

```
#include "storage/nu_storage.h"

if (NU_Delete_Partition("hda00") != NU_SUCCESS)
    printf("Error removing partition");
```

Related Topics

[NU_Create_Partition](#)

[NU_List_Partitions](#)

[Nucleus FILE System APIs](#)

NU_Disk_Abort

Category: Drive Operations

Source location: *fs_dsk.c*

This function aborts all operations on a disk.

Usage

```
STATUS NU_Disk_Abort (CHAR *path)
```

Arguments


- path
Specifies the drive letter on which to abort the operation.


Return Values

- NU_SUCCESS
Abort request completed successfully.
- NUF_BAD_USER
Task not registered as a file user.
- <0
File system specific status error value.

Description

If an application senses that there are problems with a disk, it should call `NU_Disk_Abort`. This causes all user and file system specific resources associated with that drive to be freed. No disk operations will be attempted. All file descriptors associated with the drive become invalid. After correcting the problem call [NU_Unmount_File_System](#) to unmount the disk, [NU_Mount_File_System](#) to remount the disk, and [NU_Open_Disk \(Deprecated\)](#) to reopen your files.

 **Note** This function is not valid in SAFE file systems.

 **Note** This service does not flush any buffers to disk.

Example

```
#include "storage/nu_storage.h"

/* Clear everything so we can get a fresh start */
if (NU_Disk_Abort("A:") != NU_SUCCESS)
    printf("NU_Disk_Abort error!");
```

Related Topics

[Nucleus FILE System APIs](#)

NU_Done

Category: Directory Operations

Source location: *fs_dir.c*

This function frees internal resources used by [NU_Get_First](#) and [NU_Get_Next](#).

Usage

```
STATUS NU_Done (DSTAT *statobj)
```

Arguments

- **statobj**
Pointer to a data structure, which maintains file/subdirectory information useful in traversing a directory tree.

Return Values

- **NU_SUCCESS**
Resources successfully returned.
- **NUF_NO_DISK**
Disk is removed.
- **NUF_NOT_OPENED**
The disk is not opened yet.
- **NUF_BAD_USER**
Task is not a registered file user.
- **NUF_BADPARAM**
Invalid parameter given.

Description

Given a pointer to a [DSTAT](#) structure set up by a call to [NU_Get_First](#) free internal elements used by the DSTAT.

Note



You must call this function when finished searching through a directory.

Example

```
#include "storage/nu_storage.h"
DSTAT statobj;

/* A simple directory list subroutine. */
if (NU_Get_First(&statobj, path) == NU_SUCCESS)
{
    while(1)
    {
```



```
printf("%s",statobj.sfname);
if ( statobj.fext[0] != ' ' )
    printf("%.s ", statobj.fext );
else
    printf("      ");
    if ( statobj.fattribute & ADIRENT)
    {
        printf(" <DIR>");
    }
    else if (statobj.fattribute & AVOLUME)
    {
        printf(" <VOL>");
    }
    else
    {
        printf("%6d", statobj.fsize );
    }
printf("  %d-%d-%d %d:%d:%d ",
        ((statobj.fupdate >> 9 ) & 0x007f) + 1980,
        ((statobj.fupdate >> 5 ) & 0x000f),
        (statobj.fupdate & 0x001f),
        ((statobj.fuptime >> 11 ) & 0x001f ),
        ((statobj.fuptime >> 5 ) & 0x003f ),
        ((statobj.fuptime << 1 ) & 0x003f ));
printf(" %s\n", statobj.lfname );
if (NU_Get_Next(&statobj) != NU_SUCCESS)
{
    NU_Done(&statobj);
    break;
}
}
}
```

Related Topics

[NU_Get_First](#)

[NU_Get_Next](#)

[Nucleus FILE System APIs](#)

NU_FILE_Cache_Create

Category: Cache Services

Source Location: *bcm.c*

This function creates a device layer block cache.

Usage

```
STATUS NU_FILE_Cache_Create (CHAR          *path,  
                             BCM_CACHE_TYPE cache_type,  
                             VOID          *cache_config)
```

Arguments

- **path**
The name of a mounted drive: “A:”, “B:”, and so on.
- **cache_type**
BCM_CACHE_TYPE_DL_LRU is currently the only supported cache type.
- **cache_config**
Cache configuration arguments for desired cache on given path.

Return Values

- **NU_SUCCESS**
Cache was created successfully.
- **NU_INVALID_SEMAPHORE**
An internal error was encountered, unable to create the cache.
- **NUF_BADDRIVE**
Drive is not mounted. Drive must be mounted prior to activating a cache.
- **NU_UNAVAILABLE**
Cache type is not supported. BCM_CACHE_TYPE_DL_LRU is currently the only supported cache type.
- **NU_NO_MEMORY**
Insufficient memory available for creating the cache.

Description

This function creates a device layer block cache of the specified type using the configuration parameters in *cache_config* for the mounted drive at *path*. The drive must be mounted and not have an existing active cache.

Note



This function is not valid in SAFE file systems.

Example

```
STATUS sts;
/* Assumes that "FAT" file system is registered with the system.
   Assumes that the device "rd00" was successfully created. */
sts = NU\_Mount\_File\_System ("FAT", "A", "rd00", NU_NULL);
if (sts == NU_SUCCESS)
{
    BCM_DL_LRU_CONFIG config;
    config.size = (512*5);          /* 5 * 512 byte blocks */
    config.periodic_flush = NU_NO_SUSPEND; /* Periodic flush is disabled */
    config.low_threshold = 5;
    config.high_threshold = 5;      /* Threshold processing is disabled */

    /* Activate a cache with the given parameters on "A:" */
    sts = NU\_FILE\_Cache\_Create ("A:", BCM_CACHE_TYPE_DL_LRU, &config);
}
```

Related Topics

[Nucleus FILE System APIs](#)

[NU_Mount_File_System](#)

NU_FILE_Cache_Destroy

Category: Cache Services

Source Location: *bcm.c*

This function destroys an active cache at a specified path.

Usage

```
STATUS NU_FILE_Cache_Destroy (CHAR *path)
```

Arguments

- path
Path of the currently active cache, "A:".

Return Values

- NU_SUCCESS
Cache was successfully destroyed. No device IO errors were encountered during the operation.
- NUF_BADPARAM
Invalid path pointer parameter.
- NU_UNAVAILABLE
Device does not have an active cache.
- NUF_IO_ERROR
An IO error was encountered while flushing dirty blocks from the cache. Check the error log for details.

Description

All dirty blocks are flushed from the cache in the calling context. Errors encountered during the flush are reported to the error log and return status. All resources allocated for managing the cache are returned to the system.

Note



This function is not valid in SAFE file systems.

Example

```
STATUS sts;  
/* Assumes that "FAT" file system is registered with the system.  
   Assumes that the device "rd00" was successfully created. */  
sts = NU_Mount_File_System ("FAT", "A", "rd00", NU_NULL);  
  
if (sts == NU_SUCCESS)  
{  
    BCM_DL_LRU_CONFIG config;  
    config.size          = (512*5);          /* 5 * 512 byte blocks      */  
}
```

```
config.periodic_flush = NU_NO_SUSPEND; /* Periodic flush is disabled */
config.low_threshold  = 5;
config.high_threshold = 5;          /* Threshold processing is disabled */

/* Activate a cache with the given parameters on "A:" */
sts = NU_FILE_Cache_Create ("A:", BCM_CACHE_TYPE_DL_LRU,
                           (VOID*)&config);
}

if (sts == NU_SUCCESS)
{
    sts = NU_FILE_Cache_Destroy ("A:");
}
```

Related Topics

[Nucleus FILE System APIs](#)

NU_FILE_Cache_Flush

Category: Cache Services

Source Location: *bcm.c*

This function flushes the cache at the specified path. All dirty blocks are written to the device. I/O errors are reported to the error log and returned via the return status.

Note



This function is not valid in SAFE file systems.

Usage

```
STATUS NU_FILE_Cache_Flush (CHAR *path)
```

Arguments

- path
Path of the device to flush, "A:".

Return Values

- NU_SUCCESS
Cache was flushed successfully. No I/O errors were encountered.
- NU_INVALID_SEMAPHORE
An internal error was encountered, cache may be in the process of being shutdown.
- NUF_BADPARAM
Invalid parameter give as NULL.
- NU_UNAVAILABLE
Device does not have a currently active cache.
- NUF_IO_ERROR
One or more IO errors were encountered while flushing the cache. Check the error log for details.

Example

```
STATUS          sts;  
BCM_DL_LRU_CONFIG config;  
INT fd;  
CHAR outBuf[512];  
  
/* Assumes that "FAT" file system is registered with the system. Assumes  
   that the device "rd00" was successfully created. */  
sts = NU_Mount_File_System ("FAT", "A", "rd00", NU_NULL);
```

```
if (sts == NU_SUCCESS)
{
    config.size          = (512*5);          /* 5 * 512 byte blocks */
    config.periodic_flush = NU_NO_SUSPEND; /* Periodic flush is disabled */
    config.low_threshold  = 5;
    config.high_threshold = 5;              /* Threshold processing is disabled */

    /* Activate a cache with the given parameters on "A:" */
    sts = NU_FILE_Cache_Create ("A:", BCM_CACHE_TYPE_DL_LRU,
                               (VOID*)&config);
}

if (sts == NU_SUCCESS)
{
    fd = NU_Open ("A:\\file.txt", (PO_TEXT|PO_RDWR|PO_CREAT),
                 (PS_IWRITE|PS_IREAD));
    if (fd >= 0)
    {
        /* Write something to the file */
        NU_Write (fd, &outBuf[0], 512);

        /* Force the cache to flush */
        sts = NU_FILE_Cache_Flush ("A:");
    }
}
```

Related Topics

[Nucleus FILE System APIs](#)

NU_FILE_Cache_Get_Config

Category: Cache Services

Source Location: *bcm.c*

This function retrieves the current cache configuration settings for the active cache at the specified path. The returned data type is dependent on the *cache_type* provided when the cache was created.

Note



This function is not valid in SAFE file systems.

Usage

```
STATUS NU_FILE_Cache_Get_Config (CHAR *path,  
                                VOID **config)
```

Arguments

- *path*
Path of the active cache to retrieve settings, "A:".
- *config*
Pointer to address of configuration structure to store settings.

Return Values

- **NU_SUCCESS**
Configuration settings were successfully returned.
- **NUF_BADPARAM**
Parameter given is null.
- **NUF_BADDRIVE**
Path given does not exist.
- **NU_UNAVAILABLE**
Device does not have an active cache.

Example

```
STATUS          sts;  
BCM_DL_LRU_CONFIG config;  
BCM_DL_LRU_CONFIG *pConfig;  
  
pConfig = &config;  
  
/* Assumes that "FAT" file system is registered with the system.  
   Assumes that the device "rd00" was successfully created. */  
sts = NU_Mount_File_System ("FAT", "A", "rd00", NU_NULL);
```



```
if (sts == NU_SUCCESS)
{
    config.size          = (512*5);          /* 5 * 512 byte blocks */
    config.periodic_flush = NU_NO_SUSPEND; /* Periodic flush is disabled */
    config.low_threshold  = 5;
    config.high_threshold = 5;              /* Threshold processing is disabled */

    /* Activate a cache with the given parameters on "A:" */
    sts = NU_FILE_Cache_Create ("A:", BCM_CACHE_TYPE_DL_LRU,
                               (VOID*)&config);
}

if (sts == NU_SUCCESS)
{
    sts = NU_FILE_Cache_Get_Config ("A:", (VOID**)&pConfig);
}
```

Related Topics

[Nucleus FILE System APIs](#)

NU_FILE_Cache_Set_Config

Category: Cache Services

Source Location: *bcm.c*

This function modifies the cache configuration settings for the given path.

Note



This function is not valid in SAFE file systems.

Note



The cache size may not be modified while active. Destroy and recreate the cache with new cache size settings if necessary. The given config structure must be of the data type identified when the cache was created.

Usage

```
STATUS NU_FILE_Cache_Set_Config (CHAR *path,  
                                VOID *config)
```

Arguments

- path
Path of the active cache to modify settings, "A:".
- config
Pointer to configuration structure.

Return Values

- NU_SUCCESS
Configuration settings were successfully modified.
- NUF_BADPARAM
Parameter given is null or a configuration value is invalid.
- NU_UNAVAILABLE
Device does not have an active cache.

Example

```
STATUS          sts;  
BCM_DL_LRU_CONFIG config;  
BCM_DL_LRU_CONFIG *pConfig;  
  
pConfig = &config;  
  
/* Assumes that "FAT" file system is registered with the system.  
   Assumes that the device "rd00" was successfully created. */  
sts = NU_Mount_File_System ("FAT", "A", "rd00", NU_NULL);
```

```
if (sts == NU_SUCCESS)
{
    config.size          = (512*5);          /* 5 * 512 byte blocks          */
    config.periodic_flush = NU_NO_SUSPEND; /* Periodic flush is disabled */
    config.low_threshold  = 5;
    config.high_threshold = 5;              /* Threshold processing is disabled */

    /* Activate a cache with the given parameters on "A:" */
    sts = NU_FILE_Cache_Create ("A:", BCM_CACHE_TYPE_DL_LRU,
                                (VOID*)&config);
}

if (sts == NU_SUCCESS)
{
    sts = NU_FILE_Cache_Get_Config ("A:", (VOID**)&pConfig);
    if (sts == NU_SUCCESS)
    {
        /* Modify the cache configuration */
        config.periodic_flush = 100; /* Perform periodic flush at 10
                                     ticks when cache contains dirty
                                     blocks */
        sts = NU_FILE_Cache_Set_Config ("A:", (VOID*) &config);
    }
}
```

Related Topics

[Nucleus FILE System APIs](#)

NU_Flush

Category: Simple File Operations

Source location: *fs_fl.c*

This function moves all data in a file's buffer onto the disk.

Usage

```
STATUS NU_Flush (INT fd)
```

Arguments

- `fd`
The file descriptor of the file to be flushed.

Return Values

- `NU_SUCCESS`
Operation successful.
- `NUF_BAD_USER`
Task not registered as a file user.
- `NUF_IO_ERROR`
Driver I/O function routine returned error.
- `NUF_BADFILE`
Invalid file descriptor.
- `NUF_NO_DISK`
Disk is removed.
- `NUF_INTERNAL`
Nucleus FILE internal error.

Legacy Error Codes

- `PEBADF`
Invalid file descriptor.
- `PENOSPC`
I/O error.

Example

```
#include "storage/nu_storage.h"
STATUS status;
INT fd;

/* Assume something is written to file here. */
```

```
/* Flush the file buffer. */  
status = NU_Flush(fd)
```

Related Topics

[Nucleus FILE System APIs](#)

NU_Format

Category: Drive Operations

Source location: *fs_frmt.c*

This function formats a specified device.

Usage

```
STATUS NU_Format (CHAR *fs_name,  
                  CHAR *dev_name,  
                  VOID **params )
```

Arguments

- **fs_name**
Name of registered file system to request formatting.
- **dev_name**
Name of a created non-mounted device to be formatted.
- **params**
Pointer to address of file system specific formation arguments.

Return Values

- **NU_SUCCESS**
Operation successful.
- **NUF_BADPARAM**
Invalid parameter specified.
- **NUF_IN_USE**
Device is currently mounted or in use.
- **NUF_FS_NOT_FOUND**
File system requested not registered.
- **<0**
File system specific error value.

Description

Given a registered file system name, created non-mounted logical device name, and a pointer to the location of file system specific format parameters, the file system specific format operation prepares the named device. See the definition of the file system specific format data structure for more information.

Table 2-3 lists some common NU_Format parameters for the FAT file system.

Table 2-3. NU_Format Parameters for the FAT file system

Parameter	512M FAT16 Drive	720K Drive	20M Drive
oemname User definable up to 8 characters, space padded to the left.			
secpalloc	16	2	4
secreserved	1	1	1
numfats	2	2	2
numroot	0x200	0x70	0x200
mediadesc	0xF8	0xF9	0xF8
bytespsec	512	512	512
secptrk	63	9	17
numhead	16	2	4
Numcyl	1040	80	602

FAT File System Example

```
#include "storage/nu_storage.h"
#include "storage/fat_defs.h"
FMTPARMS fmt;

VOID *x;
.
.
.

/* Format a file system on a ramdrive (assumed * logical device
   name to be rda00). */
fmt.fat_type      = FSFAT_AUTO;
fmt.partdisk      = (UINT8)0;      /* No partition */
fmt.bytespsec     = (UINT16)512;   /* Bytes per sector */
fmt.secpalloc     = (UINT8)2;      /* Sectors per cluster */
fmt.secreserved   = (UINT16)1;     /* Reserved sectors
                                     before FAT */
fmt.numfats       = (UINT8)2;      /* Number of FATs on Disk */
fmt.numroot       = (UINT16)64;    /* Max # root dir. Entries */
fmt.mediadesc     = (UINT8)0xFD;   /* Media descriptor */
fmt.secptrk       = (UINT16)24;    /* Sectors per track */
fmt.numhead       = (UINT16)2;     /* Number of heads */
fmt.numcyl        = (UINT16)2;     /* Number of cylinders */
fmt.totalsec      = (UINT8)96;     /* Total sectors */

strcpy(&fmt.oemname[0], "NUFILE ");

fmt.physical_drive_no = (UINT8)0;
fmt.binary_volume_label = 0x12345678L;
```

```
fmt.text_volume_label[0] = NU_NULL;

x = (VOID*) &fmt;

if (NU_Format("FAT", "rda00", &x) != NU_SUCCESS)
    printf("Format failed \n");
```

SAFE File System Example

```
#include storage/nu_storage.h
#include "storage/wr_api.h"

WR_SAFE_FMT_PARAMS fmt;
VOID *x;
x = (VOID*) &fmt;

if (NU_Get_Format_Info("Safe", "sfb00", &x) != NU_SUCCESS)
    printf("Getting format information failed\n");
else
    if (NU_Format("Safe", "sfb00", &x) != NU_SUCCESS)
        printf("Format failed \n");
```

Related Topics

[NU_Get_Format_Info](#)

[Nucleus FILE System APIs](#)

NU_Free_List

Category: List Services

Source location: *list.c*

This function returns resource previously allocated for linked list.

Note



The list to free must have a pointer to the next element as its first member. This function follows the next pointer, freeing each element, until it finds an NU_NULL next pointer.

Usage

```
STATUS NU_Free_List (VOID **list)
```

Arguments

- list
Pointer to a pointer cast as a VOID **, the head of list to be freed.

Return Values

- NU_SUCCESS
Operation successful.

Example

```
#include "storage/nu_storage.h"

{
    FS_LIST_S  *fs_list, *fs_index;
    DEV_LIST_S *dev_list, *dev_index;
    MNT_LIST_S *mount_list, *mount_index;

    NU_List_File_System(&fs_list);
    NU_List_Device(&dev_list);
    NU_List_Mount(&mount_list);
    .
    .
    .
    /* Return list memory to pool. */
    NU_Free_List((VOID **)&fs_list);
    NU_Free_List((VOID **)&dev_list);
    NU_Free_List((VOID **)&mount_list);
}
```

Related Topics

[Nucleus FILE System APIs](#)

NU_Free_Partition_List

Category: Partition Services

Source location: *fs_part.c*

This function traverses a specified partition list element and frees all subsequent list elements.

Note



The Partition API's are NOT compatible with Flash devices. Additionally, partitions cannot be formatted with the SAFE file system.

Usage

```
VOID NU_Free_Partition_List (PFPART_LIST_S p_list)
```

Arguments

- `p_list`
Pointer to a partition list element.

Example

```
NU_Free_Partition_List(list_head);
```

Related Topics

[NU_List_Partitions](#)

[Nucleus FILE System APIs](#)

NU_FreeSpace

Category: Drive Operations

Source location: *fs_free.c*

This function returns sector and cluster information for a specified drive.

Usage

```
STATUS NU_FreeSpace (CHAR    *path,  
                    UINT8    *secpcluster,  
                    UINT16   *bytepsec,  
                    INT32    *freecluster,  
                    UINT32   *totalcluster)
```

Arguments

- **path**
Specifies the drive letter.
- **secpcluster**
Pointer to the number of sectors per cluster.
- **bytepsec**
Pointer to the number of bytes per sector.
- **freecluster**
Pointer to the number of free clusters.
- **totalcluster**
Pointer to the number of clusters.

Return Values

- **NU_SUCCESS**
Operation successful.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_BADDRIVE**
Invalid drive specified.
- **NUF_NOT_OPENED**
Drive is not opened.

Example

```
#include "storage/nu_storage.h"
```

```
UINT8 secpcluster;  
UINT16 bytepsec;  
UINT32 freecluster, totalcluster, freebytes;  
  
status = NU_FreeSpace("A:", &secpcluster, &bytepsec, &freecluster,  
                      &totalcluster);  
  
freebytes = secpcluster * bytepsec * freecluster;
```

Related Topics

[Nucleus FILE System APIs](#)

NU_Get_Attributes

Category: File and Directory Operations

Source location: *fs_attr.c*

This function gets the attribute byte for a given file.

Usage

```
STATUS NU_Get_Attributes (UINT8 *attr,  
                          CHAR  *name)
```

Arguments

- attr

Pointer to the attribute file byte.

The following file attributes are defined in *pcdisk.h*:

ANORMAL	0x00	/*	Normal	*/
ARDONLY	0x01	/*	Read only	*/
AHIDDEN	0x02	/*	Hidden	*/
ASYSTEM	0x04	/*	System	*/
AVOLUME	0x08	/*	Volume Label	*/
ADIRENT	0x10	/*	Directory	*/
ARCHIVE	0x20	/*	Archive	*/

Note



SAFE file systems do not use the AVOLUME attribute.

- name

The path name of the file to be opened.

Return Values

- NU_SUCCESS
Operation successful.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_ACCES
Cannot get attributes on '.' or '..'.
- NUF_NO_DROBJ
No DROBJ object available.
- NUF_IO_ERROR
Driver I/O error.

- NUF_BADDRIIVE
Invalid drive specified.
- NUF_BADPARM
Wildcard is specified for the name.
- NUF_NOFILE
Specified file is not found.
- NUF_INVNAME
Invalid file name is specified.

Legacy Error Codes

- PENOENT
File or path not found.

Example

```
#include "storage/nu_storage.h"

UINT8 attr;
STATUS status;

status = NU_Get_Attributes(&attr, "FILENAME.EXT");
```

Related Topics

[NU_Set_Attributes](#)

[Nucleus FILE System APIs](#)

NU_Get_Default_Drive (Deprecated)

Caution



This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Drive Operations

Source location: *fs_env.c*

This function gets the current default drive when a path specifier does not contain a drive specifier.

Usage

```
INT16 NU_Get_Default_Drive (VOID)
```

Return Values

- INT16
The current default drive.

Example

```
#include "storage/nu_storage.h"
UINT16 driveno;

if (path[1] != ':')
    driveno = NU_Get_Default_Drive();
else
    driveno = path[0] - 'A';
```

Related Topics

[NU_Set_Default_Drive \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Get_First

Category: Directory Operations

Source location: *fs_dir.c*

This function obtains the first entry in a directory to match a pattern.

Usage

```
STATUS NU_Get_First (DSTAT *statobj,  
                    CHAR  *pattern)
```

Arguments

- **statobj**
Pointer to the [DSTAT](#) data structure, which maintains file/subdirectory information useful in traversing a directory tree.
- **pattern**
A string that represents the first directory entry searched for.

Return Values

- **NU_SUCCESS**
If a match was found.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_BADDRIVE**
Invalid drive specified.
- **NUF_INVAME**
Path or filename includes invalid character.
- **NUF_NOFILE**
The specified pattern not found.
- **NUF_NO_DROBJ**
No DROBJ buffer available.
- **NUF_IO_ERROR**
Driver I/O function routine returned error.

Description

Given a pattern that contains both a path specifier and a search pattern, this fills in the structure at **statobj** with information about the file and set up internal parts of **statobj** to supply appropriate information for calls to [NU_Get_Next](#).

Note

In SAFE file systems, [NU_Get_First](#) can be called with the “*.*” wild card.

Example

```
/* Examples of patterns are:
"D:\\USR\\RELEASE\\NETWORK\\*.C"
"D:\\USR\\BIN\\UU\\*.*)"
"D:\\MEMO\\?.*)"
"D:\\*.*)" */

#include "storage/nu_storage.h"
DSTAT statobj;
CHAR pattern[] = "*.*)" ;

/* A simple directory list subroutine. */
if (NU_Get_First(&statobj, pattern) == NU_SUCCESS)
{
    while(1)
    {
        printf("%s",statobj.sfname);
        if ( statobj.fext[0] != ' ' )
            printf("%.s ", statobj.fext );
        else
            printf(" ");
        if ( statobj.fattribute & ADIRENT)
        {
            printf(" <DIR>");
        }
        else if (statobj.fattribute & AVOLUME)
        {
            printf(" <VOL>");
        }
        else
        {
            printf("%6d", statobj.fsize );
        }
        printf(" %d-%d-%d %d:%d:%d ",
            ((statobj.fupdate >> 9 ) & 0x007f) + 1980,
            ((statobj.fupdate >> 5 ) & 0x000f),
            (statobj.fupdate & 0x001f),
            ((statobj.fuptime >> 11 ) & 0x001f ),
            ((statobj.fuptime >> 5 ) & 0x003f ),
            ((statobj.fuptime << 1 ) & 0x003f ));
        printf(" %s\n",statobj.lfname );

        if (NU_Get_Next(&statobj) != NU_SUCCESS)
        {
            NU_Done(&statobj);
            break;
        }
    }
}
```

Related Topics

[NU_Done](#)

[NU_Get_Next](#)

[Nucleus FILE System APIs](#)

NU_Get_Format_Info

Category: Format Services

Source location: *fs_frmt.c*

This function returns file system specific format parameters for the given device and file system type.

Usage

```
STATUS NU_Get_Format_Info (CHAR *fs_name,  
                           CHAR *dev_name,  
                           VOID **params)
```

Arguments

- **fs_name**
String identifying registered file system.
- **dev_name**
String identifying created device.
- **params**
Pointer to location for file system arguments.

Note



In SAFE file systems, `NU_Get_Format_Info` sets `**params` to `NU_NULL` because there are no file system-specific format parameters required with Safe. To ensure future compatibility you must format the drive following the example provided by [NU_Format](#).

Return Values

- **NU_SUCCESS**
Operation successful.
- **< 0**
File system specific error occurred.

Fat File System Example

```
#include "storage/nu_storage.h"  
  
FMT_PARAMS *params;  
  
if (NU_Get_Format_Info("FAT", "hda00", &params) != NU_SUCCESS)  
    printf("Format info returned an error");
```

SAFE File System Example:

```
#include storage/nu_storage.h  
#include "storage/wr_api.h"
```

```
WR_SAFE_FMT_PARAMS fmt;
VOID *x;
x = (VOID*) &fmt;

if (NU_Get_Format_Info("Safe","sfb00",&x) != NU_SUCCESS)
    printf("Getting format information failed\n");
```

Related Topics

[NU_Format](#)

[Nucleus FILE System APIs](#)

NU_Get_Next

Category: Directory Operations

Source location: *fs_dir.c*

This function gets the next entry in a directory that matches a pattern.

Usage

```
STATUS NU_Get_Next (DSTAT *statobj)
```

Arguments

- statobj
Pointer to the [DSTAT](#) data structure, which maintains file/subdirectory information useful in traversing a directory tree.

Return Values

- NU_SUCCESS
If a match was found.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_BADPARAM
Invalid parameter specified.
- NUF_NOFILE
The specified pattern not found.
- NUF_IO_ERROR
Driver I/O function routine returned error.

Description

Given a pointer to a [DSTAT](#) structure that is set up by a call to [NU_Get_First](#), it searches for the next match of the original pattern in the original path. It returns NU_SUCCESS if a match was found and updates STATOBJ for subsequent calls to NU_Get_Next.

Example

See [NU_Get_First](#).

Related Topics

[NU_Done](#)

[NU_Get_First](#)

[Nucleus FILE System APIs](#)

NU_Get_Partition_Info

Category: Partition Services

Source location: *fs_part.c*

Given a valid device name, this function returns the partition type (logical, primary, or extended); size in MiB; offset in MiB; and the partition identifier as written in the partition table.

Note



The Partition API's are NOT compatible with Flash devices. Additionally, partitions cannot be formatted with the SAFE file system.

Usage

```
STATUS NU_Get_Partition_Info (CHAR    *log_dev_name,  
                             UINT16  *ret_part_type,  
                             UINT32  *ret_size,  
                             UINT32  *ret_offset,  
                             UINT8   *ret_part_id)
```

Arguments

- `log_dev_name`
Unique string describing the logical device.
- `ret_part_type`
Pointer to the value of the partition type.
- `ret_size`
Pointer to the value of the partition size.
- `ret_offset`
Pointer to the value of the partition offset.
- `ret_part_id`
Pointer to the value of the partition type ID.

Return Values

- `NU_SUCCESS`
Operation successful.
- `NUF_INVALID_DEVNAME`
Matching device name was not found.

Example

```
#include "storage/nu_storage.h"

UINT8  part_id;
UINT16 part_type;
UINT32 size, offset;

if (NU_Get_Partition_Info("hda00", &part_type, &size, &offset, &part_id) <
0)
    printf("Error getting partition information");
```

Related Topics

[NU_Create_Partition](#)

[NU_Delete_Partition](#)

[NU_List_Partitions](#)

[Nucleus FILE System APIs](#)

NU_List_Device

Category: List Services

Source location: *list.c*

This function creates a linked list of all registered storage devices and places the address of the first element of the list in the caller's `dev_list` parameter.

Usage

```
STATUS NU_List_Device (DEV_LIST_S **dev_list_head)
```

Arguments

- `dev_list_head`
Pointer to a pointer to `DEV_LIST_S` structure where the list head pointer will be returned.

Return Values

- `NU_SUCCESS`
Operation successful.
- `NU_NO_MEMORY`
No memory available for the list.

Description

There is no guarantee that another file system user has not changed the object data.

The list is a chain of structures defined in *list.h* as:

```
typedef struct device_list_entry
{
    struct device_list_entry *next;
    struct device_list_entry *previous;
    const CHAR                *dev_name;
}DEV_LIST_S;
```

Example

```
#include "storage/nu_storage.h"

DEV_LIST_S *dev_list, *dev_index;

NU_List_Device(&dev_list);
printf("\n\nRegistered Devices:\n\n", &port);
dev_index = dev_list;

while (dev_index)
{
    printf("\t", &port);
    printf((CHAR *)dev_index->dev_name, &port);
    printf("\n\n", &port);
    dev_index = dev_index->next;
}
```



```
}  
NU_Free_List((VOID **)&dev_list);
```

Related Topics

[NU_Free_List](#)

[Nucleus FILE System APIs](#)

NU_List_File_System

Category: List Services

Source location: *list.c*

This function creates a linked list of all registered file systems and places the address of the first element of the list in the caller's `fs_list` parameter.

Usage

```
STATUS NU_List_File_System (FS_LIST_S **fs_list_head)
```

Arguments

- `fs_list_head`
Pointer to a pointer to an `FS_LIST_S` structure where the list head pointer is returned.

Return Values

- `NU_SUCCESS`
Operation successful.
- `NU_NO_MEMORY`
No memory available for the list.

Description

There is no guarantee that another file system user has not changed the object data.

The list is a chain of structures defined in *list.h* as:

```
typedef struct file_system_list_entry
{
    struct file_system_list_entry *next;
    struct file_system_list_entry *previous;
    const CHAR                    *fs_name;
}FS_LIST_S;
```

Example

```
#include "storage/nu_storage.h"

FS_LIST_S *fs_list, *fs_index;
NU_List_File_System(&fs_list);

printf("\n\rRegistered File Systems:\n\r", &port);
fs_index = fs_list;

while (fs_index)
{
    printf("\t", &port);
    printf((CHAR *)fs_index->fs_name, &port);
    printf("\n\r", &port);
    fs_index = fs_index->next;
}
```

```
NU_Free_List((VOID **)&fs_list);
```

Related Topics

[NU_Free_List](#)

[Nucleus FILE System APIs](#)

NU_List_Mount

Category: List Services

Source location: *list.c*

This function creates a linked list of all mounted storage and places the address of the first element of the list in the caller's `dev_list` parameter.

Usage

```
STATUS NU_List_Mount (MNT_LIST_S **mount_list_head)
```

Arguments

- `mount_list_head`
Pointer to a pointer to `MNT_LIST_S` structure where the list head pointer is returned.

Return Values

- `NU_SUCCESS`
Operation successful.
- `NU_NO_MEMORY`
No memory available for the list.

Description

There is no guarantee that another file system user has not changed the object data.

The list is a chain of structures defined in *list.h* as:

```
typedef struct mount_list_entry
{
    struct mount_list_entry *next;
    struct mount_list_entry *previous;
    const CHAR              *fs_name;
    const CHAR              *dev_name;
    const CHAR              *mnt_name;
    const VOID              *config;
}MNT_LIST_S;
```

Example

```
#include "storage/nu_storage.h"

MNT_LIST_S *mount_list, *mount_index;

NU_List_Mount(&mount_list);
printf("\n\nRegistered Mount Points:\n\n", &port);
mount_index = mount_list;

while (mount_index)
{
    printf("\t", &port);
    printf((CHAR *)mount_index->mnt_name, &port);
```

```
    printf(":\t", &port);  
    printf((CHAR *)mount_index->dev_name, &port);  
    printf("\t", &port);  
    printf((CHAR *)mount_index->fs_name, &port);  
    printf("\n\r", &port);  
    mount_index = mount_index->next;  
}  
printf("\n\n\r", &port);  
NU\_Free\_List((VOID **) &mount_list);
```

Related Topics

[NU_Free_List](#)

[Nucleus FILE System APIs](#)

NU_List_Partitions

Category: Partition Services

Source location: *fs_part.c*

This function returns an ordered linked list of partitions existing on a physical disk.

Usage

```
STATUS NU_List_Partitions (CHAR          *dev_name,  
                          PFPART_LIST_S *ret_list)
```

Arguments

- `dev_name`
Unique string describing the partition to be removed.
- `ret_list`
Pointer to first partition list element.

Return Values

- `NUF_BADPARAM`
Invalid parameter specified.
- `NUF_INVALID_DEVNAME`
Matching device name was not found.
- `NUF_INTERNAL`
Nucleus FILE internal error.
- `NUF_NO_MEMORY`
Cannot allocate internal buffer.
- `NU_SUCCESS`
List is returned.

Description

The list that this function returns is ordered by offset. The clean-up routine, [NU_Free_Partition_List](#), must be called after the list is no longer needed in order to free memory. `ret_list` is set to `NU_NULL` on error and any memory used to allocate the list is freed.

Note



The Partition API's are NOT compatible with Flash devices. Additionally, partitions cannot be formatted with the SAFE file system.

Example

```
#include "storage/nu_storage.h"

PFPART_LIST_S list_head;

if (NU_List_Partitions("hda", &list_head) < 0)
    printf("Error listing partition");
```

Related Topics

[NU_Create_Partition](#)

[NU_Delete_Partition](#)

[NU_Free_Partition_List](#)

[NU_Get_Partition_Info](#)

[Nucleus FILE System APIs](#)

NU_Make_Dir

Category: Directory Operations

Source location: *fs_dir.c*

This function creates a subdirectory in the path specified by name. It fails if a file or directory of the same name already exists or if the path is not found.

Usage

```
STATUS NU_Make_Dir (CHAR *name)
```

Arguments

- name
Directory name to be created.

Return Values

- NU_SUCCESS
Directory created successfully.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_BADDRIIVE
Invalid drive specified.
- NUF_EXIST
The directory already exists.
- NUF_NOSPC
No space to create directory in this disk.
- NUF_LONGPATH
The path or directory name is too long.
- NUF_INVNAME
The path or filename has an invalid character.
- NUF_NODROBJ
No DROBJ buffer available.
- NUF_IO_ERROR
Driver I/O function routine returned error.

Legacy Error Codes

- PENOENT
File or path not found.

- **PEEXIST**
Exclusive access requested but file already exists.
- **PENOSPC**
I/O error.

Example

```
#include "storage/nu_storage.h"

if (NU_Make_Dir("SUBDIR2") != NU_SUCCESS)
    printf("Can't create subdirectory\n");
```

Related Topics

[NU_Remove_Dir](#)

[Nucleus FILE System APIs](#)

NU_Mount_File_System

Category: Core Services

Source location: *fsc.c*

This function creates an instance of the file system using the given device at the mount point. The file system must be previously registered. The device to use must be previously created. The mount point given must be available, not currently in use.

Usage

```
STATUS NU_Mount_File_System (CHAR *fs_name,  
                             CHAR *mount_point,  
                             CHAR *dev_name,  
                             VOID *config)
```

Arguments

- **fs_name**
Name of previously registered file system type to use when operating on the mount point.
- **mount_point**
Drive identifier to locate an instance of mounted device (A,B,...).
- **dev_name**
Name of the logical device to associate with mount point.
- **config**
File system instance specific parameter passed to the fs_mount file system operation.

Return Values

- **NU_SUCCESS**
Disk was mounted.
- **NUF_FS_NOT_FOUND**
File system name was unmatched.
- **NUF_BADPARAM**
Invalid device.
- **NUF_MOUNT_NOT_AVAILABLE**
Mount location is currently in use.
- **NUF_MOUNT_TABLE_FULL**
The mount table is full. Size is determine by the define MTE_MAX_TABLE_SIZE.

Example

```
#include "storage/nu_storage.h"  
  
STATUS status;
```

```
/* Assumes MYFS has been registered, mount point A is available, and  
   the device rda01 is available */  
status = NU_Mount_File_System ("MYFS", "A", "rda01", NU_NULL);
```

Related Topics

[NU_Unmount_File_System](#)

[Nucleus FILE System APIs](#)

NU_Open

Category: Simple File Operations

Source location: *fs_fl.c*

This function opens a specified file for access as specified in flag. If creating a file, use the mode argument to set the access permissions on the file.

Usage

```
INT NU_Open (CHAR    *name,  
             UINT16  flag,  
             UINT16  mode)
```

Arguments

- name

The path name of the file to be opened.

- flag

The flags accepted by NU_Open can be broken into three different categories: File Type, Open permissions, and Action Flags.

File Type

PO_TEXT

Ignored. All files are treated as binary files.

PO_BINARY

Ignored. All files are treated as binary.

Open permissions

PO_RDONLY

Open file for read-only. The same file can be opened with multiple PO_RDONLY open permissions.

PO_WRONLY

Open for write-only.

PO_RDWR

Read/write access is allowed.

Note



When using Safe file systems, if a file is opened with PO_WRONLY or PO_RDWR in its flag combination, no other user can open this file until it is closed. If a file is opened with PO_APPEND or PO_RDWR, any number of opens with PO_RDONLY are allowed at the same time.

Action Flags

PO_CREAT

This flag creates a file if it does not exist. If the file exists the PO_CREAT is ignored. This flag must be used if you want to set the mode value for a file.

PO_EXCL

If this flag is used in conjunction with PO_CREAT and the file already exists, NUF_EXIST is returned. Otherwise, the file is created. This flag can only be used with PO_CREAT.

PO_TRUNC

Supplying this flag truncates a file if it exists; if it does not exist the flag is ignored. This flag can only be used with PO_CREAT.

PO_APPEND

This flag opens a file up for appending. When performing a write operation on a file opened for PO_APPEND, the file pointer is always be repositioned to the end before executing the write. Truncate operations cannot be performed on a file opened for appending.

- mode

The mode parameter can be used to set the read-only file attribute to one of the following values to match the description enumeration in “flag”:

PS_IREAD

Read permitted. (Always true)

PS_IWRITE

Write permitted.

These can only be set when creating the file. Therefore, PO_CREAT must be used in your flag combination.

Creating a file and only supplying PS_IREAD as the mode parameter will set the file to read-only and will prevent other users from opening the file with PO_WRONLY or PO_RDWR. No further writes are allowed on the file until the file is closed.

To change a read-only file to writable, use [NU_Get_Attributes](#) and [NU_Set_Attributes](#).

Return Values

- INT
A non-negative integer to be used as a file descriptor for calling read/write/seek/close.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_BADDRIVE
Invalid drive specified.

- **NUF_NOSPC**
No space to create file in this disk.
- **NUF_LONGPATH**
The path or directory name is too long.
- **NUF_INVNAME**
The path or filename has an invalid character.
- **NUF_PEMFILE**
No file descriptors available (too many files open).
- **NUF_INVPARM**
Invalid flag/mode is specified.
- **NUF_INVPARCMB**
Invalid flag/mode combination.
- **NUF_SHARE**
The access conflict from multiple tasks to a specific file.
- **NUF_ACCES**
Attempt to open a read only file or a special (directory) file.
- **NUF_EXIST**
The directory already exists when PO_EXCL flag is set.
- **NUF_IO_ERROR**
Driver I/O function routine returned error.
- **NUF_EXIST**
Exclusive access requested but file already exists.
- **NUF_NOFILE**
File not found.
- **NUF_NO_BLOCK**
No block buffer available.
- **NUF_NO_DROBJ**
No DROBJ buffer available.
- **NUF_NO_FINODE**
No FINODE buffer available.
- **NUF_INTERNAL**
Nucleus FILE internal error.

- **NUF_INVPARCMB**

Invalid flag combination. This can result from any of the following flag combinations:

PO_TRUNC is specified and neither PO_WRONLY nor PO_RDWR.

PO_APPEND is specified and neither PO_WRONLY nor PO_RDWR.

PO_EXCL is specified and not PO_CREATE.

PO_WRONLY and PO_RDWR are both specified.

PO_APPEND and PO_TRUNC are both specified.

- **NUF_NOT_OPENED**

The disk is not opened yet.

- **NUF_NO_DISK**

The disk is removed.

- **NUF_INVNAME**

The path or filename has an invalid character.

Legacy Error Codes

- **PENOENT**

File or path not found.

- **PENOSPC**

Create failed.

- **PEMFILE**

No file descriptors available. (Too many files open.)

- **PEEXIST**

Exclusive access requested but file already exists.

- **PEACCES**

Attempt to open a read-only file or special (directory) file.

- **PESHARE**

Already open in exclusive mode.

Example

```
#include "storage/nu_storage.h"
INT fd;
if (fd = NU_Open("\\USR\\MYFILE",
    (PO_CREAT|PO_EXCL|PO_WRONLY), PS_IWRITE) < 0))
    printf("Can't create file error:%d\n",fd);
```

Related Topics

[NU_Close](#)

[Nucleus FILE System APIs](#)

NU_Open_Disk (Deprecated)

Caution



This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Drive Operations

Source location: *fs_dsk.c*

This function opens a specified disk for the file system user. This service assigns your default directory for the newly opened disk to the disk root. If a default is not set, this becomes your default disk. This function is only required for tasks that have called [NU_Become_File_User \(Deprecated\)](#).

Usage

```
STATUS NU_Open_Disk (CHAR *path)
```

Arguments

- path
Specifies the drive letter for the disk to be opened.

Return Values

- NU_SUCCESS
The disk was successfully opened.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_BADDRIVE
Invalid drive number is specified.
- NUF_FATCORE
FAT cache table too small.
- NUF_NO_PARTITION
No partition on disk.
- NUF_FORMAT
Not formatted on this disk.
- NUF_NO_MEMORY
Cannot allocate internal buffer.
- NUF_IO_ERROR
Driver I/O function routine returned error.

- NUF_NO_DISK
Disk is removed.
- NUF_INTERNAL
Nucleus FILE internal error.
- NUF_NO_ACTION
The disk is already open.

Example

```
#include "storage/nu_storage.h"

STATUS status;
status = NU_Open_Disk("A:");
```

Related Topics

[NU_Close_Disk \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Read

Category: Simple File Operations

Source location: *fs_fl.c*

This function reads count bytes from the current file pointer of the file, puts them in the buffer, and updates the file pointer.

Usage

```
INT32 NU_Read (INT    fd,  
               CHAR   *buf,  
               INT32  count)
```

Arguments

- **fd**
File descriptor for the file to be read.
- **buf**
Buffer to be filled by the read.
- **count**
Number of bytes to be read.

Return Values

- **INT32**
A non-negative integer indicating the number of bytes read.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_BADFILE**
Invalid file descriptor.
- **NUF_IO_ERROR**
Driver I/O function routine returned error.
- **NUF_BADPARAM**
Invalid parameter specified.
- **NUF_ACCES**
Opened for write only mode.
- **NUF_NO_DISK**
Disk is removed.
- **NUF_INTERNAL**
Nucleus FILE internal error.

Legacy Error Codes

- **PEBADF**
Invalid file descriptor.

Example

```
#include "storage/nu_storage.h"

INT fd;
STATUS status;
CHAR buff[512];

if (fd = NU\_Open("\\USR\\MYFILE.TXT",
    (PO_TEXT|PO_RDWR), PS_IREAD) >= 0)
{
    status = NU\_Read(fd, buff, 512);
}
```

Related Topics

[NU_Open](#)

[Nucleus FILE System APIs](#)

NU_Register_File_System

Category: File System Services

Source location: *fsc.c*

This function registers a type of file system with Nucleus FILE.

Usage

```
STATUS NU_Register_File_System (CHAR *name,  
                               FS_S *fs)
```

Arguments

- **name**
Character pointer to an array of max size FSC_MAX_FS_NAME. This string identifies a type of file system.
- **fs**
Pointer to a structure type [FS_S](#) containing operations specific to the file system type identified by name.

Return Values

- **NU_SUCCESS**
Registration request completed successfully.
- **NUF_DUPLICATE_FSNAME**
File system name was not unique.
- **NUF_FS_TABLE_FULL**
File system table is full. Table size is defined by FSC_MAX_TABLE_SIZE.
- **NUF_BADPARAM**
Required parameter was not given.
- **Other**
Other values may be returned by the file system specific initialization routine. Refer to the file system specific documentation for the meaning of these return values.

Description

The operations passed in with the structure fs are used to manipulate file system objects of this type. Once successfully registered, a file system may be used along with a device to create a usable storage location. The name given as a parameter must be unique.

Example

```
#include "storage/nu_storage.h"  
  
FS_S fs = my_fs; /* Assumes my_fs is defined */  
status = NU_Register_File_System("MYFS", &fs);
```

Related Topics

[NU_Unregister_File_System](#)

[Nucleus FILE System APIs](#)

NU_Release_File_User (Deprecated)

Caution



This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Task Operations

Source location: *fs_users.c*

This function frees up a user structure that will be used by other tasks. Subsequent API calls fail if this routine is called. Use this function when a task is finished with the file system.

Usage

```
STATUS NU_Release_File_User (VOID)
```

Return Values

- **NU_SUCCESS**
Abort request completed successfully.
- **NUF_BAD_USER**
Task not registered as a file user.

Example

```
#include "storage/nu_storage.h"  
/* Task is done using the file system. */  
NU_Release_File_User();
```

Related Topics

[NU_Become_File_User \(Deprecated\)](#)

[NU_Check_File_User \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Remove_Dir

Category: Directory Operations

Source location: *fs_dir.c*

This function deletes the specified directory. This service does not support wildcards.

Usage

```
STATUS NU_Remove_Dir (CHAR *name)
```

Arguments

- **name**
A string representing the pathname of the directory to be removed. Name should be either the absolute path or a file that is in the current working directory.

Return Values

- **NU_SUCCESS**
The directory was successfully removed.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_NOFILE**
The directory name is not found.
- **NUF_NOEMPTY**
The directory is not empty.
- **NUF_ACCES**
The directory has at least one of the following attributes: RDONLY, HIDDEN, SYSTEM or VOLUME.
- **NUF_NO_DROBJ**
No DROBJ buffer available.
- **NUF_IO_ERROR**
Driver I/O function routine returned error.
- **NUF_SHARE**
The directory is opened.
- **NUF_BADDRIVE**
Invalid drive specified.

Legacy Error Codes

- PENOENT
File or path not found.
- PENOSPC
Remove failed.
- PEACCES
Directory is read-only.

Example

```
#include "storage/nu_storage.h"

if (NU_Remove_Dir("D:\\USR\\TEMP") != NU_SUCCESS )
    printf("Can't delete directory\n");
```

Related Topics

[NU_Make_Dir](#)

[Nucleus FILE System APIs](#)

NU_Rename

Category: File and Directory Operations

Source location: *fs_fl.c*

This function renames the file in path (name) to newname.

Usage

```
STATUS NU_Rename (CHAR *name,  
                  CHAR *newname)
```

Arguments

- **name**
The pathname of the file to be renamed. Name should be either the absolute path or a file that is in the current working directory.
- **newname**
The new pathname for the file being renamed. This should be either the absolute path or a file that is now in the current working directory.

Return Values

- **NU_SUCCESS**
The file was renamed.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_ACCES**
The file or directory has VOLUME, SYSTEM, or HIDDEN attribute.
- **NUF_BADDRIIVE**
Invalid drive specified.
- **NUF_NOFILE**
The name of file is not found.
- **NUF_NODROBJ**
No DROBJ buffer available.
- **NUF_IO_ERROR**
Driver I/O function routine returned error.

Legacy Error Codes

- **PENOENT**
File or path not found.

- **PEEXIST**
File or directory already exists.
- **PEACCES**
File or directory is read-only.

Description

It fails if the name is invalid, newname already exists, or if the path is not found. Does not test if the name is a simple file. It is possible to rename directories and files within the same mount point assuming file system-specific support.

Example

```
#include "storage/nu_storage.h"

if (NU_Rename("\\USR\\TXT\\LETTER.TXT", "LETTER.NEW") != NU_SUCCESS)
    printf("Can't rename LETTER.TXT");
```

Related Topics

[Nucleus FILE System APIs](#)

NU_Seek

Category: Simple File Operations

Source location: *fs_fl.c*

This function moves the file pointer offset bytes from the origin specified.

Usage

```
INT32 NU_Seek (INT    fd,  
               INT32  offset,  
               INT16  origin)
```

Arguments

- **fd**
File descriptor for the file, which is having its pointer moved.
- **offset**
Number of bytes the pointer is to be moved.
- **origin**
Sets the location that the offset will be counted from. This can be one of the following settings:

PSEEK_SET

Offset from beginning of file.

PSEEK_CUR

Offset from current file pointer.

PSEEK_END

Offset from end of file.

Caution



Attempting to seek beyond end of file puts the file pointer one byte past eof.

Return Values

- **INT32**
A non-negative integer indicating the current file pointer.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_BADPARM**
Invalid parameter specified.
- **NUF_IO_ERROR**
Driver I/O function routine returned error.

- NUF_INTERNAL
Internal error.
- NUF_BADFILE
File descriptor invalid.
- NUF_NO_DISK
Disk is removed.

Legacy Error Codes

- PEBADF
File or path not found.
- PEINVAL
Attempt to seek to a negative file pointer.

Example

```
#include "storage/nu_storage.h"

status = NU_Open("\\USR\\MYFILE.TXT", (PO_TEXT|PO_RDWR), PS_IREAD) >= 0))

if (NU_Seek(fd, (recsize * recno), PSEEK_SET) != (recsize*recno))
    printf("Can't find record %d\n", recno);
```

Related Topics

[NU_Open](#)

[Nucleus FILE System APIs](#)

NU_Set_Attributes

Category: File and Directory Operations

Source location: *fs_attr.c*

This function sets the attributes byte for a given file. If you want to set or clear individual attributes, call [NU_Get_Attributes](#) first and then change the bit you want before calling this routine.

Usage

```
STATUS NU_Set_Attributes (CHAR   *name,  
                        UINT8 newattr)
```

Arguments

- **name**
The path name of the file to be opened.

- **newattr**
Return variable for the attribute byte. The following constants are defined in *pcdisk.h*:

ANORMAL	0x00	/* Normal	*/
ARDONLY	0x01	/* Read only	*/
AHIDDEN	0x02	/* Hidden	*/
ASYSTEM	0x04	/* System	*/
AVOLUME	0x08	/* Volume Label	*/
ADIRENT	0x10	/* Directory	*/
ARCHIVE	0x20	/* Archive	*/

Note



SAFE file systems do not use the AVOLUME attribute.

Return Values

- **NU_SUCCESS**
Operation successful.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_ACCES**
Cannot set attributes on VOLLABEL, '.', or '..'.
- **NUF_BADDRIVE**
Invalid drive specified.
- **NUF_NOFILE**
Specified file not found.

- NUF_NO_DROBJ
No DROBJ object available.
- NUF_IO_ERROR
Driver I/O error.
- NUF_BADDRIVE
Invalid drive specified.
- NUF_INVNAME
Invalid file name is specified.

Legacy Error Codes

- PENOENT
File or path not found.
- PEACCES
Access error.
- PENOSPC
I/O error.

Example

```
#include "storage/nu_storage.h"

UINT8 newattr;

newattr = ( ANORMAL | ARCHIVE )
status = NU_Set_Attributes("FILENAME.EXT", newattr);
```

Related Topics

[NU_Get_Attributes](#)

[Nucleus FILE System APIs](#)

NU_Set_Current_Dir (Deprecated)

Caution



This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Drive Operations

Source location: *fs_env.c*

This interface finds the path specified. If the path is a subdirectory, that subdirectory will be the current working directory for the drive. If this API is used by a task that has not called [NU_Become_File_User \(Deprecated\)](#), then the global environment is modified.

Note



SAFE file system's root directory does not contain a '.' or '..' entry, as opposed to FAT file systems.

Note



Attempting to set the current working directory to the root's parent results in the current working directory being set to root.

Usage

```
STATUS NU_Set_Current_Dir (CHAR *path)
```

Arguments

- path
A string representing the path name of the directory that is to become the current working directory.

Return Values

- NU_SUCCESS
The current working directory was changed.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_BADDRIVE
Invalid drive specified.
- NUF_ACCES
The path was not a directory
- NUF_LONGPATH
The path or directory name is too long.

- **NUF_NOFILE**
The directory was not found.
- **NUF_NO_DROBJ**
No DROBJ buffer available.
- **NUF_NO_ACTION**
The disk is already open.
- **NUF_NO_BLOCK**
No block buffer available.
- **NUF_NO_FINODE**
No FINODE buffer available.
- **NUF_IO_ERROR**
Driver I/O function routine returned error.
- **NUF_INTERNAL**
Nucleus FILE internal error.

Example

```
if (NU_Set_Current_Dir("D:\\USR\\DATA\\FINANCE") != NU_SUCCESS)
    printf("Can't change working directory\n");
```

Related Topics

[NU_Current_Dir \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Set_Default_Drive (Deprecated)

 **Caution** This API has been deprecated. The file user and environment setting API's are no longer required for a task to access the drives.

Category: Drive Operations

Source location: *fs_env.c*

This function sets the current default drive used when a path specifier does not contain a drive specifier. If this API is used by a task that has not called [NU_Become_File_User \(Deprecated\)](#), then the global environment is modified.

Usage

```
STATUS NU_Set_Default_Drive (UINT16 driveno)
```

Arguments

- driveno

The drive number to become the default drive. This is zero (drive A:) by default.

Return Values

- NU_SUCCESS
The current default drive was changed.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_BADDRIIVE
The drive number is out of range.
- NUF_NOT_OPENED
The disk is not opened yet.
- NUF_NO_DISK
Disk is removed.

Example

```
/* Set default to C: */  
if (NU_Set_Default_Drive(2) != NU_SUCCESS)  
    printf("Can't change default drive\n");
```

Related Topics

[NU_Get_Default_Drive \(Deprecated\)](#)

[Nucleus FILE System APIs](#)

NU_Storage_Device_Wait

Category: Drive Operations

Source location: *dev_wait.c*

This function allows an application to wait for a storage device to become available.

Usage

```
STATUS NU_Storage_Device_Wait (CHAR      *mount_name,  
                               UNSIGNED  suspend)
```

Arguments

- `mount_name`

Drive letter or NULL. If NULL the function will return on any device mount.

- `suspend`

The following is a summary of the possible values for the suspend parameter.

`NU_NO_SUSPEND`

The service returns immediately regardless of whether or not the request can be satisfied. This is the only valid option if the service is called from a non-task thread.

`NU_SUSPEND`

The calling task is suspended until the event flag combination is available.

`timeout value`

(1 – 4,294,967,293). The calling task is suspended until the event flag combination is available or until the specified number of ticks has expired.

Return Values

- `NU_SUCCESS`

The current default drive was changed.

- Error

Indicates one of the called functions failed.

Example

```
STATUS status;  
  
/* Wait indefinitely for C drive to become mounted */  
status = NU_Storage_Device_Wait("C", NU_SUSPEND);  
  
if (status == NU_SUCCESS)  
    printf("Drive C: is mounted\n");
```

Related Topics

[Nucleus FILE System APIs](#)

NU_Truncate

Category: Simple File Operations

Source location: *fs_fl.c*

This interface moves the file pointer offset bytes from the beginning of the file and truncates the file beyond that point by adjusting the file size and freeing the cluster chain past the file pointer.

Usage

```
STATUS NU_Truncate (INT    fd,  
                   INT32  offset)
```

Arguments

- **fd**
File descriptor for the file to be truncated.
- **offset**
The place within the file at which the truncation is to begin.

Return Values

- **NU_SUCCESS**
Truncate successful.
- **NUF_BAD_USER**
Task not registered as a file user.
- **NUF_BADPARM**
Invalid parameter specified.
- **NUF_BADFILE**
File descriptor invalid.
- **NUF_ACCES**
Attempt to write to opened file with read-only mode.
- **NUF_SHARE**
Cannot truncate a file opened by more than one handle.
- **NUF_IO_ERROR**
Driver I/O function routine returned error.
- **NUF_INTERNAL**
Internal error.
- **NUF_NO_DISK**
Disk is removed.

- NUF_NO_BLOCK
No block buffer available.

Example

```
#include "storage/nu_storage.h"
INT fd;

fd = NU\_Open("\\USR\\MYFILE", (PO_TEXT|PO_RDWR), PS_IWRITE) < 0);
status = NU_Truncate(fd, 64);
```

Related Topics

[NU_Open](#)

[Nucleus FILE System APIs](#)

NU_Unmount_File_System

Category: Core Services

Source location: *fsc.c*

This function removes a mount point that was previously mounted.

Usage

```
STATUS NU_Unmount_File_System (CHAR *mount_point)
```

Arguments

- `mount_point`
Mount point to remove (A, B, ...).

Return Values

- `NU_SUCCESS`
Operation successful.
- `NUF_MOUNT_NOT_AVAILABLE`
Invalid location given.
- `NUF_BADPARAM`
Invalid pointer or parameter.
- `NUF_IN_USE`
Disk is currently in use.

Description

The mount point must no longer be open. This requires all users that have opened the disk with the service [NU_Open_Disk \(Deprecated\)](#) to have also called [NU_Close_Disk \(Deprecated\)](#). Once the service completes successfully the mount point is available to be reused.

Example

```
#include "storage/nu_storage.h"

if (NU_Unmount_File_System("A") != NU_SUCCESS)
    printf("Mount removal failed.");
```

Related Topics

[NU_Mount_File_System](#)

[Nucleus FILE System APIs](#)

NU_Unregister_File_System

Category: File System Services

Source location: *fsc.c*

This function removes a previously registered file system type for the system.

Usage

```
STATUS NU_Unregister_File_System (CHAR *name)
```

Arguments

- **name**
File system to remove.

Return Values

- **NU_SUCCESS**
File system was successfully removed.
- **NUF_BADPARAM**
Invalid input parameter. File system with matching name was not found.
- **NUF_IN_USE**
File system type is currently mounted.

Description

The file system type must not be in use or mounted at the time service is called. When the file system is successfully unregistered, file systems of this type are no longer available for mounting.

Example

```
#include "storage/nu_storage.h"

if (NU_Unregister_File_System("MYFS") != NU_SUCCESS)
    printf("Unregister failed.");
```

Related Topics

[NU_Register_File_System](#)

[Nucleus FILE System APIs](#)

NU_Utime

Category: Simple File Operations

Source location: *fs_dir.c*

Given a pointer to a [DSTAT](#) structure, this function sets the DSTAT members that correspond to the parameters passed in.

Note



This function is not supported by SAFE file systems.

Usage

```
STATUS NU_Utime (DSTAT *statobj,  
                 UINT16 access_date,  
                 UINT16 access_time,  
                 UINT16 update_date,  
                 UINT16 update_time,  
                 UINT16 create_date,  
                 UINT16 create_time)
```

Arguments

- `statobj`
Caller's buffer's values to set.
- `access_date`
Access date to be set to `facdate`.
- `access_time`
Access time to be set to `facctime`.

Note



FAT file systems do not support access time. The `access_time` parameter should be set to `0xFFFF` when using this API.

- `update_date`
Update date to be set to `fupdate`.
- `update_time`
Update time to be set to `fuptime`.
- `create_date`
Creation date to be set to `fcrdate`.
- `create_time`
Creation time to be set to `fcrttime`.

Return Values

- **NU_SUCCESS**
The statobj members where set.
- **NUF_BADPARM**
Invalid parameter specified.
- **NUF_BADDRIVE**
Invalid drive specified.
- **NUF_ACCES**
This file has at least one of the following attributes: RDONLY, HIDDEN, SYSTEM, VOLUME
- **NU_NO_MEMORY**
No memory available for the list.

Description

For example, statobj's facdate member is set to access_date, as long as access_date is valid. If a parameter is 0xFFFF, it is set to the current system time or date depending on the parameter. The time and date are both stored according to the FAT32 File System Specification, which means for the date bits 0-4 represent the day of the month, bits 5-8 represent the month, and bits 9-15 represent the year relative to 1980. The time bits 0-4 represent a 2-sec count, bits 5-10 represent the minutes, and bits 11-15 represent the hours. This function allows you to manually change the access, creation, and update times and dates.

Example

```
#include "storage/nu_storage.h"

UINT16 time = 314; /* (10:45:32) */
UINT16 date = 21936; /* (Sept. 26 1980) */
DSTAT statobj;
STATUS ret_val;

ret_val = NU_Get_First(&statobj, "foo.txt");

/* Alter the creation time and date only. */
ret_val = NU_Utime(&statobj, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, date, time);
```

Related Topics

[NU_Done](#)[NU_Get_Next](#)[NU_Get_First](#)[Nucleus FILE System APIs](#)

NU_Write

Category: Simple File Operations

Source location: *fs_fl.c*

This function writes count bytes from the specified buffer to the current file pointer of the file described by fd. The file pointer is updated upon completion of the write operation.

Usage

```
INT32 NU_Write (INT    fd,  
                CHAR   *buf,  
                INT32  count)
```

Arguments

- fd
File descriptor to which the file is written.
- buf
Buffer from which information is written.
- count
Number of bytes to be written.

Return Values

- INT32
A non-negative integer to be used as a number of bytes written.
- NUF_BAD_USER
Task not registered as a file user.
- NUF_NOSPC
Write failed, presumably because of no space.
- NUF_BADFILE
File descriptor invalid.
- NUF_IO_ERROR
Driver I/O function routine returned error.
- NUF_NO_DISK
Disk is removed.
- NUF_ACCES
Neither PO_WRONLY or PO_RDWR open flags are set or the file attribute is ARDONLY.
- NUF_INTERNAL
Nucleus FILE internal error.

Legacy Error Codes

- **PEBADF**
File or path not found.
- **PEACCES**
Attempt to open a read only file or special (directory) file.
- **PENOSPC**
No space to write.

Example

```
#include "storage/nu_storage.h"

INT fd;
CHAR buff[512];

/* You might want to put something in buff here */

if (NU_Write(fd, buff, 512) < 0)
    printf("Can't write to file\n");
```

Related Topics

[NU_Open](#)

[Nucleus FILE System APIs](#)

FILE APIs Available to a Nucleus Process

The following is a list of the FILE APIs available from a Nucleus Process:

- [NU_Check_Disk](#)
- [NU_Close](#)
- [NU_Create_Partition](#)
- [NU_Delete](#)
- [NU_Delete_Partition](#)
- [NU_Disk_Abort](#)
- [NU_Done](#)
- [NU_FILE_Cache_Create](#)
- [NU_FILE_Cache_Destroy](#)
- [NU_FILE_Cache_Flush](#)
- [NU_FILE_Cache_Get_Config](#)

- [NU_FILE_Cache_Set_Config](#)
- [NU_Flush](#)
- [NU_Format](#)
- [NU_Free_List](#)
- [NU_Free_Partition_List](#)
- [NU_FreeSpace](#)
- [NU_Get_Attributes](#)
- [NU_Get_First](#)
- [NU_Get_Format_Info](#)
- [NU_Get_Next](#)
- [NU_Get_Partition_Info](#)
- [NU_List_Device](#)
- [NU_List_File_System](#)
- [NU_List_Mount](#)
- [NU_List_Partitions](#)
- [NU_Make_Dir](#)
- [NU_Mount_File_System](#)
- [NU_Open](#)
- [NU_Read](#)
- [NU_Register_File_System](#)
- [NU_Remove_Dir](#)
- [NU_Rename](#)
- [NU_Seek](#)
- [NU_Set_Attributes](#)
- [NU_Storage_Device_Wait](#)
- [NU_Truncate](#)
- [NU_Unmount_File_System](#)
- [NU_Unregister_File_System](#)
- [NU_Utime](#)

- [NU_Write](#)

Nucleus FILE System Shell Commands

The following commands can be issued from a Nucleus Shell command terminal (for example a serial terminal, telnet, and so on).

Note



The Nucleus FILE System Shell commands use a default drive if full path information is not specified, with the default drive set as *A:*. The default drive used by the commands is configurable through the *nu.os.stor.file.shell.default_drv* configuration option.

- [dir](#)
- [del](#)
- [mkdir](#)
- [rmdir](#)
- [copy](#)

dir

Shows the directory listing in the target file system.

Usage

```
dir <file or directory name>
```

Arguments

- `<file or directory name>`
File/directory name for which to show the directory listing. Wildcards are supported.

Return Values

- If operation is successful:
Shows the directory listing.
- If operation fails:
Applicable error string.

Description

Provides a means to discover the contents of the target file system to support development and/or debugging of processes.

Examples

```
> dir
Directory of A:\*. *

03/28/1988  07:37 PM                12345 myapp.load
03/28/1988  07:37 PM      <DIR>          mydir
                                1 File(s)      12345 bytes
                                1 Dir(s)

> dir myapp.*
Directory of A:\myapp.*
03/28/1988  07:37 PM                12345 myapp.load
                                1 File(s)      12345 bytes
                                0 Dir(s)
```

Related Topics

[Nucleus FILE System Shell Commands](#)

[mkdir](#)

[rmdir](#)

del

Deletes a file in the target file system.

Usage

```
del <file name>
```

Arguments

- <file name>
Name (path) of the file to be deleted. Wildcards are supported.

Return Values

- If operation is successful:
No message.
- If operation fails:
Applicable error string.

Description

Provides the ability to delete a file on the target file system to support development and/or debugging of processes.

Examples

```
> del my_app.load

> del not_here.load
ERROR: Could Not Find not_here.load >

del
ERROR: Invalid Usage!
Format: del <file or directory name> >

del in_use.load
ERROR: Failed to delete in_use.load (error=-3034)
```

Related Topics

[Nucleus FILE System Shell Commands](#) [del](#)

mkdir

Makes a directory on the target system.

Usage

```
mkdir <directory name>
```

Arguments

- <directory name>
Name (path) of the directory to be created.

Return Values

- If operation is successful
No message.
- If operation fails
Applicable error string.

Description

Provides the ability to create a directory on the target file system to support development and/or debugging of processes.

Examples

```
> mkdir mydir

> mkdir
ERROR: Invalid Usage!
Format: mkdir <directory name>

> mkdir mydir
ERROR: A subdirectory or file A:\mydir already exists.
```

Related Topics

[Nucleus FILE System Shell Commands](#)

[rmdir](#)

rmdir

Removes a directory on the target file system.

Usage

```
rm <directory name>
```

Arguments

- <directory name>
Name (path) of the directory to be removed.

Return Values

- If operation is successful:
No message.
- If operation fails:
Applicable error string.

Description

Provides the ability to remove a directory on the target file system to support development and/or debugging of processes.

Examples

```
> mkdir mydir  
  
> rmdir mydir  
  
> rmdir  
ERROR: Invalid Usage!  
Format: rmdir <directory name> >
```

```
rmdir mydir ERROR: The system cannot find the file specified.
```

Related Topics

[Nucleus FILE System Shell Commands](#)

[mkdir](#)

copy

Copies a file from one location (source) to another location (destination) in the target file system.

Usage

```
copy <source name> <destination name>
```

Arguments

- <source name>
Path to where the source file is located. Wildcards are not supported.
- <destination name>
Path to where the destination file will be located. Wildcards are not supported.

Return Values

- If operation successful
No output (return to prompt).
- If operation fails
Applicable error string.

Examples

```
> dir
Directory of A:\*
03/28/1988  07:37 PM                14308 test12.load
                1 File(s)            14308 bytes
                0 Dir(s)

> copy test12.load test12_copy.load

> dir
Directory of A:\*
03/28/1988  07:37 PM                14308 test12.load
03/28/1988  07:37 PM                14308 test12_copy.load
                2 File(s)            28616 bytes
                0 Dir(s)

> copy not_here.txt still_not_here.txt

ERROR: Source file does not exist.

>mkdir mydir
```

```
> dir
Directory of A:\*
03/28/1988  07:37 PM                14308 test12.load
03/28/1988  07:37 PM                14308 test12_copy.load
03/28/1988  07:37 PM    <DIR>          mydir
                2 File(s)            28616 bytes
                1 Dir(s)

> copy A:\test12.load A:\mydir\test12.load

> dir A:\mydir
Directory of A:\mydir\*
03/28/1988  07:37 PM    <DIR>          .
03/28/1988  07:37 PM    <DIR>          ..
03/28/1988  07:37 PM                14308 test12.load
                1 File(s)            14308 bytes
                2 Dir(s)
```

Related Topics

[Nucleus FILE System Shell Commands](#)

[del](#)

Chapter 3

Nucleus SQLite

Nucleus SQLite is an open source, public domain implementation of an SQL-based database management system (<http://www.sqlite.org>). It is lightweight, compact, and embeddable, making it ideal for embedded systems. SQLite is ported for three operating systems: Unix, Microsoft Windows, and OS/2.

Nucleus SQLite ports SQLite to the Nucleus Operating System (OS) so it can be used in Nucleus OS-based applications. A separate porting layer maintains the original code independent from Nucleus OS-specific code.

Nucleus SQLite Features

The following are some Nucleus SQLite features inherited from SQLite:

- Nucleus SQLite is a server independent library. It does not require being set up as a server in its own thread or task. Similarly, it does not require any kind of inter task communication. Nucleus SQLite comes in the form of a static library and can be used by calling its APIs directly.
- Nucleus SQLite requires zero configuration. It does not require installation, configuration, nor administrative management.
- Nucleus SQLite is embeddable and compact. It is written in ANSI C and can be included, as a static library, to any embedded application. The size of the library is very small, making it ideal for both high-end and low-end embedded systems.
- Nucleus SQLite uses variable length records. If a 2-byte value is stored in a column of width 1000, it consumes only 2-bytes on the disk.
- Nucleus SQLite is thread safe. Multiple threads (or tasks) can operate using the same data simultaneously (databases, tables, rows, columns, and so on) provided that they use separate database session handles.

Caution



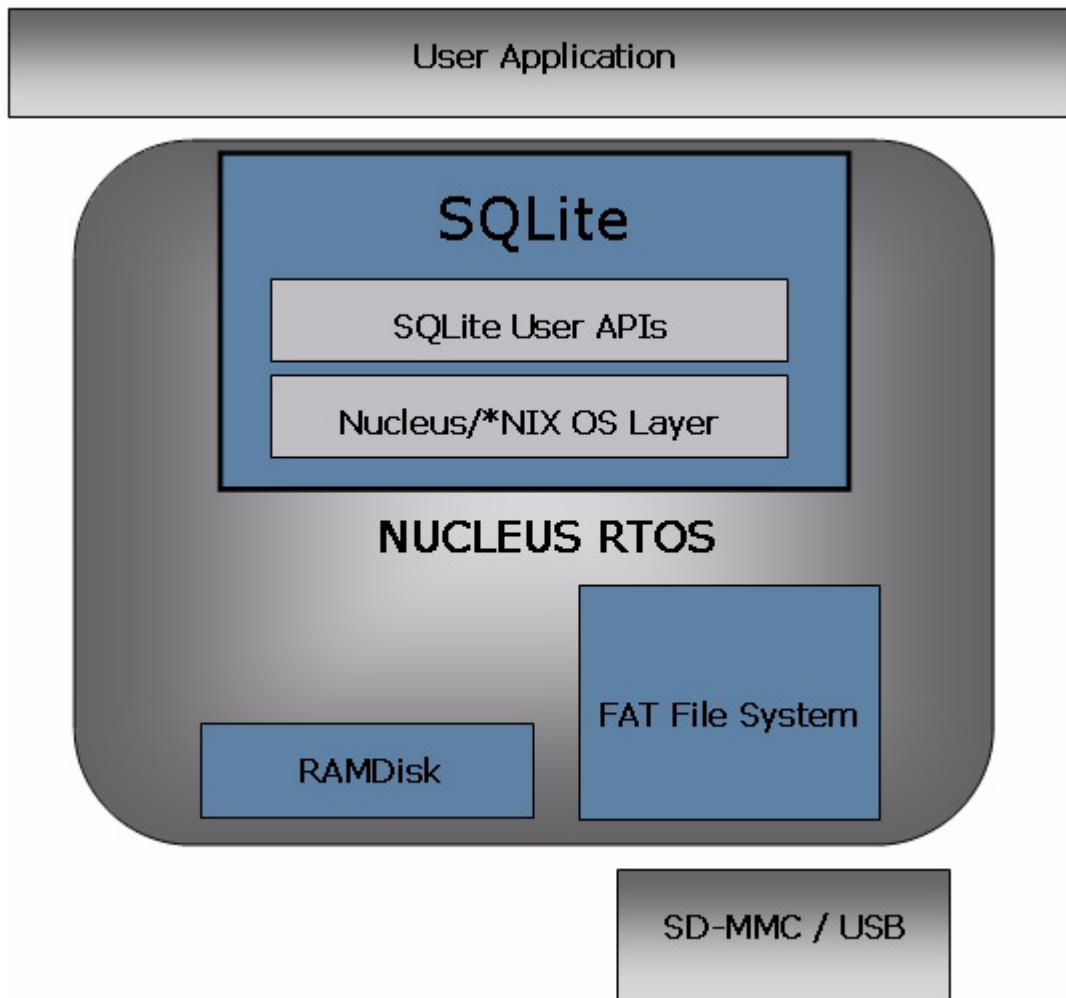
Do NOT use the same database session handle in multiple threads.

SQLite Design

Figure 3-1 shows the Nucleus SQLite Architecture. SQLite provides an OS abstraction layer; therefore, it can be ported just by changing the abstraction layer. The original source code

contains the implementation of the OS abstraction layer for three operating systems: *NIX OS, Windows and OS/2. Nucleus SQLite provides an additional abstraction layer for Nucleus OS.

Figure 3-1. Nucleus SQLite Architecture



Nucleus provides SQLite access to multiple storage mediums through the [Nucleus FILE System](#). In general a persistent storage (SD-MMC, IDE-Drive) is used to maintain databases. The list of persistent storage devices supported are platform dependent. A non-persistent, RAMDisk storage, is suitable for testing purposes.

Initializing Nucleus SQLite

SQLite is enabled through the *.metadata* file, as with other components. If the Nucleus SQLite component is enabled in Nucleus, then it automatically gets initialized on system start-up and all SQLite APIs are ready to be used in the user's application without any explicit initialization operations.

Introduction to the SQLite C Interface

A brief introduction to SQLite and its workings is available at <http://www.sqlite.org/cintro.html>.

Function Reference

All the documentation for the C APIs is available at <http://www.sqlite.org/c3ref/funclist.html>.

Data Structures Reference

All the documentation for the C data structures is available at <http://www.sqlite.org/c3ref/objlist.html>.

Embedded Software and Hardware License Agreement

The latest version of the Embedded Software and Hardware License Agreement is available on-line at:
www.mentor.com/eshla

IMPORTANT INFORMATION

USE OF ALL PRODUCTS IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF PRODUCTS INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

EMBEDDED SOFTWARE AND HARDWARE LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Products (as defined in Section 1) between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Products received electronically, certify destruction of Products and all accompanying items within five days after receipt of such Products and receive a full refund of any license fee paid.

1. **Definitions.** As used in this Agreement and any applicable quotation, supplement, attachment and/or addendum ("Addenda"), these terms shall have the following meanings:
 - 1.1. "Customer's Product" means Customer's end-user product identified by a unique SKU (including any Related SKUs) in an applicable Addenda that is developed, manufactured, branded and shipped solely by Customer or an authorized manufacturer or subcontractor on behalf of Customer to end-users or consumers;
 - 1.2. "Developer" means a unique user, as identified by a unique user identification number, with access to Embedded Software at an authorized Development Location. A unique user is an individual who works directly with the embedded software in source code form, or creates, modifies or compiles software that ultimately links to the Embedded Software in Object Code form and is embedded into Customer's Product at the point of manufacture;
 - 1.3. "Development Location" means the location where Products may be used as authorized in the applicable Addenda;
 - 1.4. "Development Tools" means the software that may be used by Customer for building, editing, compiling, debugging or prototyping Customer's Product;
 - 1.5. "Embedded Software" means Software that is embeddable;
 - 1.6. "End-User" means Customer's customer;
 - 1.7. "Executable Code" means a compiled program translated into a machine-readable format that can be loaded into memory and run by a certain processor;
 - 1.8. "Hardware" means a physically tangible electro-mechanical system or sub-system and associated documentation;
 - 1.9. "Linkable Object Code" or "Object Code" means linkable code resulting from the translation, processing, or compiling of Source Code by a computer into machine-readable format;
 - 1.10. "Mentor Embedded Linux" or "MEL" means Mentor Graphics' tools, source code, and recipes for building Linux systems;
 - 1.11. "Open Source Software" or "OSS" means software subject to an open source license which requires as a condition for redistribution of such software, including modifications thereto, that the: (i) redistribution be in source code form or be made available in source code form; (ii) redistributed software be licensed to allow the making of derivative works; or (iii) redistribution be at no charge;
 - 1.12. "Processor" means the specific microprocessor to be used with Software and implemented in Customer's Product;
 - 1.13. "Products" means Software, Term-Licensed Products and/or Hardware;
 - 1.14. "Proprietary Components" means the components of the Products that are owned and/or licensed by Mentor Graphics and are not subject to an Open Source Software license, as more fully set forth in the product documentation provided with the Products;

- 1.15. “Redistributable Components” means those components that are intended to be incorporated or linked into Customer’s Linkable Object Code developed with the Software, as more fully set forth in the documentation provided with the Products;
- 1.16. “Related SKU” means two or more Customer Products identified by logically-related SKUs, where there is no difference or change in the electrical hardware or software content between such Customer Products;
- 1.17. “Software” means software programs, Embedded Software and/or Development Tools, including any updates, modifications, revisions, copies, documentation and design data that are licensed under this Agreement;
- 1.18. “Source Code” means software in a form in which the program logic is readily understandable by a human being;
- 1.19. “Sourcery CodeBench Software” means Mentor Graphics’ Development Tool for C/C++ embedded application development;
- 1.20. “Sourcery VSIPL++” is Software providing C++ classes and functions for writing embedded signal processing applications designed to run on one or more processors;
- 1.21. “Stock Keeping Unit” or “SKU” is a unique number or code used to identify each distinct product, item or service available for purchase;
- 1.22. “Subsidiary” means any corporation more than 50% owned by Customer, excluding Mentor Graphics competitors. Customer agrees to fulfill the obligations of such Subsidiary in the event of default. To the extent Mentor Graphics authorizes any Subsidiary’s use of Products under this Agreement, Customer agrees to ensure such Subsidiary’s compliance with the terms of this Agreement and will be liable for any breach by a Subsidiary; and
- 1.23. “Term-Licensed Products” means Products licensed to Customer for a limited time period (“Term”).

2. Orders, Fees and Payment.

- 2.1. To the extent Customer (or if agreed by Mentor Graphics, Customer’s appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (“Order(s)”), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement and any applicable Addenda, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 2.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. All invoices will be sent electronically to Customer on the date stated on the invoice unless otherwise specified in an Addendum. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer’s sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer’s behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 2.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics’ delivery of Software by electronic means is subject to Customer’s provision of both a primary and an alternate e-mail address.

3. Grant of License.

- 3.1. The Products installed, downloaded, or otherwise acquired by Customer under this Agreement constitute or contain copyrighted, trade secret, proprietary and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software as described in the applicable Addenda. The limited licenses granted under the applicable Addenda shall continue until the expiration date of Term-Licensed Products or termination in accordance with Section 12 below, whichever occurs first. **Mentor Graphics does NOT grant Customer any right to (a) sublicense or (b) use Software beyond the scope of this Section without first signing a separate agreement or Addenda with Mentor Graphics for such purpose.**
- 3.2. License Type. The license type shall be identified in the applicable Addenda.
- 3.2.1. Development License: During the Term, if any, Customer may modify, compile, assemble and convert the applicable Embedded Software Source Code into Linkable Object Code and/or Executable Code form by the number of Developers specified, for the Processor(s), Customer’s Product(s) and at the Development Location(s) identified in the applicable Addenda.

- 3.2.2. End-User Product License: During the Term, if any, and unless otherwise specified in the applicable Addenda, Customer may incorporate or embed an Executable Code version of the Embedded Software into the specified number of copies of Customer's Product(s), using the Processor Unit(s), and at the Development Location(s) identified in the applicable Addenda. Customer may manufacture, brand and distribute such Customer's Product(s) worldwide to its End-Users.
- 3.2.3. Internal Tool License: During the Term, if any, Customer may use the Development Tools solely: (a) for internal business purposes and (b) on the specified number of computer work stations and sites. Development Tools are licensed on a per-seat or floating basis, as specified in the applicable Addenda, and shall not be distributed to others or delivered in Customer's Product(s) unless specifically authorized in an applicable Addenda.
- 3.2.4. Sourcery CodeBench Professional Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software (i) if the license is a node-locked license, by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, or (ii) if the license is a floating license, by the authorized number of concurrent users on one or more machines provided that only the authorized number of copies of the Software are in use at any one time, and (b) distribute the Redistributable Components of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.5. Sourcery CodeBench Standard Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.6. Sourcery CodeBench Personal Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.7. Sourcery CodeBench Academic Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software for non-commercial, academic purposes only by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.3. Mentor Graphics may from time to time, in its sole discretion, lend Products to Customer. For each loan, Mentor Graphics will identify in writing the quantity and description of Software loaned, the authorized location and the Term of the loan. Mentor Graphics will grant to Customer a temporary license to use the loaned Software solely for Customer's internal evaluation in a non-production environment. Customer shall return to Mentor Graphics or delete and destroy loaned Software on or before the expiration of the loan Term. Customer will sign a certification of such deletion or destruction if requested by Mentor Graphics.

4. Beta Code.

- 4.1. Portions or all of certain Products may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. Restrictions on Use.

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use, including archival and backup purposes. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Except where embedded in Executable Code form in Customer's Product, Customer shall maintain a record of the number and location of all copies of Software, including copies merged with other software and products, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees, authorized manufacturers or authorized contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics immediate written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use.
- 5.2. Customer acknowledges that the Products provided hereunder may contain Source Code which is proprietary and its confidentiality is of the highest importance and value to Mentor Graphics. Customer acknowledges that Mentor Graphics may be seriously harmed if such Source Code is disclosed in violation of this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any Source Code from Products that are not provided in Source Code form. Except as embedded in Executable Code in Customer's Product and distributed in the ordinary course of business, in no event shall Customer provide Products to Mentor Graphics competitors. Log files, data files, rule files and script files generated by or for the Software (collectively "Files") constitute and/or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Under no circumstances shall Customer use Products or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent, which shall not be unreasonably withheld, and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.
- 5.4. Notwithstanding any provision in an OSS license agreement applicable to a component of the Sourcery CodeBench Software that permits the redistribution of such component to a third party in Source Code or binary form, Customer may not use any Mentor Graphics trademark, whether registered or unregistered, in connection with such distribution, and may not recompile the Open Source Software components with the --with-pkgversion or --with-bugurl configuration options that embed Mentor Graphics' trademarks in the resulting binary.
- 5.5. The provisions of this Section 5 shall survive the termination of this Agreement.

6. Support Services.

- 6.1. Except as described in Sections 6.2, 6.3 and 6.4 below, and unless otherwise specified in any applicable Addenda to this Agreement, to the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current End-User Software Support Terms located at <http://supportnet.mentor.com/about/legal/>.
- 6.2. To the extent Customer purchases support services for Sourcery CodeBench Software, support will be provided solely in accordance with the provisions of this Section 6.2. Mentor Graphics shall provide updates and technical support to Customer as described herein only on the condition that Customer uses the Executable Code form of the Sourcery CodeBench Software for internal use only and/or distributes the Redistributable Components in Executable Code form only (except as provided in a separate redistribution agreement with Mentor Graphics or as required by the applicable Open Source license). Any other distribution by Customer of the Sourcery CodeBench Software (or any component thereof) in any form, including distribution permitted by the applicable Open Source license, shall automatically terminate any remaining support term. Subject to the foregoing and the payment of support fees, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current Sourcery CodeBench Software Support Terms located at <http://www.mentor.com/codebench-support-legal>.
- 6.3. To the extent Customer purchases support services for Sourcery VSIPL++, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased solely in accordance with Mentor Graphics' then-current Sourcery VSIPL++ Support Terms located at <http://www.mentor.com/vsipl-support-legal>.
- 6.4. To the extent Customer purchases support services for Mentor Embedded Linux, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased

solely in accordance with Mentor Graphics' then-current Mentor Embedded Linux Support Terms located at <http://www.mentor.com/mel-support-legal>.

7. **Third Party and Open Source Software.** Products may contain Open Source Software or code distributed under a proprietary third party license agreement. Please see applicable Products documentation, including but not limited to license notice files, header files or source code for further details. Please see the applicable Open Source Software license(s) for additional rights and obligations governing your use and distribution of Open Source Software. Customer agrees that it shall not subject any Product provided by Mentor Graphics under this Agreement to any Open Source Software license that does not otherwise apply to such Product. In the event of conflict between the terms of this Agreement, any Addenda and an applicable OSS or proprietary third party agreement, the OSS or proprietary third party agreement will control solely with respect to the OSS or proprietary third party software component. The provisions of this Section 7 shall survive the termination of this Agreement.
8. **Limited Warranty.**
 - 8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual and/or specification. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Products under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; OR (B) PRODUCTS PROVIDED AT NO CHARGE, WHICH ARE PROVIDED "AS IS" UNLESS OTHERWISE AGREED IN WRITING.
 - 8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE TO CUSTOMER AND DO NOT APPLY TO ANY END-USER. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, WITH RESPECT TO PRODUCTS OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.
9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, AND EXCEPT FOR EITHER PARTY'S BREACH OF ITS CONFIDENTIALITY OBLIGATIONS, CUSTOMER'S BREACH OF LICENSING TERMS OR CUSTOMER'S OBLIGATIONS UNDER SECTION 10, IN NO EVENT SHALL: (A) EITHER PARTY OR ITS RESPECTIVE LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF SUCH PARTY OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES; AND (B) EITHER PARTY OR ITS RESPECTIVE LICENSORS' LIABILITY UNDER THIS AGREEMENT, INCLUDING, FOR THE AVOIDANCE OF DOUBT, LIABILITY FOR ATTORNEYS' FEES OR COSTS, EXCEED THE GREATER OF THE FEES PAID OR OWING TO MENTOR GRAPHICS FOR THE PRODUCT OR SERVICE GIVING RISE TO THE CLAIM OR \$500,000 (FIVE HUNDRED THOUSAND U.S. DOLLARS). NOTWITHSTANDING THE FOREGOING, IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
10. **Hazardous Applications.**
 - 10.1. Customer agrees that Mentor Graphics has no control over Customer's testing or the specific applications and use that Customer will make of Products. Mentor Graphics Products are not specifically designed for use in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support systems, medical devices or other applications in which the failure of Mentor Graphics Products could lead to death, personal injury, or severe physical or environmental damage ("Hazardous Applications").
 - 10.2. CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING PRODUCTS USED IN HAZARDOUS APPLICATIONS AND SHALL BE SOLELY LIABLE FOR ANY DAMAGES RESULTING FROM SUCH USE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF PRODUCTS IN ANY HAZARDOUS APPLICATIONS.
 - 10.3. CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING REASONABLE ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10.1.
 - 10.4. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. Infringement.

- 11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay any costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.
 - 11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense, and in addition to its obligations under Section 11.1, either (a) replace or modify the Product so that it becomes noninfringing; or (b) procure for Customer the right to continue using the Product. If Mentor Graphics determines that neither of those alternatives is financially practical or otherwise reasonably available, Mentor Graphics may require the return of the Product and refund to Customer any purchase price or license fee(s) paid.
 - 11.3. Mentor Graphics has no liability to Customer if the claim is based upon: (a) the combination of the Product with any product not furnished by Mentor Graphics, where the Product itself is not infringing; (b) the modification of the Product other than by Mentor Graphics or as directed by Mentor Graphics, where the unmodified Product would not infringe; (c) the use of the infringing Product when Mentor Graphics has provided Customer with a current unaltered release of a non-infringing Product of substantially similar functionality in accordance with Subsection 11.2(a); (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells, where the Product itself is not infringing; (f) any Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; (h) Open Source Software, except to the extent that the infringement is directly caused by Mentor Graphics' modifications to such Open Source Software; or (i) infringement by Customer that is deemed willful. In the case of (i), Customer shall reimburse Mentor Graphics for its reasonable attorneys' fees and other costs related to the action.
 - 11.4. THIS SECTION 11 IS SUBJECT TO SECTION 9 ABOVE AND STATES: (A) THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND (B) CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.
12. **Termination and Effect of Termination.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized Term.
- 12.1. Termination for Breach. This Agreement shall remain in effect until terminated in accordance with its terms. Mentor Graphics may terminate this Agreement and/or any licenses granted under this Agreement, and Customer will immediately discontinue use and distribution of Products, if Customer (a) commits any material breach of any provision of this Agreement and fails to cure such breach upon 30-days prior written notice; or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination. For the avoidance of doubt, nothing in this Section 12 shall be construed to prevent Mentor Graphics from seeking immediate injunctive relief in the event of any threatened or actual breach of Customer's obligations hereunder.
 - 12.2. Effect of Termination. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination or expiration of the Term, Customer will discontinue use and/or distribution of Products, and shall return Hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form, except to the extent an Open Source Software license conflicts with this Section 12.2 and permits Customer's continued use of any Open Source Software portion or component of a Product. Upon termination for Customer's breach, an End-User may continue its use and/or distribution of Customer's Product so long as: (a) the End-User was licensed according to the terms of this Agreement, if applicable to such End-User, and (b) such End-User is not in breach of its agreement, if applicable, nor a party to Customer's breach.
13. **Export.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies. Customer acknowledges that the regulation of product export is in continuous modification by local governments and/or the United States Congress and administrative agencies. Customer agrees to complete all documents and to meet all requirements arising out of such modifications.
14. **U.S. Government License Rights.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

15. **Third Party Beneficiary.** For any Products licensed under this Agreement and provided by Customer to End-Users, Mentor Graphics or the applicable licensor is a third party beneficiary of the agreement between Customer and End-User. Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
16. **Review of License Usage.** Customer will monitor the access to and use of Software. With prior written notice, during Customer's normal business hours, and no more frequently than once per calendar year, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system, records, accounts and sublicensing documents deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all Customer information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. Such license review shall be at Mentor Graphics' expense unless it reveals a material underpayment of fees of five percent or more, in which case Customer shall reimburse Mentor Graphics for the costs of such license review. Customer shall promptly pay any such fees. If the license review reveals that Customer has made an overpayment, Mentor Graphics has the option to either provide the Customer with a refund or credit the amount overpaid to Customer's next payment. The provisions of this Section 16 shall survive the termination of this Agreement.
17. **Controlling Law, Jurisdiction and Dispute Resolution.** This Agreement shall be governed by and construed under the laws of the State of California, USA, excluding choice of law rules. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the state and federal courts of California, USA. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer or its Subsidiary in the jurisdiction where Customer's or its Subsidiary's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.
18. **Severability.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
19. **Miscellaneous.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.