pC/SFS Reference

V1.14b

Das (small) Serial-File-System basiert auf einfach verketteten Blöcken, deren Elemente (DIR/FILE) durch einen 32bit Hash über ihren Namen referenziert werden und wo die Zahl der max. Einträge pro Verzeichnis festgelegt werden muss.
Durch die Verwendung eines 32bit-Hash als Namen kann das Filesystem massiv vereinfacht werden, jedoch sind somit keine Rückgaben zu den Namen der Einträge möglich. Der Hash eines jeden Eintrages muss innerhalb seines Verzeichnisses einmalig sein. Da ein 32bit-Hash Wert an sich nicht garantiert einmalig sein kann - d.h. unter Umständen können zwei Namen den selben Hash Wert ergeben - sind mehrere 32bit-Hash Algorithmen hinterlegt, die Selektion muss aber zu compile-Zeit festgelegt werden.
Das FileSystem verwaltet dabei alle Namen von Verzeichnissen/Dateien als Hash über einen STRING, das heißt auch Sonderzeichen wie Punkt, Leerzeichen und alle anderen gehören zum Namen des Elementes.

Reservierte Names-Elemente:

.. - ein Verzeichnis zurück
/ - am Anfang des Pfades: für ab ROOT
./ - am Anfang des Pfades: für ab current DIR (optional)
/ - innerhalb des Pfades: als Trennung der Verzeichnis/Datei-Namen

Das gesamte FileSystem arbeitet dabei Case-Sensitive !
Als Hardware sind verschiedenste serielle NVM Speicherbausteine (SPI & I2C) getestet (MRAM, FRAM, ReRAM, EEPROM). Außerdem kann das File System auch auf parallelem Speicher wie RAM, FRAM, MRAM oder EEPROM laufen. Ideal sind alle Typen, die byte-weises Schreiben durch internes Puffern von Sectoren/Pages unterstützen, jedoch kann dies auch der Low-Level Hardware-Treiber übernehmen.
Die Verwendung von seriellem Flash-Speicher wurde auf Grund der Sektorengröße von ≥64kB, der notwendigen Pufferung einer Solchen zum Update, sowie die lange Programmierzeit einer ganzen Seite nicht vorgesehen. Bei Verwendung von EEPROM ist außerdem unbedingt daran zu denken, daß 1.000.000 cycles zwar schon recht viel ist, aber auch dies eine endliche Lebenszeit angibt.
Des Weiteren beachtet das SerialFileSystem keine Hot-Spots (High-update files) oder Transaktionen.

User-Functions:

SFS-Control:	Description
SFS_Init	Initialisierung des File-Systems
SFS_GetRev	Gibt Zeiger auf SFS-Revision zurück
SFS_Flush	sichern des SFS als Image in einer Windows/LINUX-Datei
SFS_Format	Formatieren des Laufwerkes

User:	Description
SFS_BecomeUser	User anmelden
SFS_CloseUser	User abmelden

Directories:	Description
SFS_CreateDir	Verzeichnis erstellen
SFS_RemoveDir	Verzeichnis löschen
SFS_RemoveDirTree	Verzeichnis und alle Sub-Elemente löschen
SFS_RenameDir	Verzeichnis umbenennen
SFS_ChangeDir	aktuellen Pfad ändern
SFS_ChangeDirTemp	aktuellen Pfad temporär ändern
SFS_BackDirTemp	temporären Pfad zurücksetzen

Files:	Description

SFS_CreateFile	Datei erstellen
SFS_RemoveFile	Datei löschen
SFS_RenameFile	Datei umbenennen
SFS_AttribFile	Attribute einer Datei ändern
SFS_GetFileAttrib	Gibt Attribute einer Datei zurück
SFS_GetFileSize	Gibt die aktuelle Dateigröße zurück
SFS_GetMaxFileSize	Gibt die maximale Dateigröße zurück
SFS_OpenFile	Datei in "modi" öffnen
SFS_CloseFile	Datei schließen
SFS_SeekFile	Zeiger in geöffneter Datei absolut setzen
SFS_TellFile	Gibt den Zeiger in geöffnete Datei zurück
SFS_SetEOF	setzt EndOfFile in offener Datei an aktuellen R/W-Zeiger
SFS_ReadFile	Lesen aus Datei
SFS_WriteFile	Schreiben in Datei
SFS_GetErrNo	Fehlercode von Open, Read, Write ... lesen

Links:	Description
SFS_CreateLink	Link auf ein Verzeichnis oder eine Datei erstellen
SFS_RemoveLink	Link löschen

Entries:	Description
SFS_GetEntry	Gibt alle Infos des angegebenen Elementes (dir/file/link) zurück

Error-Codes:

Name	Decimal_Value	Description
SFS_NO_ERR	0	no errors
SFS_USR_OVF	200	no user free
SFS_DBL_USER	201	double user
SFS_NO_USER	202	not a valid user
SFS_NAME_EXIST	210	name of entry exist in this DIR
SFS_NOT_EXIST	211	DIR or FILE not exist
SFS_PATH_ERR	212	error on PATH
SFS_TMP_DIR	213	Temp-Dir is still used / not set
SFS_NO_FILE	214	error on PATH / no FILE-Name given
SFS_FILE_RO	215	file to open for writing is read-only
SFS_FILE_WO	216	file to open for reading is write-only
SFS_FILE_EOF	217	end-of-file while reading or writing
SFS_NOT_EMPTY	218	entry not empty
SFS_FILE_OPEN	219	current user have a opened file
SFS_NO_DATA	220	current file-lenght is zero
SFS_WRONG_PTR	221	offset into open file is wrong (or size for R/W)
SFS_NO_ENTRY	222	no free entry in directory
SFS_WRONG_A	223	wrong access flags or attributes given
SFS_LINKED	230	entry is linked
SFS_MAX_LINK	231	entry is max count linked
SFS_LINK_ERR	232	error in link-mechanism
SFS_NO_LINK	233	no link to an entry in link-entry
SFS_MEM_ERR	240	error in block-memory manager
SFS_MEM_OVF	241	memory overflow
SFS_FORMAT_ERR	250	error in found format or during formatting
SFS_PORT_ERR	251	error in HW-port

Konfiguration des File-Systems

Das pC/SFS File-System stellt neben dem verwendetem SPI, I2C oder parallel Speicher-Typ mehrere Möglichkeiten zur Konfiguration von Services sowie zur Reduzierung des Speicherbedarfs - Code-size bei Compilern die "unused code" nicht eindeutig identifizieren können - zur Verfügung. Diese sind in der Datei "SFS_cfg.h" zusammengefaßt.

user configuration	description
SFS_MAX_USER	max users (tasks)
SFS_HANDLES	max files opened by a user (task)
SFS_AUTO_FORMAT	auto-format during init if no valid ROOT-entry was found (alltimes DEEP_FORMAT)
SFS_DEEP_FORMAT	SFS_Format() clears the hole memory of the drive
SFS_AUTO_CLOSE	close all open files of a user automatically on SFS_CloseUser()
SFS_TempDIR	use the one-level temporary current-dir feature
SFS_LINKS	create & delete of links supported
SFS_REMOVEDIRTREE	delete a dir and all sub-elements supported

internal configuration	description
SFS_BLOCK_SIZE	bytes per managed block
SFS_ENTRIES_PER_DIR	entries per dir a 16 byte (one is lost for "..")
SFS_MEM_....	exact type of used SPI, I2C or parallel memory device (to config the LLdriver)

Wenn `SFS_LINKS` nicht gesetzt ist, können keine Links angelegt oder gelöscht werden und somit können auch die gelinken Einträge (DIR/FILE) nicht gelöscht werden. Im File System enthaltene Links können aber ansonsten komplett verarbeitet werden.

Ist das bei der Initialisierung vorgefundene File System kleiner oder anders konfiguriert aber kompatibel (HW-kompatibilität vorrausgesetzt), so werden dessen Einstellungen übernommen und es kann mit diesem File System gearbeitet werden.

allgemeines

SFS_Init

```
U08 SFS_Init(void)
```

Initialisiert das File-System und installiert bei Verwendung eines RTOS die benötigte Mutex/Semaphore. Bei Nutzung des Windows/Linux-HOSTs wird vor dem Formattest versucht ein bestehendes IMAGE aus einer Windows/Linux-Datei zu laden. Sollte das System als unformatiert/inkompatibel erkannt werden, so wird (wenn `SFS_AUTO_FORMAT` enabled ist)`SFS_Format()` ausgeführt.
Diese Funktion muß vor allen anderen SFS-Diensten bei der Systeminitialisierung einmal aufgerufen werden.

Parameters

none

Return Value

SFS_NO_ERR	erfolgreich initialisiert
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung
SFS_FORMAT_ERR	Formatierung unbekannt/inkompatibel
SFS_FILE_OPEN	bei `SFS_AUTO_FORMAT`: ein File ist durch einen user geöffnet

Example

```
void main(void)
{
    U08  returnOk;

    OS_Init();
    .
    .
    returnOk=SFS_Init();
    .
    .
    OS_Start();
}
```

SFS_GetRev

```
void SFS_GetRev(SFS_PATHNAME OS_HUGE **pointer)
```

Gibt einen Zeiger auf die SFS-Revision (NULL-terminiertes ASCII-Array) zurück.

Parameters

**pointer	pointer to pointer will get the address of array

Return Value

none

Example

```
void OS_FAR Task1(void *data)
{
    SFS_PATHNAME OS_HUGE  *Revision;

    .
    .
    while(1)
    {
        .
        SFS_GetRev(&Revision);
        .
    }
}
```

SFS_Flush

```
S32 SFS_Flush(void)
```

Nur bei Nutzung des Windows oder Linux HOSTs
Sichert das File-System als IMAGE in eine Windows/Linux-Datei.

Parameters

none

Return Value

SFS_NO_ERR	*Laufwerk gesichert*
from Windows/LINUX	*see Windows/LINUX*

Example

```
void main(void)
{
    U08  returnOk;

    .
    returnOk=SFS_Init();
    .
    .
    returnOk=SFS_Flush();
}
```

SFS_Format

```
U08 SFS_Format(void)
```

Formatiert das SFS-Laufwerk und trägt den ROOT-Eintrag ein. Dabei darf kein User am File-Sytem angemeldet sein.

Parameters

none

Return Value

SFS_NO_ERR	Laufwerk formatiert
SFS_USER	mindestens ein User ist angemeldet
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung
SFS_MEM_OVF	File-System zu klein für ROOT-Eintrag

Example

```
void OS_FAR Task1(void *data)
{
    U08  returnOk;
    .
    .
    while(1)
    {
        .
        returnOk=SFS_Format();
        .
        .
    }
}
```

SFS_BecomeUser

```
U08 SFS_BecomeUser(SFS_USER OS_HUGE *SFSUser)
```

Legt einen neuen User an.
Diese Funktion initialisiert den User-Control-Block und trägt den neuen User in die interne Liste ein. Jeder Task kann sich nur einmal als User anmelden, anschließend stehen jedem User SFS_HANDLES für Datei-Zugriffe zur Verfügung. Erst nach Anmeldung eines Tasks als User kann dieser Laufwerks- und Dateizugriffe aufrufen.

Parameters

*SFSUser	pointer to user-control-block

Return Value

SFS_NO_ERR	User erfolgreich angelegt
SFS_DBL_USER	dieser User ist bereits angemeldet
SFS_USR_OVF	es sind bereits SFS_maxUSER angemeldet

Example

```
SFS_USER SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
    }
}
```

SFS_CloseUser

```
U08 SFS_CloseUser(void)
```

Löscht einen eingetragenen User aus der internen Liste. Dieser User/Task muß erst erneut angemeldet werden, bevor Zugriffe auf das Laufwerk vom ihm akzepiert werden.

Parameters

none

Return Value

SFS_NO_ERR	User erfolgreich abgemeldet
SFS_FILE_OPEN	User hat aktuell eine Datei geöffnet
SFS_NO_USER	User unbekannt

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_CloseUser();
        .
    }
}
```

Directory and File-Handlings

SFS_CreateDir

```
U08 SFS_CreateDir(SFS_PATHNAME OS_HUGE *Name)
```

Erstellt das angegebene Verzeichnis im aktuellen oder übergebenen Pfad. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Extensions der Verzeichnisse gehören in diesem System mit zum Namen.

Parameters

*Name	Directory-name [with path]

Return Value

SFS_NO_ERR	Verzeichnis erstellt
SFS_NO_USER	User unbekannt
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_NAME_EXIST	ein Verzeichnis mit gleichem Namen existiert bereits in diesem Verzeichnis
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung
SFS_NO_ENTRY	Verzeichnis voll

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_CreateDir("/usr");               // from ROOT
        .
        returnOk=SFS_CreateDir("./demo/test");        // from current directory
        .
        returnOk=SFS_CreateDir("../local/test.src");  // from one level back
        .
        returnOk=SFS_CreateDir("config.save");        // in current directory
        .
        .
        returnOk=SFS_CloseUser();
        .
    }
}
```

SFS_RemoveDir

```
U08 SFS_RemoveDir(SFS_PATHNAME OS_HUGE *Name)
```

Löscht das angegebene Verzeichnis im aktuellen oder übergebenen Pfad. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Das zu löschende Verzeichnis muß dabei leer sein. Desweiteren darf dieses Verzeichnis nicht gelinkt sein.

Parameters

*Name	Directory-name [with path]

Return Value

SFS_NO_ERR	Verzeichnis gelöscht
SFS_NO_USER	User unbekannt
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_NOT_EMPTY	das Verzeichnis ist nicht leer
SFS_LINKED	das Verzeichnis ist durch einen anderen Eintrag gelinkt
SFS_TMP_DIR	Verzeichnis ist aktuelles Verzeichnis eines Users
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_RemoveDir("config.save");      // in actual directory
        .
        .
    }
}
```

SFS_RemoveDirTree

```
U08 SFS_RemoveDirTree(SFS_PATHNAME OS_HUGE *Name)
```

Löscht das angegebene Verzeichnis im aktuellen oder übergebenen Pfad und alle darin enthaltenen Sub-Elemente. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Das zu löschende Verzeichnis muß nicht leer sein. Alle Sub-Einträge werden auch gelöscht. Nur ein Link von außerhalb dieses Sub-Trees in diesen hinein kann nicht aufgelöst werden und erzeugt einen Fehler-code. In diesem Fall bleibt ein bis auf dieses gelinke Element geleertes Verzeichnis zurück. Das Verzeichnis selbst darf nicht gelinkt sein.

Parameters

*Name	Directory-name [with path]

Return Value

SFS_NO_ERR	Verzeichnis gelöscht
SFS_NO_USER	User unbekannt
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_LINKED	ein Element in diesem Verzeichnis-Baum ist von außen gelinkt
SFS_TMP_DIR	Verzeichnis ist aktuelles Verzeichnis eines Users
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_RemoveDirTree("config.V08");   // in actual directory
        .
        .
    }
}
```

SFS_RenameDir

```
U08 SFS_RenameDir(SFS_PATHNAME OS_HUGE *Name, SFS_PATHNAME OS_HUGE *NewName)
```

Ändert den Namen des angegebenen Verzeichnises im aktuellen oder übergebenen Pfad. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Eine Pfadangabe im neuen Namen wird nicht verarbeitet, d.h. das Verzeichnis kann dadurch nicht bewegt werden !

Parameters

*Name	Directory-name [with path]
*NewName	new Directory-name (a path will ignored)

Return Value

SFS_NO_ERR	Verzeichnis umbenannt
SFS_NO_USER	User unbekannt
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_NAME_EXIST	ein Verzeichnis mit gleichem Namen existiert bereits in diesem Verzeichnis
SFS_NOT_EXIST	Verzeichnis existiert nicht in diesem Verzeichnis
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_RenameDir("/usr", "user");      // in ROOT
        .
        .
    }
}
```

SFS_ChangeDir

```
U08 SFS_ChangeDir(SFS_PATHNAME OS_HUGE *Name)
```

wechselt das aktuelle Verzeichnis. Die Pfadangabe kann dabei absolut oder relativ erfolgen.

Parameters

*Name	Directory-name [with path]

Return Value

SFS_NO_ERR	Verzeichnis gewechselt
SFS_NO_USER	User unbekannt
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_NOT_EXIST	das Verzeichnis existiert nicht

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_ChangeDir("../test.src");
        .
        .
        .
    }
}
```

SFS_ChangeDirTemp

```
U08 SFS_ChangeDirTemp(SFS_PATHNAME OS_HUGE *Name)
```

wechselt vorrübergehend (temporär) das aktuelle Verzeichnis. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Das bisherige aktuelle Verzeichnis wird intern vermerkt. Diese Funktionalität kann nur ein Level per User verwendet werden.

Parameters

*Name	Directory-name [with path]

Return Value

SFS_NO_ERR	Verzeichnis gewechselt
SFS_NO_USER	User unbekannt
SFS_TMP_DIR	Temp-Dir ist bereits in Verwendung
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_NOT_EXIST	das Verzeichnis existiert nicht

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_ChangeDirTemp("../userfiles");
        if(returnOk == SFS_NO_ERR) {
            .
            .
            do {
                .
                .
                .
            } while(returnOk == SFS_NO_ERR);
            .
            returnOk=SFS_BackDirTemp();
        }
        .
        .
    }
}
```

SFS_BackDirTemp

```
U08 SFS_BackDirTemp(void)
```

wechselt zurück zum Verzeichnis vor dem vorrübergehenden (temporären) Wechsel in ein anderes Verzeichnis. Das von `SFS_ChangeDirTemp` vermerkte Originalverzeichnis wird wieder das aktuelle Verzeichnis.

Parameters

none

Return Value

SFS_NO_ERR	Verzeichnis gewechselt
SFS_NO_USER	User unbekannt
SFS_TMP_DIR	Temp-Dir ist/wurde nicht gesetzt

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_ChangeDirTemp("../userfiles");
        if(returnOk == SFS_NO_ERR) {
            .
            .
            do {
                .
                .
                .
            } while(returnOk == SFS_NO_ERR);
            .
            returnOk=SFS_BackDirTemp();
        }
        .
        .
    }
}
```

Directory and File-Handlings

SFS_CreateFile

```
U08 SFS_CreateFile(SFS_PATHNAME OS_HUGE *Name, SFS_ATTR Attr, SFS_LONG size)
```

Erstellt die angegebene Datei im aktuellen oder übergebenen Pfad in angegebener Größe. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Als Attribute können ReadOnly oder WriteOnly angegeben werden. ACHTUNG! Dateien besitzen in diesem System keinen Typ.

Parameters

*Name	File-name [with path]
Attr	Attributes of this file
size	max size of file in bytes

Return Value

SFS_NO_ERR	Datei erstellt
SFS_NO_USER	User unbekannt
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_NAME_EXIST	eine Datei mit gleichem Namen existiert bereits in diesem Verzeichnis
SFS_WRONG_A	falsche File Attribute
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung
SFS_MEM_OVF	Laufwerk voll

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_CreateFile("/file1", SFS_ATTR_RO, 100);

                                                    // from ROOT
        .
        returnOk=SFS_CreateFile("../test.src/main.c", SFS_ATTR_RW, 350);

                                                    // from one level back
        .
        returnOk=SFS_CreateFile("makefile.mak", SFS_ATTR_WO, 140);

                                                    // in actual directory
        .
        .
    }
}
```

SFS_RemoveFile

```
U08 SFS_RemoveFile(SFS_PATHNAME OS_HUGE *Name)
```

Löscht die angegebene Datei im aktuellen oder übergebenen Pfad. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Die zu löschende Datei darf dabei von keinem anderen Nutzer geöffnet sein. Desweiteren darf diese Datei nicht gelinkt sein.

Parameters

*Name	File-name [with path]

Return Value

SFS_NO_ERR	Datei gelöscht
SFS_NO_USER	User unbekannt
SFS_FILE_OPEN	User (selber oder anderer) hat aktuell diese Datei geöffnet
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_LINKED	die Datei ist durch einen anderen Eintrag gelinkt
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_RemoveFile("makefile.mak");      // in actual directory
        .
        .
    }
}
```

SFS_RenameFile

```
U08 SFS_RenameFile(SFS_PATHNAME OS_HUGE *OldName, SFS_PATHNAME OS_HUGE *NewName)
```

Ändert den Namen der angegebenen Datei im aktuellen oder übergebenen Pfad. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Die zu ändernde Datei darf dabei von keinem anderen Nutzer geöffnet sein. Eine Pfadangabe im neuen Namen wird nicht verarbeitet, d.h. die Datei kann dadurch nicht bewegt werden !

Parameters

*OldName	File-name [with path]
*NewName	new File-name (a path will ignored)

Return Value

SFS_NO_ERR	Dateiname geändert
SFS_NO_USER	User unbekannt
SFS_NO_FILE	kein Datei-Name im Pfad oder als neuer Name angegeben
SFS_FILE_OPEN	User (selber oder anderer) hat aktuell diese Datei geöffnet
SFS_NAME_EXIST	eine Datei mit gleichem Namen existiert bereits in diesem Verzeichnis
SFS_NOT_EXIST	Datei existiert nicht in diesem Verzeichnis
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_RenameFile("/test.src/main.c", "modul.c");
        .
        .
    }
}
```

SFS_AttribFile

```
U08 SFS_AttribFile(SFS_PATHNAME OS_HUGE *Name, SFS_ATTR Attribs)
```

Ändert die Attribute der angegebenen Datei im aktuellen oder übergebenen Pfad. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Die zu ändernde Datei darf dabei nicht geöffnet sein.

Parameters

*Name	File-name [with path]
Attribs	ew File-attributs

Return Value

SFS_NO_ERR	Dateiattribute geändert
SFS_NO_USER	User unbekannt
SFS_NO_FILE	kein Datei-Name im Pfad angegeben
SFS_FILE_OPEN	User (selber oder anderer) hat aktuell diese Datei geöffnet
SFS_NOT_EXIST	Datei existiert nicht in diesem Verzeichnis
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_WRONG_A	falsche File Attribute

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        returnOk=SFS_AttribFile("../SFS_err.log", SFS_ATTR_RO);
        .
        .
    }
}
```

SFS_GetFileAttrib

```
SFS_ATTR SFS_GetFileAttrib(SFS_PATHNAME OS_HUGE *Name)
```

Gibt die Attribute der angegebenen Datei im aktuellen oder übergebenen Pfad zurück. Die Pfadangabe kann dabei absolut oder relativ erfolgen.

Parameters

*Name	File-name [with path]

Return Value

Sind die Attibute gleich -1, so kann mittels `SFS_GetErrNo()` nachfolgend der FehlerCode abgeholt werden.

SFS_NO_ERR	Dateiattribute gelesen
SFS_NO_USER	User unbekannt
SFS_NO_FILE	kein Datei-Name im Pfad angegeben
SFS_NOT_EXIST	Datei existiert nicht in diesem Verzeichnis
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08       returnOk;
    SFS_ATTR  attribs;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        attribs=SFS_GetFileAttrib("../SFS_err.log");
        if(attribs==(SFS_ATTR)(-1))
            returnOk=SFS_GetErrNo();
        .
        .
    }
}
```

SFS_GetFileSize

```
SFS_LONG SFS_GetFileSize(SFS_PATHNAME OS_HUGE *Name)
```

Gibt die derzeitige Größe der angegebenen Datei im aktuellen oder übergebenen Pfad zurück. Die Pfadangabe kann dabei absolut oder relativ erfolgen.

Parameters

*Name	File-name [with path]

Return Value

Ist die Größe gleich 0, so kann mittels `SFS_GetErrNo()` nachfolgend der FehlerCode abgeholt werden.

SFS_NO_ERR	Dateigröße gelesen
SFS_NO_USER	User unbekannt
SFS_NO_FILE	kein Datei-Name im Pfad angegeben
SFS_NOT_EXIST	Datei existiert nicht in diesem Verzeichnis
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08       returnOk;
    SFS_LONG  filesize;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        filesize=SFS_GetFileSize("/Dir2/config.sys");
        if(!filesize)
            returnOk=SFS_GetErrNo();
        .
        .
    }
}
```

SFS_GetMaxFileSize

```
SFS_LONG SFS_GetMaxFileSize(SFS_PATHNAME OS_HUGE *Name)
```

Gibt die maximale Größe der angegebenen Datei im aktuellen oder übergebenen Pfad zurück. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Das ist die Größe, mit der die Datei einmal angelegt wurde.

Parameters

*Name	File-name [with path]

Return Value

Ist die Größe gleich 0, so kann mittels `SFS_GetErrNo()` nachfolgend der FehlerCode abgeholt werden.

SFS_NO_ERR	Dateigröße gelesen
SFS_NO_USER	User unbekannt
SFS_NO_FILE	kein Datei-Name im Pfad angegeben
SFS_NOT_EXIST	Datei existiert nicht in diesem Verzeichnis
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08       returnOk;
    SFS_LONG  filesize;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        filesize=SFS_GetMaxFileSize("/Dir2/config.sys");
        if(!filesize)
            returnOk=SFS_GetErrNo();
        .
        .
    }
}
```

File-Access

SFS_OpenFile

```
SFS_HANDLE SFS_OpenFile(SFS_PATHNAME OS_HUGE *Name, U08 Mode)
```

Öffnet die angegebenen Datei im aktuellen oder übergebenen Pfad. Die Pfadangabe kann dabei absolut oder relativ erfolgen. Für die nachfolgenden Zugriffe wird ein Handle zurückgegeben, unter diesem auf die Datei-Daten zugegriffen werden kann. Dieses Handle ist intern mit dem User referenziert und wird überprüft. Sollte ein NULL-Handle zurück gegeben worden sein, so ist mittels `SFS_GetErrNo()` der ERROR-Code abholbar.

Access-Bedingungen:
Wenn eine Datei bereits (mehrfach) zum Lesen geöffnet ist, kann ein anderer User diese nicht zum Schreiben öffnen - wenn eine Datei zum Schreiben geöffnet ist, kann kein anderer User diese zum Lesen oder Schreiben öffnen.

Parameters

*Name	File-name [with path]
mode	mode of access (ReadOnly/WriteOnly/ReadWrite)

Return Value

Ist das Handle gleich NULL, so kann mittels `SFS_GetErrNo()` nachfolgend der FehlerCode abgeholt werden.

SFS_NO_ERR	Datei geöffnet
SFS_NO_USER	User unbekannt
SFS_NO_FILE	kein Datei-Name im Pfad angegeben
SFS_FILE_OPEN	aktuell ist diese Datei geöffnet (siehe Access-Bedingungen)
SFS_FILE_RO	Datei ist ReadOnly und kann nicht "write" geöffnet werden
SFS_FILE_WO	Datei ist WriteOnly und kann nicht "read" geöffnet werden
SFS_NOT_EXIST	Datei existiert nicht in diesem Verzeichnis
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_WRONG_A	falsche Access Attribute

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08         returnOk;
    SFS_HANDLE  handl;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        handl=SFS_OpenFile("/Dir2/config.sys", SFS_ACC_RO);
        if(!handl)
            returnOk=SFS_GetErrNo();
        .
        .
```

```
        }
}
```

SFS_CloseFile

```
U08 SFS_CloseFile(SFS_HANDLE Handl)
```

Schließt die aktuell geöffnete Datei.

Parameters

Handl	File-Handle

Return Value

SFS_NO_ERR	Datei geschlossen
SFS_NO_USER	User unbekannt
SFS_NO_FILE	Handle war nicht vergeben

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08         returnOk;
    SFS_HANDLE  handl;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        handl=SFS_OpenFile("/Dir2/config.sys", SFS_ACC_RO);
        if(handl) {
            .
            returnOk=SFS_CloseFile(handl);
            .
        }
        .
    }
}
```

SFS_SeekFile

```
U08 SFS_SeekFile(SFS_LONG offset, SFS_HANDLE Handl)
```

Setzt den R/W-Zeiger innerhalb der geöffneten Datei absolut.

Parameters

offset	absolut pointer-position (in bytes), `0` für start of file, `SFS_SEEK_EOF` für end of file
Handl	File-Handle

Return Value

SFS_NO_ERR	Zeiger in Datei gesetzt
SFS_NO_HAND	Handle ungültig
SFS_NO_USER	User unbekannt
SFS_NO_FILE	Handle war nicht vergeben
SFS_NO_DATA	File has zero-lenght
SFS_WRONG_PTR	offset greater file-size

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08         returnOk;
    SFS_HANDLE  handl;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        handl=SFS_OpenFile("/Dir2/config.sys", SFS_ACC_RO);
        if(handl) {
            .
            returnOk=SFS_SeekFile(100, handl);
        }
        .
    }
}
```

SFS_TellFile

```
SFS_LONG SFS_TellFile(SFS_HANDLE Handl)
```

Gibt den R/W-Zeiger der geöffneten Datei zurück.

Parameters

Handl	File-Handle

Return Value

Ist die Position gleich 0, so kann mittels `SFS_GetErrNo()` nachfolgend der FehlerCode abgeholt werden.

SFS_NO_ERR	Zeiger in Datei gelesen (start of file)
SFS_NO_HAND	Handle ungültig
SFS_NO_USER	User unbekannt
SFS_NO_FILE	Handle war nicht vergeben
SFS_NO_DATA	File has zero-lenght

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08         returnOk;
    SFS_HANDLE  handl;
    SFS_LONG    posit;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        handl=SFS_OpenFile("/Dir2/config.sys", SFS_ACC_RO);
        if(handl) {
            .
            posit=SFS_TellFile(handl);
            if(!posit)
                returnOk=SFS_GetErrNo();

        }
        .
    }
}
```

SFS_SetEOF

```
U08 SFS_SetEOF(SFS_HANDLE Handl)
```

Setzt EndOfFile in offener Datei an aktuellen R/W-Zeiger.

Parameters

Handl	File-Handle

Return Value

SFS_NO_ERR	EndOfFile gesetzt
SFS_NO_HAND	Handle ungültig
SFS_NO_USER	User unbekannt
SFS_NO_FILE	Handle war nicht vergeben
SFS_NO_DATA	File has zero-lenght
SFS_FILE_RO	File or Open-mode is Read-Only

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08         returnOk;
    SFS_HANDLE  handl;
    U08         writebuffer[]={"SFS_Test_File R/W"};
    SFS_LONG    written;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        handl=SFS_OpenFile("/Dir2/config.sys", SFS_ACC_RW);
        if(handl) {
            written=SFS_WriteFile(writebuffer, strlen(writebuffer)-1, handl);
            if(written == strlen(writebuffer)-1)
                returnOk=SFS_SetEOF(handl);
            .
        }
        .
    }
}
```

SFS_ReadFile

```
SFS_LONG SFS_ReadFile(U08 OS_HUGE *dest, SFS_LONG size, SFS_HANDLE Handl)
```

Liest angegebene Zahl Bytes aus aktuell geöffneter Datei ab aktueller Position. Nach erfolgreichem Lesen steht der R/W-Zeiger hinter dem gelesenen Block.
Zurückgegeben wird die Zahl der gelesenen Bytes. Ist diese nicht identisch mit dem Aufruf, so kann mittels `SFS_GetErrNo()` der Fehlercode ermittelt werden.

Parameters

*dest	Pointer to buffer where the bytes must written in
size	Bytes to read
Handl	File-Handle

Return Value

Ist die zurückgegebene Länge ungleich der zu lesenden Bytes, so kann mittels `SFS_GetErrNo()` nachfolgend der FehlerCode abgeholt werden.

SFS_NO_ERR	Bytes aus Datei gelesen
SFS_NO_HAND	Handle ungültig
SFS_NO_USER	User unbekannt
SFS_NO_FILE	Handle war nicht vergeben
SFS_NO_DATA	File has zero-lenght
SFS_FILE_EOF	End-Of-File
SFS_FILE_WO	File or Open-mode is Write-Only
SFS_WRONG_PTR	offset and/or size greater file-size

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08         returnOk;
    SFS_HANDLE  handl;
    U08         readbuffer[100];
    SFS_LONG    readed;

    .
    ...
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        handl=SFS_OpenFile("/Dir2/config.sys", SFS_ACC_RO);
        if(handl) {
            returnOk=SFS_SeekFile(50, handl);
            .
            readed=SFS_ReadFile(&readbuffer[0], 80, handl);
            if(readed != 80)
                returnOk=SFS_GetErrNo();
            .
        }
        .
        .
    }
```

}

SFS_WriteFile

```
SFS_LONG SFS_WriteFile(U08 OS_HUGE *src, SFS_LONG size, SFS_HANDLE Handl)
```

Schreibt angegebene Zahl Bytes in aktuell geöffneter Datei ab aktueller Position. Nach erfolgreichem Schreiben steht der R/W-Zeiger hinter dem geschriebenen Block.
Zurückgegeben wird die Zahl der geschriebenen Bytes. Ist diese nicht identisch mit dem Aufruf, so kann mittels `SFS_GetErrNo()` der Fehlercode ermittelt werden.

Parameters

*src	Pointer to source-buffer
size	Bytes to write
Handl	File-Handle

Return Value

Ist die zurückgegebene Länge ungleich der zu schreibenden Bytes, so kann mittels `SFS_GetErrNo()` nachfolgend der FehlerCode abgeholt werden.

SFS_NO_ERR	Bytes in Datei geschrieben
SFS_NO_HAND	Handle ungültig
SFS_NO_USER	User unbekannt
SFS_NO_FILE	Handle war nicht vergeben
SFS_NO_DATA	File has zero-lenght
SFS_FILE_RO	File or Open-mode is Read-Only
SFS_WRONG_PTR	actual offset plus size greater file-size

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08         returnOk;
    SFS_HANDLE  handl;
    U08         writebuffer[]={"SFS_Test_File R/W"};
    SFS_LONG    written;

    .
    ...
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        handl=SFS_OpenFile("/Dir2/config.sys", SFS_ACC_WO);
        if(handl) {
            written=SFS_WriteFile(writebuffer, strlen(writebuffer)-1, handl);
            if(written != strlen(writebuffer)-1)
                returnOk=SFS_GetErrNo();
            .
            .
            .
        }
        .
    }
}
```

SFS_GetErrNo

```
U08 SFS_GetErrNo(void)
```

Liefert den Fehler-code von Open, Read, Write ...

Parameters

none

Return Value

error-code	Fehler-code der letzten aufgerufenen und mit Fehler behafteten Funktion ohne Fehlercode Rückgabe in API

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08         returnOk;
    SFS_HANDLE  handl;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        .
        handl=SFS_OpenFile("/Dir2/config.sys", SFS_ACC_RO);
        if(!handl)
            returnOk=SFS_GetErrNo();
        .
        .
    }
}
```

Link-Handling

SFS_CreateLink

```
U08 SFS_CreateLink(SFS_PATHNAME OS_HUGE *OrgName, SFS_PATHNAME OS_HUGE *Name)
```

Mit diesem Befehl kann ein Link auf eine Datei oder ein Verzeichnis erstellt werden. Dabei kann sich der Originaleintrag auch in einem anderen Zweig des Verzeichnisbaumes befinden. Es gelten für diesen Link die Attribute des gelinkten Eintrages.

Parameters

*OrgName	File/Directory-name to link [with path]
*Name	Link-name [with path]

Return Value

SFS_NO_ERR	Link erstellt
SFS_NO_USER	User unbekannt
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_NAME_EXIST	ein Eintrag mit gleichem Namen existiert bereits in diesem Verzeichnis
SFS_NO_ENTRY	Verzeichnis voll
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung
SFS_MAX_LINK	*OrgName ist SFS_MAX_Links gelinkt

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    ...
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        returnOk=SFS_CreateLink("/Dir2/config.sys", "../test.dir/linked.conf");
        .
        .
    }
}
```

SFS_RemoveLink

```
U08 SFS_RemoveLink(SFS_PATHNAME OS_HUGE *LinkName)
```

Mit diesem Befehl kann ein erstellter Link auf eine Datei oder ein Verzeichnis gelöscht werden.
--- ZU LÖSCHEN VON DIR / FILE: ---
Gelinkte Verzeichnisse / Dateien können solange nicht gelöscht werden, bis auch der letzte Link auf diesen Eintrag vorher gelöscht wurde !

Parameters

*Name	Link-name [with path]

Return Value

SFS_NO_ERR	Link gelöscht
SFS_NO_USER	User unbekannt
SFS_PATH_ERR	ein Element der Pfadangabe existiert nicht
SFS_MEM_ERR	Fehler innerhalb der Speicherverwaltung
SFS_LINKED	*Name ist selber gelinkt
SFS_NO_LINK	Link zeigt auf kein Element bzw. Eintrag ist kein Link

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08  returnOk;

    .
    ...
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        returnOk=SFS_CreateLink("/Dir2/config.sys", "/test.dir/linked.conf");
        .
        .
        returnOk=SFS_RemoveLink("../test.dir/linked.conf");
    }
}
```

Entries

SFS_GetEntry

```
U08 SFS_GetEntry(SFS_PATHNAME OS_HUGE *Name, SFS_GET OS_HUGE *get)
```

gibt alle relevanten Informationen des angegebenen Elementes in der übergebenen Struktur zurück.

Parameters

*Name	Entry-name (dir/file/link) [with path]
*get	pointer to GET struct

Return Value

SFS_NO_ERR	Eintrag gelesen
SFS_NOT_EXIST	dieser Eintrag existiert nicht
SFS_NO_USER	User unbekannt

Example

```
SFS_USER  SFS_User1;

void OS_FAR Task1(void *data)
{
    U08      returnOk;
    SFS_GET  get;

    .
    while(1)
    {
        .
        returnOk=SFS_BecomeUser(&SFS_User1);
        .
        returnOk=SFS_GetEntry("/Dir2/element", &get);
        .
        if (get.Attr & SFS_ATTR_DIR) {
            .
            .
        }
        .
        .
    }
}
```

Comments

Comments

Comments