Micrium

Empowering Embedded Systems

µC/FS

V4.06.01

Migration Guide

www.Micrium.com

Registration

Please register the software via email. This way we can make sure you will receive updates or notifications of updates as soon as they become available. For registration please provide the following information:

- Your full name and the name of your supervisor
- Your company name
- Your job title
- Your email address and telephone number
- Company name and address
- Your company's main phone number
- Your company's web site address
- Name and version of the product

Please send this information to: licensing@micrium.com

Contact address

Micrium
949 Crestview Circle
Weston, FL 33327-1848
U.S.A.
Phone : +1 954 217 2036
FAX : +1 954 217 2037
WEB : www.micrium.com
Email : support@micrium.com

Document versions

If you find any errors in this document, please inform us and we will make the appropriate corrections for future releases.

Manual Version	Date	Description
V4.06.01	2013/07/10	Updated.
V4.06.00	2013/06/27	Updated.
V4.05.01	2012/09/10	Updated.
V4.05.00	2012/08/16	Updated.
V4.04.05	2012/06/12	Updated.
V4.04.04	2012/06/06	Updated.
V4.04.03	2012/05/10	Updated.
V4.04.01	2010/11/29	Updated.
V4.04	2010/11/11	Updated.
V4.03	2010/05/27	First version of document.

µC/FS v4.06.00 to v4.06.01

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.06.01 from v4.06.00. The changes are easy to make, and updating your project should take a short time.

1.01 New source code

µC/FS v4.06.01 is comprised of mostly bugfixes in various modules. The first step is to replace every file of your project by the new ones.

1.02 API changes

No API changes have been made in µC/FS v4.06.01.

µC/FS v4.05.03 to v4.06.00

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.06.00 from v4.05.03. The changes are easy to make, and updating your project should take a short time.

1.01 New source code and features

The major feature of µC/FS v4.06.00 is the redesignof the journaling module. Other features include support for dumping raw NAND images in the NAND driver along with bug fixes. The first step is to replace every file of your project by the new ones.

1.02 API changes

- Due to a redesignof the journaling module, you must first make sure that any journaled volume used through µC/FS V4.05.03 or earlier has been cleanly unmounted before upgrading to µC/FS V4.06.00. The new journal format is incompatible with the old one and thus any existing old format entry in the journal file would be lost.

- In the previous V4.05.03 release, the default value for the UB_CntMax field of the NAND configuration structure (FS_NAND_DfltCfg) was changed from 10 to 3, causing FSDev_Open to return with error code FS_ERR_DEV_INCOMPATIBLE_LOW_PARAMS on already low-formatted NAND devices. The default value has been reduced to relax the RAM requirements of the default configuration. Changing that value back to 10 on V4.05.03+ will allow previously low-formatted NAND devices to be mounted. Another option would be low-format the NAND device.

  If your project uses the fs_app..c/h template, it was impossible to change the UB_CntMax field without directly modifying fs_app.c. In V4.06.00, a new hidden configuration #define has been added: APP_CFG_FS_NAND_UB_CNT_MAX. This #define, if set, allows you to change the value for UB_CntMax, which would allow you, by setting it to the value 10, to mount pre-V4.05.03 devices without low-level reformating them.

- Some unneeded fields have been removed from core µC/FS structures like the FS_FILE, FS_VOL, FS_DEV, FS_BUF, FS_NAND_DATA, etc. Those structures are not meant to be directly read or modified from the applications, but you may have to adapt your application if you were accessing them directly.

- The FS_CFG_BUILD configuration option in fs_cfg.h has been removed. µC/FS will always be built as it was with FS_CFG_BUILD equal to FS_BUILD_FULL.

µC/FS v4.05.00 to v4.05.01

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.05.01 from v4.05.00. The changes are easy to make, and updating your project should take a short time.

1.01 New source code and features

µC/FS v4.05.01 is comprised of mostly bugfixes in various modules. The first step is to replace every file of your project by the new ones.

1.02 API changes

Some changes were made to the NAND Driver generic controller BSP API and to the NAND static part layer implementation configuration structure. The list below details the changes you need to make to migrate your projects:

- The field NbrOfPgmPerPage has been renamed to NbrPgmPerPg in the FS_NAND_PART_STATIC_CFG static part layer configuration structure. If you use the fs_app.c/h application template, nothing needs to be changed, but if setting the field at runtime, the assignment will need to be corrected.
- Every function in the NAND driver generic controller BSP API except Close(), ChipSelEn() and ChipSelDis() requires the addition of a FS_ERR* error code pointer argument. Use that error code pointer to return any error that need to be reported by the BSP. The Open() and WaitWhileBusy() functions need to have their return types changed to void, the errors being now reported through the new FS_ERR* argument.

µC/FS v4.04.05 to v4.05.00

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.05.00 from v4.04.05. The changes are easy to make, and updating your project should take a short time.

1.01 New source code and features

The major feature of µC/FS v4.05.00 is the release of a new NAND driver. Changes to the rest of the file system is mostly bug fixes and minor features. It is important to update every files of uC/FS and make sure that the required depencies are updated too.

1.02 API changes

No changes to the calling conventions of the API has been done in this release. However, a few error codes were revised. A summary of these changes is presented in the following list.

- FS_DevDrvAdd() now returns FS_ERR_DEV_DRV_NONE_AVAIL instead of FS_ERR_DEV_DRV_NO_TBL_POS_AVAIL when unable to add a driver when reaching the configured limit.
- All functions that were found to return FS_ERR_NULL_PTR when given a NULL pointer as a file name now return the more appropriate FS_ERR_NAME_NULL error code. This is only in effect when argument checking is enabled.
- The error code returned by FSCache_Create() when given an invalid cache configuration is now FS_ERR_INVALID_CFG instead of the unrelated FS_ERR_CACHE_INVALID_SEC_TYPE.

1.03 Multi-Cluster Writes and Reads

uC/FS is now able to perform reads and writes across clusters as one device access if possible. This has the side effect of increasing the maximum transfer size of device access. Users who perform large file access in their application should make sure that the BSPs are able to cope with the increased transfer size.

µC/FS v4.04.04 to v4.04.05

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.04.05 from v4.04.04. The changes are easy to make, and updating your project should take a short time.

1.01 New source code

µC/FS v4.04.04 is comprised of mostly bugfixes in various modules. The first step is to replace every file of your project by the new ones.

1.01 API changes

No API changes have been made in µC/FS v4.04.05.

µC/FS v4.04.03 to v4.04.04

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.04.04 from v4.04.03. The changes are easy to make, and updating your project should take a short time.

1.01 New source code

µC/FS v4.04.04 is comprised of mostly bugfixes in various modules. The first step is to replace every file of your project by the new ones.

1.02 API changes

No API changes have been made in µC/FS v4.04.04.

1.03 Device query

FSDev_Query will now returns valid data for the óFixedô and óStateô fields even if a devices is not accessible. An approriate error is returned exactly as before if the device cannot be accessed. Existing applications should not be impacted.

1.04 RAMDisk removable status

RAMDisk type of devices will now correctly identify themselves as fixed devices instead of removable. Applications that relies on that information should be updated accordingly.

µC/FS v4.04.02 to v4.04.03

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.04.03 from v4.04.02. The changes are easy to make, and updating your project should take a short time.

1.01 New source code

µC/FS v4.04.03 include many improvements and changes in the source code from the last version. Many of them improve either the reliability, the readability or the speed of µC/FS. To migrate to v4.04.01, the first step is to replace every file of your project by the new ones.

1.02 Deprecated source files

The source files fs_pool.c and fs_pool.h are deprecated and should not be included any longer.

1.03 Optional master include file

A new master include file called fs_inc.h can optionally be used instead of including individual header files. User wishing to port projects from µC/FS 4.04 may use it as a drop in replacement for fs.h. However, the fs_api.h containing the defenition of the POSIX compliant API is not included in fs_inc.h.

1.04 Changes in fs_cfg.h

Two new configuration must be #defined in fs_cfg.h, FS_CFG_64_BITS_LBA_EN and FS_CFG_BUF_ALIGN_OCTETS. The first one is used to enable support for 64 bit LBA existing applications should generally set this to DEF_DISABLED to ensure compatibility. The second is used to specify a minimum alignment in octets for the internal file system buffers. See the µC/FS user manual appendix E, titled µC/FS configuration guide, for more details about these new configurations.

1.05 Change to the OS port API

The function FS_OS_WorkingDirSet() was modified to include an error return argument. Projects using a custom OS port not provided Micriµm should update their port accordingly.

1.06 Change to the SD BSP API

Error return arguments were changed from FS_DEV_SD_CARD_ERR to FS_ERR to be more consistent throughout µC/FS.

εC/FS v4.04 to v4.04.01

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.04.01 from v4.04. The changes are easy to make, and updating your project should take a short time.

1.01 New source code

µC/FS v4.04.01 include many improvements and changes in the source code from the last version. Many of them improve either the reliability, the readability or the speed of µC/FS. To migrate to v4.04.01, the first step is to replace every file of your project by the new ones.

1.02 Error codes redesigned

µC/FS v4.04.01 uses enums for the error codes. This is useful when debugging because most debuggers will show the full name instead of a value. Also, fs_err.c is now deprecated and should be removed from your project.

1.03 File inclusion changes

To improve readability and compatibility with third-party tools, the header files are not all included by a master include file anymore. Instead of including fs.h solely, you will have to include appropriate headers, depending on the API you wish to use (fs_file.h, fs_dir.h, fs_vol.h, fs_dev.h, etc.).

1.04 Changes in fs_cfg.h

You must now `#define FS_CFG_BUILD`, `FS_TRACE_LEVEL` and `FS_TRACE` in the file `fs_cfg.h`. Refer to the template in the µC/FS archive for an example.

µC/FS v4.03 to v4.04

The following is a comprehensive list of the modifications you must apply to your µC/FS projects to update them to v4.04 from v4.03. The changes are easy to make, and updating your project should take a short time.

1.01 New source code

µC/FS v4.04 include many improvements and changes in the source code from the last version. Many of them improve either the reliability, the readability or the speed of µC/FS. To migrate to v4.04, the first step is to replace every file of your project by the new ones.

1.02 About µC/Clk

µC/FS v4.04 now includes µC/Clk. µC/Clk εC/Clk is a module that implements a Y2K-compliant clock/calendar. Your application can obtain timestamps in three formats: εC/Clk, UNIX, NTP. εC/Clk purpose is to unify the time/date management across every Micrium products. For example, you could set the timestamp of a file from the time information obtained via a SNTP server (using µC/TCPIP with the optional SNTP module). Note that you can still use a hardware clock (i.e. RTC) or the time from any RTOS.

For v4.04, the use of µC/Clk is mandatory. The `fs_time` module is no longer supported.

Below are the steps you must follow to use µC/Clk. Pleaser refer to the µC/Clk user manual for further informations.

1. Remove files `fs_time.c` and `fs_time.h` from your project
2. Add `clk.c` and `clk.h` to your project
3. Add the path for `clk.h` to your list of include directories
4. Add the file `clk_cfg.h` to your project (use the template in the µC/Clk distribution)
   - Set `CLK_CFG_EXT_EN` appropriately in the file `clk_cfg.h` If `CLK_CFG_EXT_EN` is `DEF_ENABLED`, you must define your own time management functions (`Clk_ExtTS_Init(), Clk_ExtTS_Get()` and `Clk_ExtTS_Set()`).
   - `If CLK_CFG_EXT_EN is DEF_DISABLED`, time will be managed by the OS. You must then add the port files `clk_os.c` and `clk_os.h` to your project. Theses files already exist for µC/OS-II and µC/OS-III. If you are using your own OS, youđll have to write that port yourself. Please refer to µC/Clk user manual for further information.
5. Remove the `#define FS_CFG_GET_TS_FROM_OS` from the file `fs_cfg.h`

If you call `fs_time` functions in your application, you must replace them with calls to the equivalent µC/Clk functions. Below a list of equivalent calls in `fs_time` and µC/Clk.

`FS_DATE_TIME`	replaced by	`CLK_DATE_TIME`
`FSTime_TimeGet()`	replaced by	`Clk_GetDateTime()`
`FSTime_TimeSet()`	replaced by	`Clk_SetDateTime()`
`FSTime_Time_to_Str()`	replaced by	`Clk_DateTimeToStr()`

```
FSTime_Time_to_TS()     replaced  by  Clk_DateTimeToTS_Unix()
FSTime_TS_to_Time()     replaced  by  Clk_TS_UnixToDateTime()
```

```
Day of month  is now [1, 31 ] instead of [0, 30 ].
Month of year is now [1, 12 ] instead of [0, 11 ].
Day of week   is now [1, 7  ] instead of [0, 6  ].
Day of year   is now [1, 366] instead of [0, 365].
```

1.03 Elimination of fs_bsp

The `fs_bsp` module has been eliminated from µC/FS. In order to compile your project with µC/FS 4.04, you must remove the files `fs_bsp.h` and `fs_bsp.c` from your project. If you are not using an OS, you must define the function `FS_BSP_Dly_ms`().

1.04 Changes in fs_cfg.h

You must `#define FS_CFG_CONCURRENT_ENTRIES` in file `fs_cfg.h`. You must also delete the definition of `FS_CFG_GET_TS_FROM_OS`. Refer to the template in the µC/FS archive for an example.

1.05 API changes

Parameter `dir` of function `FSEntry_Create()` has been replaced. The following is the new declaration:
:

```
void  FSEntry_Create(CPU_CHAR        *name_full,
                     FS_FLAGS         entry_type,
                     CPU_BOOLEAN      excl,
                     FS_ERR          *p_err);
```

Parameter `entry_type` can now take the following values:

- `FS_ENTRY_TYPE_FILE`
- `FS_ENTRY_TYPE_DIR`

Parameter `file` of function `FSEntry_Del()`has been replaced. The following is the new declaration:

```
void  FSEntry_Del(CPU_CHAR        *name_full,
                  FS_FLAGS         entry_type,
                  FS_ERR          *p_err);
```

The parameter `entry_type` can now take the following values:

```
-  FS_ENTRY_TYPE_FILE
-  FS_ENTRY_TYPE_DIR
-  FS_ENTRY_TYPE_ANY
```

In `FSEntry_TimeSet()`, the type of the parameter `p_time` has been changed from (`FS_DATE_TIME *)` to (`CLK_DATE_TIME *)`, as in the following delclaration:

```
void  FSEntry_TimeSet (CPU_CHAR      *name_full,
                       CLK_DATE_TIME *p_time,
                       CPU_INT08U     flag,
                       FS_ERR        *p_err);
```

µC/FS v3.xx to v4.04

Though µC/FS V3.xx and V4.04 provide compatibleð and often interchangeableð constructs for accessing files and directories, many differences exist. Some of these, such as the new stratified software directory, little affect the typical userôs experience. Others, particularly in the realm of initialization and device/volume management, will require code modifications for any µC/FS V3 application to work with V4.

1.01 Basic differences

1.01.01 Directories

The directory structure and file names have changed significantly.

Figure 1-1 shows µC/FS V4 directories. See uC-FS-V4 manual for more details regarding each folder.

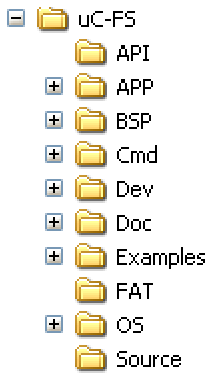

Figure 1-1. µC/FS V4 directories.

1.01.02 Data types

µC/FS V4 now uses µC/CPU (for CPU- and compiler-independent) data types (e.g., `CPU_INT32U`). Since the software no longer defines the `Uxx` and `Ixx` integer types (defined in `global.h` file for µC/FS V3), any application that uses them will need to create `typedef`s to µC/CPU equivalents or replace them by µC/CPU equivalent data types .

1.01.03 Used memory management

µC/FS V4 now relies on µC/LIB for standard library services. The memory for objects used in the software are now allocated from the µC/LIB heap (instead of a block assigned to the file system suite).

µC/FS V4 required memory is allocated by setting the following define to the approriate size in bytes. In general, it is a good practice to have such defines grouped in the same file such as `app_cfg.h` file.

```
#define  LIB_MEM_CFG_HEAP_SIZE                      8192u
```

1.01.04 Configuration file

The configuration file has been renamed *fs_cfg.h* (from *fs_conf.h*) and the configuration define names and values changed. See uC-FS-V4 manual for more details regarding the configuration constant defines.

1.01.05 Device driver

µC/FS V4 provides a template file `uC-FS\APP\Template\fs_app.c` which can be used to initialize the file system and select the approriate device driver.
µC/FS V4 select a driver based on a constant that should be declared by the application . In general, these defines are declared in the `app_cfg.h` file as follow.

```
                                                                  /* Cfg driver inclusion. */

#define  APP_CFG_FS_IDE_EN                   DEF_DISABLED
#define  APP_CFG_FS_MSC_EN                   DEF_DISABLED
#define  APP_CFG_FS_NOR_EN                   DEF_DISABLED
#define  APP_CFG_FS_RAM_EN                   DEF_DISABLED
#define  APP_CFG_FS_SD_EN                    DEF_DISABLED
#define  APP_CFG_FS_SD_CARD_EN               DEF_ENABLED
```

In the above example, the SD card driver in card mode is selected. µC/FS V4 Manual provides more details about each device driver

1.01.06 General configuration

As described in the previous section µC/FS V4 provides a template file `fs_app.c` used to initialize the file system and select the approriate device driver. Initializing µC/FS requires a general configuration structure to be passed as a pointer of type `FS_CFG` to `FS_init()` function.. The following presents an example for the configuration,

```
#define  APP_CFG_FS_DEV_CNT                                1u
#define  APP_CFG_FS_VOL_CNT                                3u
#define  APP_CFG_FS_FILE_CNT                               5u
#define  APP_CFG_FS_DIR_CNT                                1u
#define  APP_CFG_FS_BUF_CNT                (2 * APP_CFG_FS_VOL_CNT)
#define  APP_CFG_FS_DEV_DRV_CNT                            1u
#define  APP_CFG_FS_MAX_SEC_SIZE                         512u
```

The above defines are generally declared in the `app_cfg.h` file and are used in the `fs_app.c` file. See µC/FS V4 manual for more details regarding `FS_CFG` structure.

1.01.07 Long file name (LFN) support

µC/FS V4 now relies on constant define `FS_FAT_CFG_LFN_EN` declared in `fs_cfg.h` file to include/exclude the long file name support.

1.02 API differences

µC/FS V4 still supports *stdio.h*-style file access; however, the function names have been re-aligned with the standard names. Table 1-1 translates the names from µC/FS V3 to V4 and includes in addition functions that have been newly added to the software.

µC/FS V3 Function	**µC/FS V4 Function**
`FS_ClearErr()`	`fs_clearerr()`
`FS_FClose()`	`fs_fclose()`
`FS_FEof()`	`fs_feof()`
`FS_FError()`	`fs_ferror()`
`FS_FOpen()`	`fs_fopen()`
`FS_FRead()`	`fs_fread()`
`FS_FSeek()`	`fs_fseek()`
`FS_FTell()`	`fs_ftell()`
`FS_FWrite()`	`fs_fwrite()`
`FS_GetFilePos()`	`fs_fgetpos()`
`FS_MkDir()`	`fs_mkdir()`
`FS_Remove()`	`fs_remove()`
`FS_Rename()`	`fs_rename()`
`FS_RmDir()`	`fs_rmdir()`
`FS_SetFilePos()`	`fs_fsetpos()`
`FS_Truncate()`	`fs_ftruncate()`
----	`fs_fflush()`
----	`fs_flockfile()`
----	`fs_ftrylockfile()`
----	`fs_funlockfile()`
----	`fs_rewind()`
----	`fs_setbuf()`
----	`fs_setvbuf()`
----	`fs_chdir()`
----	`fs_getcwd()`

Table 1-1. POSIX-Compatible API Translation

Table 1-2 lists approximate µC/FS V4 equivalents to the remaining V3 interface functions. For more information on V4 functions, see µC/FS V4 Manual, Appendix A.

Outside of the POSIX function, API differences are more significant. The volume paradigm used for Initialization differences µC/FS V4 separates volume functionality (access of a file system on some medium) from device functionality (access and control of the medium). This greater formality results in major discrepancies between the interface functions and internal operation of the two versions. Consequently, applications which used the previous software version cannot be upgraded by mere function-for-function replacement. µC/FS V4 Manual includes complete descriptions of all volume interface functions, along with example code listings.

µC/FS V3 Function	**µC/FS V4 Function**
`FS_CopyFile()`	`FSEntry_Copy()`
`FS_FindFirstFile()`	use `fs_opendir()`
`FS_FindNextFile()`	use `fs_readdir_r()`
`FS_FindClose()`	use `fs_closedir()`
`FS_GetFileAttributes()`	`FSEntry_Query()`
`FS_GetFileTime()`	`FSEntry_Query()`
`FS_GetFileTimeEx()`	`FSEntry_Query()`
`FS_TimeStampToFileTime()`	use `FSTime_TS_to_Time()`
`FS_FileTimeToTimeStamp()`	use `FSTime_Time_to_TS()`
`FS_GetFileSize()`	use `FSFile_Query()`
`FS_GetNumVolumes()`	`FSVol_GetVolCnt()`
`FS_GetFreeSpace()`	use `FSVol_Query()`
`FS_GetTotalSpace()`	use `FSVol_Query()`
`FS_GetVolumeFreeSpace()`	use `FSVol_Query()`
`FS_GetVolumeInfo()`	use `FSVol_Query()`
`FS_GetVolumeLabel()`	`FSVol_LabelGet()`
`FS_GetVolumeSize()`	use `FSVol_Query()`
`FS_GetVolumeName()`	`FSVol_GetVolName()`
`FS_GetVolumeStatus()`	use `FSVol_Query()`
`FS_IsVolumeMounted()`	`FSVol_IsMounted()`
`FS_Move()`	use `fs_rename()`
`FS_Read()`	use `fs_fread()`
`FS_SetEndOfFile()`	use `fs_truncate()`
`FS_SetFileAttributes()`	`FSEntry_AttribSet()`
`FS_SetFileTime()`	`FSEntry_TimeSet()`
`FS_SetFileTimeEx()`	`FSEntry_TimeSet()`
`FS_SetVolumeLabel()`	`FSVol_LabelSet()`
`FS_Verify()`	----
`FS_Write()`	use `fs_fwrite()`
`FS_IsHLFormatted()`	use `FSVol_Query()`
`FS_IsLLFormatted()`	use `FSDev_Query()`
`FS_FormatLLIfRequired()`	use `FSDev_Query()` + `FSDev_NOR_LowFmt()`
`FS_FormatLow()`	`FSDev_NOR_LowFmt()`
`FS_Format()`	`FSVol_Fmt()`

Table 1-2. API Translation

The Windows-style directory listing functions, `FS_FindFirstFile()`, `FS_FindNextFile()` and `FS_FindClose()` are not directly represented in µC/FS V4; however, implementations of the *dirent.h* functions (`fs_opendir()`, `fs_readdir_r()` and `fs_close()`) provide the same capabilities.

1.03 Initialization differences

Both µC/FS V3 and V4 have a `FS_Init()` function. In each version, this function initializes core structures. However, in V3, it also calls `FS_X_AddDevices()`, in which the user is expected to place code to add and configure file system devices.

Rather, in V4, the user is expected to add and configure devices after `FS_Init()` returns. The first step, actually, is to register device drivers with the file system software (with `FS_DevDrvAdd()`). After that, one or more devices which use those drivers can be added to file system with `FSDev_Open()`. The first argument of this function is, like the first argument of V3ĉs various volume access functions, a name composed of the device driver name, a colon, the device unit number and another colon. Unlike V3, in which the unit number is assigned by the device driver, v4 uses unit numbers that are also known to the device driver or device driver BSP. If there is a device driver BSP, then the same number is used as the first argument of BSP functions to address the proper device.

In µC/FS V3, the `FS_AddDevice()` function returns a pointer to a `FS_VOLUME`. However, in V4, the concept of device and volume are fully separate, so after a device is added, a volume must be opened on it using `FSVol_Open()`. The first argument of this function is the string that is the volume identifier; the second argument is the device identifier (the string used in `FSDev_Open()`). The two strings may be identical, but that is not necessary, though it may be convenient.

A template file `fs_app.c` for initializing µC/FS V4 is provided within µC/FS V4 distribution. This file is located under `uC-FS\APP\Template` folder.
For more information about initializing a µC/FS V4 project, see the example application described in µC/FS V4 Getting Started document.

1.04 Cache differences

µC/FS V3 integrated into its single cache module two separate concepts of ñcacheò:

1. **Volume cache**, which stores complete sectors of the volume in RAM to accelerate future accesses.

2. **File buffering**, which involves the buffering of file data, as part of reads or writes, so that the user can read/write small chunks of data from/to the file without a volume medium access being necessary for each chunk.

In µC/FS V4, these concepts are treated separately. Volume cache functionality is separated into a cache module. File buffering is implemented in file module functions, and this functionality is accessible in the POSIX API layerĉs `fs_fflush()` and `fs_setvbuf()` function, equivalent to the standard `fflush()` and `setvbuf()`. The corresponding file module functions are `FSFile_BufFlush()` and `FSFile_BufAssign()`.

µC/FS V3 Function	**µC/FS V4 Function**
`FS_AssignCache()`	use `FSVol_CacheAssign()`
`FS_CACHE_Clean()`	use `FSVol_CacheFlush ()`
`FS_CACHE_SetMode()`	----
`FS_CACHE_SetQuota()`	----
----	`FSVol_CacheInvalidate()`

Table 1-3. Cache API Translation

µ