minIni

### 版本 1.4

### 2021年3月

MININI是一个在嵌入式系统中用来读"INI"文件的程序库，MININI使用少量资源，拥有一个确定 的内存占用而且可以被配置于不同的I/O库。

对于MININI最重要的目标是可以被用于嵌入式RTOS(甚至不用任何操作系统)。MININI要求系统

提供一种存储方式和文件和流 I/O，但是它并不要求文件和流 I/O与C/C++标准库兼容—事实上，标准库对于嵌入式系统来说常常过于累赘且造成资源不够用。

**目录**

**介绍** [1](#_bookmark2)

局限性 [1](#_bookmark2)

INI文件语法 [2](#_bookmark4)

**使用**minIni [4](#_bookmark7)

胶水文件 [4](#_bookmark7)

多重任务处理 [10](#_bookmark21)

浏览文件内容 [11](#_bookmark24)

列举键与节 [12](#_bookmark27)

**函数说明** [13](#_bookmark29)

**附录** [24](#_bookmark51)

A: 胶水文件示例 [24](#_bookmark51)

B: 许可证 [29](#_bookmark56)

**索引** [32](#_bookmark58)

### CompuPhase

ii

### **商标**

"Linux"是Linus Torvalds所注册的商标

"Microchip"是Microchip Technology Inc.所注册的商标

"Microsoft"和"MIcrosoft Windows"是Microsoft Corporation所注册的商标

"Unicode"是Unicode Inc.所注册的商标

"wxWidgets"是由Julian Smart和Robert Roebing所领导的公众项目

**版权**

⃝

c 版权 2008–2021, CompuPhase; 第一工业街 19–21 Bussum, 荷兰: (+31)-(0)35 6939 261;

电子邮件: info@compuphase.com; 主页: [http://www.compuphase.com](http://www.compuphase.com/)

MININI库的部分来源于Joseph J.Graf在1994年3月的Dobb博士期刊上发表的文章“Multiplatform.INI文件”。

本手册中的示例和程序具有指导价值。它们经过仔细测试，但不保证适用于所有场合。

**介绍***局限性* — 1

MININI是一个用来读写简单的与"INI"文件格式兼容的配置的库。MININI库功能的代码很少而且它只需要一点点内存(e.g. RAM)。因此它适合用于(小型的)嵌入式系统

INI文件在Microsoft Windows上最为所知, 它的函数 GetProfileString与WritePro-fileString 是用读写 INI 文件的。MININI 中的函数是以Windows SDK为原型, 但是它们并不是完全的与(Windows SDK中的)它们兼容。

尽管 MININI 最突出的是它的体积小,但是它也拥有一些其他功能:

MININI 支持读取不属于任何节的键, 因此它支持不使用节的配置文件 (但是其他方面需要与 INI 文件相兼容)。

⋄

⋄ 支持列举键与节。

你可以使用一个冒号分开键名和键值; 冒号等价于等于号。换个说法,字符串“Name: Value” 与 “Name=Value” 是一个意思

⋄

末端注释 (i.e. 在一对键名/键值 的同一行) 是可以的. 哈希符 (“#”) 是可以替代冒号就行一个解 释。

⋄

⋄

当一个键值包含注释符(“;” or “#”)时, 键值将会自动地将其放入一对双引号之间; 当编写键值 这些引号将排除在外。 当双引号自身出现在其中时, 这些符号将会避开它。

⋄ 灵活的有理数支持, 支持定点数与浮点数。

⋄

因为在闪存(SD/MMC cards, USB memory sticks)中写入速度大大低于读取速度 , MININI 缓存 “文件写入” 用于获得最好的性能, 而且这样的方式不需要额外的内存。

内存占用是固定的。不需要分配动态内存。

⋄

## **局限性**

MININI 力求做到在尽可能少的固定内存中拥有更多功能。 它并不追求速度。 在对表达敏感的代码中, 我建议提前读取所需的任何值或设置，并将其存储在 变量中。

2 — *INI文件语法*

详细的说, MININI 并不会存储任何从INI文件中读取的键名/键值对 。 它不会在两次访问之间保持 INI 文件 处于打开状态 ; MININI 会在每一次读写操作完成后关闭文件

当写入INI 文件时, MININI 将复制源文件并创建一个临时文件 (和修改) 成功完成操作后, 它会删除源文件并重命名临时文件并取而代之。如果必须更改多个设置，则每个设置都会重复此循环。

这没有内置防止多个应用程序访问同一个INI文件(或线程，或任务)的文件锁定机制 。如果一个 INI 文件 被共享到多个程序/线程/任务, 选择与建议请看第10面章节多重任务处理。

## **INI文件语法**

一个INI文件在文本文件中有一个简单的 键名/键值 语法。 每一个键名都要是独特的(在每一节中) 而且与之配对的键值必须在同一行。一个INI文件通常都被分成几个节——在 MININI, 这只是可选的。 一个节的名字需在两个中括号之间,如“[Network]” 在下面的示例列表中.

列表: INI 文件示例

[Network]

hostname = My Computer address = dhcp

dns = 192.168.1.1

在 API 和这篇文章中, “name” 表示为键名。 键值和键名被等号 (“=”)分开 。MININI 支持用冒号 (“:”) 替代等号分隔键名/键值。

在MININI 节和键名不区分大小写(与在 Microsoft Windows API情况一样)。因此, 在INI文件中 “DNS = 192.168.1.1” 等同于 “dns = 192.168.1.1”。

在键值和键名后的空格会被一处。 如果你想在值里面添加空格。 请把值放在两个双引号中间。 [ini\_gets](#_bookmark40) 函数会将双引号除去再返回值。如果值包含空格(或特殊字符) [ini\_puts](#_bookmark50) 函数会将其添入双引号中。

MININI 会忽略 “=” 或 “:” 两旁的空格,节两旁的“[” 和 “]”一样会忽略空格。它并不会删除键与节中的空格。 因此键与节中间可以添加空格。

INI中的注释前必须要有分号 (“;”) 或一个哈希符 (“#”), 而且结尾也一样。 一段注释可以另起一行, 也可以跟在一对 键名/键值 后。

INI文件只有一个层次结构: 节与键。

3 — *使用 minIni*

# **使用** minIni

使用MININI的第一步是确保它能够编译。 该库只由一个C文件和两个头文件组成, 所以要配置的量很少。 如果你不能使用 C/C++ 标准库, 这个就是。然而, 还是有一些配置文件 (或 “胶水” 文件)你必须制作 ; this ﬁle is explained in the next section. The MININI distribution comes with a default conﬁguration ﬁle that maps to the standard C library (speciﬁcally the ﬁle I/O functions from the “stdio” package) and example glue ﬁles for a few embedded ﬁle system libraries for embedded systems —see [appendix A](#_bookmark52) of this manual.

Once you have a good glue ﬁle, you can add the source ﬁle of MININI to your project and include the header ﬁle “minIni.h” in your source code ﬁles. In your source code, you can then use the functions in the MININI library to read text and values from INI ﬁles and to write text and values to an INI ﬁle. See the [Function reference](#_bookmark30) for details.

MININI uses string functions from the standard C/C++ library, including one function that is not in the ANSI C standard: strnicmp. On the Unix and Linux platforms, this function is usually called strncasecmp. If you are using a GNU GCC compiler, but you are not compiling for a Linux or “BSD” platform, you may need to deﬁne strnicmp as strncasecmp in the glue ﬁle (see below). If your compiler provides neither strnicmp nor strncasecmp, you can use a portable implementation in MININI by deﬁning the macro PORTABLE\_STRNICMP in the glue ﬁle (or on the compiler command line).

A notable limitation of MININI is that there is a (ﬁxed) maximum length of a line that can be read from an INI ﬁle. This maximum length is conﬁgurable (at compile-time, not at run-time) and it may be short on embedded systems

—see [page 6](#_bookmark13).

When running in an Unicode environment or when moving the INI ﬁle across platforms, there may be other considerations concerning the use of MININI — see the relevant sections in this chapter, speciﬁcally the section on Unicode on [page 8](#_bookmark18).

## The glue ﬁle

The MININI library must be conﬁgured for a platform with the help of a so- called “glue ﬁle”. This glue ﬁle contains macros (and possibly “inline” func- tions) that map ﬁle reading and writing functions used by the MININI library to those provided by the operating system. The glue ﬁle must be called “minGlue.h”.

*胶水文件* — 4

One general conﬁguration is whether internal error checking via “asser- tions” is active. The MININI library uses the assert macro to help catch errors in the MININI library and/or catch errors in how the application in- terfaces with the MININI library. To build a release version, one typically recompiles all source code with the NDEBUG macro set.

In the case that your (embedded) platform lacks an assert.h ﬁle, you may want to deﬁne NDEBUG in the minGlue.h ﬁle.

### I/O functions

The MININI source code requires functions from a ﬁle I/O library to perform the actual reading and writing. This can be any library; MININI does not rely on the availability of a standard C library, because embedded operating systems may have limited support for ﬁle I/O. Even on full operating sys- tems, separating the ﬁle I/O from the INI format parsing carries advantages, because it allows you to cache the INI ﬁle and thereby enhance performance.

The functions that you need to implement, or map to standard ﬁle I/O func- tions are:

LISTING: Functions to map in the “glue ﬁle”

int ini\_openread(const char \*filename, INI\_FILETYPE \*file) int ini\_openwrite(const char \*filename, INI\_FILETYPE \*file) int ini\_close(INI\_FILETYPE \*file)

int ini\_read(char \*buffer, size\_t size, INI\_FILETYPE \*file) int ini\_write(char \*buffer, INI\_FILETYPE \*file)

int ini\_rename(const char \*source, const char \*dest) int ini\_remove(const char \*filename)

int ini\_tell(INI\_FILETYPE \*file, INI\_FILEPOS \*pos) int ini\_seek(INI\_FILETYPE \*file, INI\_FILEPOS \*pos)

All functions should return zero on failure and a non-zero value on success. For examples of “implementations” for the above functions, see appendix A on [page 24](#_bookmark52).

The ini\_remove function is redundant if ini\_rename overwrites an existing destination ﬁle. When using GCC and Glibc as the C library, you may use the behaviour of the rename function (from Glibc) to overwrite the destina- tion if it exists. Removing the original ﬁle before renaming the new ﬁle to the original name is then redundant, and the deﬁnition of ini\_remove is then likewise redundant. When not using GCC and Glibc, you should con- sult the documentation for your compiler and library; standard C deﬁnes the behaviour of rename in this condition as implementation-deﬁned.

5 — *胶水文件*

The INI\_FILETYPE type used in the above “glue” functions, must also be de- ﬁned in the glue ﬁle. If you are using the standard C/C++ ﬁle I/O library, this is the “FILE\*” type of the standard C/C++ ﬁle I/O library. On embedded systems with a diﬀerent I/O library, chances are that you need a diﬀerent handle or “structure” to identify the storage. For example:

#define INI\_FILETYPE HANDLE

The MININI functions will declare variables of the INI\_FILETYPE type and pass these variables to sub-functions (including the glue interface functions) by reference.

For read-only support of INI ﬁles, only the macros/functions ini\_openread, ini\_close and ini\_read are needed (see also [page 7](#_bookmark15)). The other functions are only needed for writing support. The type that holds the “ﬁle position” (for functions ini\_tell and ini\_seek) must be declared as well. For appli- cations that use the standard C/C++ ﬁle I/O library functions fgetpos and fsetpos, this is the fpos\_t type.

#define INI\_FILEPOS fpos\_t

Function ini\_openread is for opening an existing ﬁle, and for opening it for reading only. Function ini\_openwrite must create a new ﬁle, or delete and re-create an existing ﬁle. The deﬁnition of the function ini\_openrewrite is optional; if available, it is used to open an existing ﬁle for writing, but without truncating the ﬁle (many libraries call this “read + write mode”). Function ini\_openrewrite allows for an optimization in the special case that an update of a setting does not cause the ﬁle length to be changed.

On Microsoft Windows and DOS, ﬁles can be opened in either “text mode” or in “binary mode”, and this relates mostly on the line termination translation. Despite INI ﬁles being text ﬁles, it is advised to open the INI ﬁle in binary mode.

See see appendix A on [page 24](#_bookmark52) for examples of glue ﬁles for various ﬁle systems.

### Buffer size (maximum line length)

Another item that needs to be conﬁgured is the buﬀer size. The functions in the MININI library allocate this buﬀer on the stack, so the buﬀer size is directly related to the stack usage. In addition, the buﬀer size determines the maximum line length that is supported in the INI ﬁle and the maximum path name length for the temporary ﬁle (for writing support). For example, minGlue.h could contain the deﬁnition:

*胶水文件* — 6

#define INI\_BUFFERSIZE 512

The above macro limits the line length of the INI ﬁles supported by MININI to 512 characters.

The buﬀer size declared here is also the size of the “write cache” that MININI

uses to optimize performance on ﬁle writes.

### Read-only support

In its default conﬁguration, MININI supports both reading and writing INI ﬁles. If your application does not require write support, you can add a setting to the minGlue.h ﬁle to strip out the unneeded code.

#define INI\_READONLY

When writing a setting to an INI, MININI writes it to a temporary ﬁle, copies the other sections and keys from the original INI ﬁle, and then deletes the original ﬁle and renames the temporary ﬁle to the name of the original ﬁle. This approach uses the least amount of memory. The disadvantage is that writes to an INI ﬁle are slow, especially on large INI ﬁles.

Furthermore, when writing to the temporary ﬁle, MININI repeatedly looks ahead in the source INI and jumps back to a position that it marked earlier. The goal of this design is to minimize the number of inidividual “write ac- tions” to the ﬁle, because on Flash memory (and EEPROM memory), writing is an order of magnitude slower than reading.

The path name of the temporary ﬁle is the same as the input ﬁle, but with the last character set to a tilde (“~”).

### Browsing support

An aﬀecient way to scan through the complete INI ﬁle and read all settings, is with [ini browse](#_bookmark31), see [page 11](#_bookmark26). Browsing support may be excluded from the MININI library by deﬁning the INI\_NOBROWSE deﬁnition.

### Rational number support

MININI can be conﬁgured to support reading and writing single-precision ﬂoating point values —see the functions [ini getf](#_bookmark34) and [ini putf](#_bookmark47). Embed- ded processors may lack ﬂoating point hardware and software emulation of ﬂoating-point operations may be too costly in resources (memory). For these platforms, alternatives are to switch to a ﬁxed-point representation or, when rational numbers are not relevant for the project, to disable the rational number support in MININI altogether.

7 — *胶水文件*

To enable rational number support, a macro for the type and macros or inter- face functions for number-to-text conversions must be added to minGlue.h. For the standard C/C++ library, you can add the following deﬁnitions to the glue ﬁle:

#define INI\_REAL float

#define ini\_ftoa(string,value) sprintf((string),"%f",(value)) #define ini\_atof(string) (INI\_REAL)strtod((string),NULL)

For a diﬀerent representation of rational numbers, only the deﬁnitions in minGlue.h have to change. The following example is based on the “fixedptc” library by Ivan Voras.

#define INI\_REAL fixedpt

#define ini\_ftoa(string,value) fixedpt\_str((value),(string)) #define ini\_atof(string) fixedpt\_val((string))

To disable rational number support, remove the declaration for the INI\_REAL

type from the minGlue.h ﬁle.

### Unicode (启用/禁用)

MININI can be compiled with Unicode support, but it delegates storing the actual characters to the “glue” routines. Although you can use standard Unicode ﬁle reading and writing routines to create and query INI ﬁles in Unicode text format, it is advised to keep the INI ﬁle format as ASCII, for best compatibility with other implementations. To store Unicode characters in the ASCII ﬁle, convert the Unicode data to (and from) UTF-8 (the MININI library does not provide functions for this conversion).

It is advised to keep the section and key names as ASCII or ANSI Latin-1; only the “values” of each key should be encoded as UTF-8.

Currently, all distributions of Linux lack a header ﬁle called tchar.h which adds a portability layer for source code that can be compiled as ASCII or as Unicode. MININI relies on tchar.h when compiling for Unicode. Therefore, when compiling a Unicode application under Linux, you have two options: create a minimal version of tchar.h yourself, or compile MININI for the 8-bit ANSI character set, while the remainder of the application is Unicode. To force-compile MININI for ANSI, add the deﬁnition INI\_ANSIONLY in the glue ﬁle (“minGlue.h”). For example:

#define INI\_ANSIONLY /\* ignore UNICODE or \_UNICODE macros, compile as ASCII/ANSI \*/

### Line termination

*胶水文件* — 8

On Microsoft Windows and DOS, lines of text ﬁles are usually terminated by a CR-LF character pair (“\r\n” in C/C++ terminology). On Linux and Unix (and macOS), the line terminator is only the LF character.

The line termination convention is not important when reading from INI ﬁles, because MININI strips oﬀ all trailing white space (and control characters such as carriage-return and line-feed are considered white space). The line ter- mination convention is also not important when the INI ﬁle is only accessed by MININI. Finally, if you use the standard C/C++ library as the back-end for reading and writing ﬁles, this standard C/C++ library may already handle the platform-dependent line termination for you.

However, if you wish to read and adjust the INI ﬁles with other applications, across platforms —e.g. edit the INI ﬁle with a simple text editor as Notepad on Microsoft Windows and then store it on an embedded device with a Linux- based operating system, then it may be advantageous to tell MININI the line termination characters to use. To do so, deﬁne the macro INI\_LINETERM in the ﬁle “minGlue.h” and set it to the character or characters to use. For example:

#define INI\_LINETERM "\r\n"

### Summary of conﬁguration macros

INI ANSIONLY If this macro is deﬁned, INI ﬁles are forced to be written with 8-bit characters (ASCII or ANSI character sets), regard- less of whether the remainder of the application is written as Unicode. See [page 8](#_bookmark18).

INI BUFFERSIZE The maximum line length that is supported, as well as the maximum path length for temporary ﬁle (if write access is enabled). The default value is 512. See [page 6](#_bookmark13).

INI FILEPOS The type for a position in a ﬁle. This is a *required setting*

if writing support is enabled.

INI FILETYPE The type for a variable that represents a ﬁle. This is a

*required setting*. See [page 6](#_bookmark11).

INI LINETERM This macro should be set to the line termination charac- ter (or characters). If left undeﬁned, the default is a line- feed character. Note that the standard ﬁle I/O library may translate a line-feed character to a carriage-return/line- feed pair (this depends on the ﬁle I/O library). See [page](#_bookmark20) [9](#_bookmark20)

9 — *多重任务处理*

INI NOBROWSE Exclude the [ini browse](#_bookmark31) function from the MININI library.

INI READONLY If this macro is deﬁned, write access is disabled (and the code for writing INI ﬁles is stripped from the MININI library. See [page 7](#_bookmark15)

INI REAL The type for a variable that represents a rational number. If left undeﬁned, rational number support is disabled. See [page 7](#_bookmark16).

NDEBUG If deﬁned, the assert macro in the MININI source code is disabled. Typically developers build with assertions en- abled during development and disable them for a release version. If your platform lacks an assert macro, you may want to deﬁne the NDEBUG macro in the minGlue.h ﬁle.

PORTABLE STRNICMP

If deﬁned, MININI uses an internal, portable strnicmp func- tion. This is required for platforms that lack this func- tion —note that MININI already handles the case where this function is called strncasecmp. See [page 4](#_bookmark9).

## Multi-tasking

The MININI library does not have any global variables and it does not use any dynamically allocated memory. Yet, the library should not be considered “thread-safe” or re-entrant, because it implicitly uses a particular shared resource: the ﬁle system.

Multiple tasks reading from an INI ﬁle do not pose a problem. However, when one task is writing to an INI ﬁle, no other tasks should access this INI ﬁle —neither for reading, nor for writing. It might be easier, in the imple- mentation, to serialize *all* accesses of the INI ﬁle.

The ﬁrst advise in protecting resources from concurrent access in a multi- tasking environment is to avoid sharing resources between tasks. If only a single task uses a resource, no semaphore protection is necessary and no priority inversion or deadlock problems can occur. This advise also applies to the MININI library. If possible, make a single task the “owner” of the INI ﬁle and create a client/server architecture for other tasks to query and adjust settings.

If access to the INI ﬁle must be shared between tasks (and at least one of the tasks writes to the INI ﬁle), you need to write wrappers around the functions of the MININI library that block on a mutex or binary semaphore, or that use the ﬁle locking mechanism in the operating system. See the next sections for tips speciﬁc to an operating systems.

### Linux

*浏览文件内容* — 10

An option in Linux (and other Unix-like environments) is to use an advisory lock on the calls to open the INI ﬁle. In the snippet below, the glue function ini\_openread sets a “shared” ﬁle lock (allowing others to also open the ﬁle for reading), but ini\_openwrite sets an “exclusive” lock on the ﬁle.

The lock can only be set after opening the ﬁle, which is why ini\_openwrite ﬁrst attempts to open an existing ﬁle, and creates a new ﬁle if no existing ﬁle can be opened. If it had started by creating a new ﬁle, the call to fopen with the mode "w" would have truncated the ﬁle *before* aquiring the lock

—thereby possibly truncating a locked ﬁle. If an existing ﬁle was indeed opened, and the lock aquired, it must now be explicitly truncated.

The flock function is blocking by default, meaning that it does not pro- ceed if it cannot aquire the lock. It thereby implicitly functions as a kind of semaphore.

LISTING: Glue function using ﬁle locking

static inline int ini\_openread(const char \*filename, INI\_FILETYPE \*file) { if ((\*file = fopen((filename),"r")) == NULL)

return 0;

return flock(fileno(\*file), LOCK\_SH) == 0;

}

static inline int ini\_openwrite(const char \*filename, INI\_FILETYPE \*file) { if ((\*file = fopen((filename),"r+")) == NULL

&& (\*file = fopen((filename),"w")) == NULL) return 0;

if (flock(fileno(\*file), LOCK\_EX) < 0) return 0;

return ftruncate(fileno(\*file), 0) == 0;

}

Note that no “unlock” request is needed: the ﬁle is implicitly unlocked when the ﬁle is closed.

## Browsing through the ﬁle contents

The “browse” function [ini browse](#_bookmark31) processes the complete INI ﬁle and in- vokes a callback function for every setting that it reads from the ﬁle.

An alternative to browsing through the INI ﬁle is by enumerating the sec- tions and the keys, see the next section. Browsing is more eﬀicient when the whole INI ﬁle must be processed, enumeration allows you to scan only through speciﬁc sections.

11 — *Key and section enumeration*

## Key and section enumeration

MININI can list all sections in an INI ﬁle and all keys in a section, but in a diﬀerent way than the function GetProfileString from the Microsoft Win- dows API. To list all sections, call function [ini getsection](#_bookmark42) with an incre- mental “index” number until it fails. Similarly, to list all keys in a section, call [ini getkey](#_bookmark36) with an incremental “index” number (plus the name of the section) until it fails.

LISTING: Browsing through all keys and all sections in “conﬁg.ini”

int s, k;

char section[40], key[40];

for (s = 0; ini\_getsection(s, section, sizeof section, "config.ini") > 0; s++) { printf("[%s]\n", section);

for (k = 0; ini\_getkey(section, k, key, sizeof key, "config.ini") > 0; k++) printf("\t%s\n", key);

} /\* for \*/

*ini browse / minIni::browse* — 12

# **函数说明**

In addition to the functions in plain C, minIni comes with a C++ class. When creating a variable of the minIni class, you pass in the name of the INI ﬁle once, so that this name does not need to be passed to every other function. The class exists for the standard C++ string library and for wxWidgets, using the wxString type. The function reference only lists the methods with the std::string type, but these are replaced by versions that use wxString when compiling for wxWidgets.

minIni::minIni class constructor

The minIni constructor creates an instance of the minIni class.

C++: minIni(const std::string& filename)

filename The full ﬁle name of the INI ﬁle to use for all reads and writes, through this instance. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O library.

Example: Creating a class instance to read a setting (C++ only):

minIni ini("config.ini");

std::string username = ini.gets("Users", "admin");

ini browse / minIni::browse Browse through all settings

ini\_browse runs through the ﬁle and invokes a callback on every setting.

C: int ini browse(INI CALLBACK Callback, void \*UserData,

const char \*Filename)

C++: bool browse(INI CALLBACK Callback, void \*UserData)

Callback The function that is invoked on every setting, see the notes below.

UserData A general-purpose application-deﬁned value that is passed to the callback function.

13 — *minIni::del*

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: 1/true on success, 0/false on failure. Notes: The callback function is deﬁned as:

int Callback(const char \*Section, const char \*Key, const char \*Value, void \*UserData);

The Section string may an empty string, for the settings that are outside any section. The UserData parameter is the same as what is passed to the ini\_browse function.

The callback function should return a non-zero value on success, or zero to abort further browsing through the INI ﬁle.

The callback function should not write into the INI ﬁle.

See also: [ini getkey](#_bookmark36), [ini getsection](#_bookmark42)

minIni::del Delete a section or a key

Delete a key or an entire section.

C++: bool del(const std::string& Section,

const std::string& Key)

C++: bool del(const std::string& Section) Section The name of the section.

Key The name of the key. Returns: true on success, false on failure.

Notes: This method is the equivalent of ini\_puts with the parameter Key

and/or Value parameters to *NULL*.

This function is unavailable if MININI is conﬁgured as a read-only library ([page 7](#_bookmark15)).

See also: [ini puts](#_bookmark50)

*ini getbool / minIni::getbool* — 14

ini getbool / minIni::getbool Read a “truth” ﬂag

ini\_getbool returns zero for false or one for true, depending on the value that is found in the given section and at the given key.

C: int ini getbool(const char \*Section, const char \*Key,

int DefValue, const char \*Filename)

C++: bool getbool(const std::string& Section,

const std::string& Key, bool DefValue=false)

Section The name of the section. If this parameter is *NULL* or an empty string, the Key is searched outside any section.

Key The name of the key. This parameter may not be

*NULL*.

DefValue The default value, which will be returned if the key is not present in the INI ﬁle. Even though it is declared as an “int” in the C interface, it should be either 0 (zero) or 1 (one).

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: The true/false ﬂag as interpreted from the value read at the given key, or DefValue if the key is not present in the given section (or if it cannot be interpreted to either a “true” or a “false” ﬂag).

Speciﬁcally, the return value depends on the ﬁrst letter of the value read at the key. If that ﬁrst character is:

⋄ “Y”, “T” or “1”, the function returns true (or 1);

⋄ “N”, “F” or “0”, the function returns false (or 0);

⋄ anything else, the function returns parameter DefValue.

Notes: To set a boolean value in the C++ interface, use [minIni::put](#_bookmark48); For the C interface, use either [ini putl](#_bookmark48). Alternatively, you can store texts like “Yes” and “No” at the key using [minIni::put](#_bookmark50) and [ini puts](#_bookmark50).

See also: [ini getl](#_bookmark38)

15 — *ini getf / minIni::getf*

ini getf / minIni::getf Read a rational number

ini\_getf returns the numeric value that is found in the given section and at the given key. The value may have a fractional part (i.e. rational numbers).

C: INI REAL ini getf(const char \*Section, const char \*Key,

INI REAL DefValue, const char \*Filename)

C++: INI REAL getf(const std::string& Section,

const std::string& Key, INI REAL DefValue=0)

Section The name of the section. If this parameter is *NULL* or an empty string, the Key is searched outside any section.

Key The name of the key. This parameter may not be

*NULL*.

DefValue The default value, which will be returned if the key is not present in the INI ﬁle.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: The value read at the given key, or DefValue if the key is not present in the given section.

Notes: Rational number support must have been *enabled* to use the func- tion —see [page 7](#_bookmark16). The type for the rational numbers (the INI\_REAL type), depends on the conﬁguration of MININI.

See also: [ini getl](#_bookmark38), [ini putf](#_bookmark47)

ini getkey / minIni::getkey Enumerate keys Read the name of an indexed key inside a given section.

C: int ini getkey(const char \*Section, int Index,

char \*Buffer, int BufferSize, const char \*Filename)

C++: str::string getkey(const std::string& Section, int Index)

Section The name of the section. If this parameter is *NULL* or an empty string, the keys outside any section are enumerated.

*ini getl / minIni::getl* — 16

Index The zero-based index of the key to return.

Buffer The buﬀer into which this function will store the key name.

BufferSize The size of the buﬀer in parameter Buffer. This is the maximum number of characters that will be read and stored.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: The C function returns the number of characters that were read, or zero if no (more) keys are present in the speciﬁed section. The C++ method returns the name of the key in a string.

Example: Enumerating keys in section “Devices”:

int k;

char name[20];

for (k = 0; ini\_getkey("Devices", k, name, 20, "config.ini") > 0; k++) printf("%s\n", name);

See also: [ini browse](#_bookmark31), [ini getsection](#_bookmark42), [ini haskey](#_bookmark44)

ini getl / minIni::getl Read a numeric value

ini\_getl returns the integer value (a “whole number”) that is found in the given section and at the given key.

C: long ini getl(const char \*Section, const char \*Key,

long DefValue, const char \*Filename)

C++: long getl(const std::string& Section,

const std::string& Key, long DefValue=0)

C++: int geti(const std::string& Section,

const std::string& Key, int DefValue=0)

Section The name of the section. If this parameter is *NULL* or an empty string, the Key is searched outside any section.

Key The name of the key. This parameter may not be

*NULL*.

17 — *ini gets / minIni::gets*

DefValue The default value, which will be returned if the key is not present in the INI ﬁle.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: The value read at the given key, or DefValue if the key is not present in the given section.

If the key is present, but it does not represent a decimal number, this function may return zero or an incorrect value.

Notes: The number must be in decimal or in hexadecimal format. For hexadecimal values, the value must be preceded with “0x”; for example, 0x1234 stands for the decimal value 4660.

See also: [ini getf](#_bookmark34), [ini gets](#_bookmark40), [ini putl](#_bookmark48)

ini gets / minIni::gets Read a string

ini\_gets reads the textual value that is found in the given section and at the given key.

C: int ini gets(const char \*Section, const char \*Key,

const char \*DefValue, char \*Buffer, int BufferSize, const char \*Filename)

C++: std::string gets(const std::string& Section,

const std::string& Key,

const std::string& DefValue="")

Section The name of the section. If this parameter is *NULL* or an empty string, the Key is searched outside any section.

Key The name of the key. This parameter may not be

*NULL*.

DefValue The default value, which will be returned (in parame- ter Buffer) if the key is not present in the INI ﬁle.

Buffer The buﬀer into which this function will store the data read.

*ini getsection / minIni::getsection* — 18

BufferSize The size of the buﬀer in parameter Buffer. This is the maximum number of characters that will be read and stored.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: The C function returns the number of characters that were read. The C++ method returns the string read at the given key, or Def- Value if the key is not present in the given section.

See also: [ini getl](#_bookmark38), [ini puts](#_bookmark50)

ini getsection / minIni::getsection Enumerate sections

ini\_getsection reads the name of an indexed section.

C: int ini getsection(int Index, char \*Buffer, int BufferSize,

const char \*Filename)

C++: std::string getsection(int Index)

Index The zero-based index of the section to return.

Buffer The buﬀer into which this function will store the sec- tion name.

BufferSize The size of the buﬀer in parameter Buffer. This is the maximum number of characters that will be read and stored.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: The C function returns the number of characters that were read, or zero if no (more) sections are present in the INI ﬁle. The C++ method returns the name of the section in a string.

Example: Enumerating all sections in ﬁle “conﬁg.ini”:

19 — *ini haskey / minIni::haskey*

int s;

char name[20];

for (s = 0; ini\_getsection(s, name, 20, "config.ini") > 0; s++) printf("%s\n", name);

See also: [ini browse](#_bookmark31), [ini getkey](#_bookmark36), [ini hassection](#_bookmark45)

ini haskey / minIni::haskey Check whether a key exists

ini\_haskey checks whether a key exists in a section( without returning its value).

C: int ini haskey(const char \*Section, const char \*Key,

const char \*Filename)

C++: bool haskey(const std::string& Section,

const std::string& Key)

Section The name of the section. If this parameter is *NULL* or an empty string, the Key is searched outside any section.

Key The name of the key. This parameter may not be

*NULL*.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: 1/true if the key is present, 0/false otherwise. See also: [ini getkey](#_bookmark36), [ini hassection](#_bookmark45)

ini hassection / minIni::hassection Check whether a section exists

ini\_hassection checks whether a key exists in a section( without returning its value).

C: int ini hassection(const char \*Section,

const char \*Filename)

C++: bool hassection(const std::string& Section)

Section The name of the section. This parameter may not be

*NULL*.

*ini putf / minIni::put* — 20

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: 1/true if the section is present, 0/false otherwise. See also: [ini getsection](#_bookmark42), [ini haskey](#_bookmark44)

ini putf / minIni::put Store a rational number

ini\_putf stores the numeric value that in the given section and at the given key. The numeric value is written as a rational number, with a “whole part” and a fractional part.

C: int ini putf(const char \*Section, const char \*Key,

INI REAL Value, const char \*Filename)

C++: bool put(const std::string& Section,

const std::string& Key, INI REAL Value)

Section The name of the section. If this parameter is *NULL* or an empty string, the Key is stored outside any sec- tion (i.e. above the ﬁrst section, if the INI ﬁle has any sections).

Key The name of the key. This parameter may not be

*NULL*.

Value The value to write at the key and the section.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: 1/true on success, 0/false on failure.

Notes: This function is unavailable if MININI is conﬁgured as a read-only library ([page 7](#_bookmark15)). It is also unavailable if rational number support has *not* been enabled ([page 7](#_bookmark16)).

The type for the rational numbers, INI\_REAL, depends on the con- ﬁguration of MININI.

See also: [ini getf](#_bookmark34), [ini putl](#_bookmark48)

21 — *ini putl / minIni::put*

ini putl / minIni::put Store a numeric value

ini\_putl stores the numeric value that in the given section and at the given key.

C: int ini putl(const char \*Section, const char \*Key,

long Value, const char \*Filename)

C++: bool put(const std::string& Section,

const std::string& Key, long Value)

Section The name of the section. If this parameter is *NULL* or an empty string, the Key is stored outside any sec- tion (i.e. above the ﬁrst section, if the INI ﬁle has any sections).

Key The name of the key. This parameter may not be

*NULL*.

Value The value to write at the key and the section.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: 1/true on success, 0/false on failure.

Notes: This function is unavailable if MININI is conﬁgured as a read-only library ([page 7](#_bookmark15)).

See also: [ini getl](#_bookmark38), [ini puts](#_bookmark50)

ini puts / minIni::put Store a string

ini\_puts stores the text parameter that in the given section and at the given key.

C: int ini puts(const char \*Section, const char \*Key,

const char \*Value, const char \*Filename)

C++: bool put(const std::string& Section,

const std::string& Key, const std::string& Value)

C++: bool put(const std::string& Section,

const std::string& Key, const char\* Value)

*ini puts / minIni::put* — 22

Section The name of the section. If this parameter is *NULL* or an empty string, the Key is stored outside any sec- tion (i.e. above the ﬁrst section, if the INI ﬁle has any sections).

Key The name of the key. If this parameter is *NULL*, the function erases all keys (and their associated values) from the section.

Value The text to write at the key and the section. This string should not contain carriage-return or line-feed characters.

If this parameter is *NULL*, the function erases the key/value pair.

Filename The name of the INI ﬁle. The ﬁlename format and speciﬁcations, and whether or not this parameter may include a path, depends on the underlying ﬁle I/O li- brary.The C++ class uses the ﬁlename speciﬁed in the class constructor.

Returns: 1/true on success, 0/false on failure.

Notes: This function can also be used to delete entries or sections, by setting the Key or Value parameters to *NULL*.

This function is unavailable if MININI is conﬁgured as a read-only library ([page 7](#_bookmark15)).

See also: [ini gets](#_bookmark40), [ini putl](#_bookmark48)

23 — *Example glue ﬁles*

Example glue ﬁles

附录 A

### stdio (standard C/C++ library)

On Microsoft Windows or DOS, it is advised to open the INI ﬁle in binary mode, despite INI ﬁles being *text* ﬁles. If text mode is unavailable on yor platform, change "rb" and "wb" to "r" and "w" respectively.

/\* map required file I/O types and functions to the standard C library \*/ #include <stdio.h>

#define INI\_FILETYPE FILE\*

#define ini\_openread(filename,file) ((\*(file) = fopen((filename),"rb")) != NULL) #define ini\_openwrite(filename,file) ((\*(file) = fopen((filename),"wb")) != NULL) #define ini\_openrewrite(filename,file) ((\*(file) = fopen((filename),"r+b")) != NULL) #define ini\_close(file) (fclose(\*(file)) == 0)

#define ini\_read(buffer,size,file) (fgets((buffer),(size),\*(file)) != NULL) #define ini\_write(buffer,file) (fputs((buffer),\*(file)) >= 0)

#define ini\_rename(source,dest) (rename((source), (dest)) == 0) #define ini\_remove(filename) (remove(filename) == 0)

#define INI\_FILEPOS long int

#define ini\_tell(file,pos) (\*(pos) = ftell(\*(file)))

#define ini\_seek(file,pos) (fseek(\*(file), \*(pos), SEEK\_SET) == 0)

/\* for floating-point support, define additional types and functions \*/ #define INI\_REAL float

#define ini\_ftoa(string,value) sprintf((string),"%f",(value)) #define ini\_atof(string) (INI\_REAL)strtod((string),NULL)

### CCS FAT library (http://www.ccsinfo.com)

#define INI\_BUFFERSIZE 256 /\* maximum line length, maximum path length \*/ #ifndef FAT\_PIC\_C

#error FAT library must be included before this module #endif

#define const /\* keyword not supported by CCS \*/

#define INI\_FILETYPE FILE

#define ini\_openread(filename,file) (fatopen((filename), "r", (file)) == GOODEC) #define ini\_openwrite(filename,file) (fatopen((filename), "w", (file)) == GOODEC) #define ini\_close(file) (fatclose((file)) == 0)

#define ini\_read(buffer,size,file) (fatgets((buffer), (size), (file)) != NULL) #define ini\_write(buffer,file) (fatputs((buffer), (file)) == GOODEC) #define ini\_remove(filename) (rm\_file((filename)) == 0)

#define INI\_FILEPOS fatpos\_t

#define ini\_tell(file,pos) (fatgetpos((file), (pos)) == 0) #define ini\_seek(file,pos) (fatsetpos((file), (pos)) == 0)

#ifndef INI\_READONLY

/\* CCS FAT library lacks a rename function, so instead we copy the file to the

\* new name and delete the old file

*Example glue ﬁles* — 24

\*/

static int ini\_rename(char \*source, char \*dest)

{

FILE fr, fw; int n;

if (fatopen(source, "r", &fr) != GOODEC) return 0;

if (rm\_file(dest) != 0) return 0;

if (fatopen(dest, "w", &fw) != GOODEC) return 0;

/\* With some "insider knowledge", we can save some memory: the "source"

* parameter holds a filename that was built from the "dest" parameter. It
* was built in a local buffer with the size INI\_BUFFERSIZE. We can reuse
* this buffer for copying the file.

\*/

while (n=fatread(source, 1, INI\_BUFFERSIZE, &fr)) fatwrite(source, 1, n, &fw);

fatclose(&fr); fatclose(&fw);

/\* Now we need to delete the source file. However, we have garbled the buffer

* that held the filename of the source. So we need to build it again.

\*/

ini\_tempname(source, dest, INI\_BUFFERSIZE); return rm\_file(source) == 0;

}

#endif

### EFSL [(http://ww](http://www.efsl.be/))w[.efsl.be/)](http://www.efsl.be/))

#define INI\_BUFFERSIZE 256 /\* maximum line length, maximum path length \*/ #define INI\_LINETERM "\r\n" /\* set line termination explicitly \*/

#include "efs.h"

extern EmbeddedFileSystem g\_efs;

#define INI\_FILETYPE EmbeddedFile

\

#define ini\_openread(filename,file) (file\_fopen((file), &g\_efs.myFs,

(char\*)(filename), 'r') == 0) #define ini\_openwrite(filename,file) (file\_fopen((file), &g\_efs.myFs,

\

(char\*)(filename), 'w') == 0)

#define ini\_close(file) file\_fclose(file)

#define ini\_read(buffer,size,file) (file\_read((file), (size), (buffer)) > 0) #define ini\_write(buffer,file) (file\_write((file), strlen(buffer),

\

(char\*)(buffer)) > 0)

#define ini\_remove(filename) rmfile(&g\_efs.myFs, (char\*)(filename))

#define INI\_FILEPOS euint32

#define ini\_tell(file,pos) (\*(pos) = (file)->FilePtr)) #define ini\_seek(file,pos) file\_setpos((file), (\*pos))

#if ! defined INI\_READONLY

/\* EFSL lacks a rename function, so instead we copy the file to the new name

\* and delete the old file

\*/

static int ini\_rename(char \*source, const char \*dest)

25 — *Example glue ﬁles*

{

EmbeddedFile fr, fw; int n;

if (file\_fopen(&fr, &g\_efs.myFs, source, 'r') != 0) return 0;

if (rmfile(&g\_efs.myFs, (char\*)dest) != 0) return 0;

if (file\_fopen(&fw, &g\_efs.myFs, (char\*)dest, 'w') != 0) return 0;

/\* With some "insider knowledge", we can save some memory: the "source"

* parameter holds a filename that was built from the "dest" parameter. It
* was built in buffer and this buffer has the size INI\_BUFFERSIZE. We can
* reuse this buffer for copying the file.

\*/

while (n=file\_read(&fr, INI\_BUFFERSIZE, source)) file\_write(&fw, n, source);

file\_fclose(&fr); file\_fclose(&fw);

/\* Now we need to delete the source file. However, we have garbled the buffer

* that held the filename of the source. So we need to build it again.

\*/

ini\_tempname(source, dest, INI\_BUFFERSIZE); return rmfile(&g\_efs.myFs, source) == 0;

}

#endif

### FAT Filing System [(http://ww](http://www.embedded-code.com/))w[.embedded-code.com/)](http://www.embedded-code.com/))

#define INI\_BUFFERSIZE 256 /\* maximum line length, maximum path length \*/ #include <mem-ffs.h>

#define INI\_FILETYPE FFS\_FILE\*

#define ini\_openread(filename,file) ((\*(file) = ffs\_fopen((filename),"r")) != NULL) #define ini\_openwrite(filename,file) ((\*(file) = ffs\_fopen((filename),"w")) != NULL) #define ini\_close(file) (ffs\_fclose(\*(file)) == 0)

#define ini\_read(buffer,size,file) (ffs\_fgets((buffer),(size),\*(file)) != NULL) #define ini\_write(buffer,file) (ffs\_fputs((buffer),\*(file)) >= 0)

#define ini\_rename(source,dest) (ffs\_rename((source), (dest)) == 0) #define ini\_remove(filename) (ffs\_remove(filename) == 0)

#define INI\_FILEPOS long

#define ini\_tell(file,pos) (ffs\_fgetpos(\*(file), (pos)) == 0) #define ini\_seek(file,pos) (ffs\_fsetpos(\*(file), (pos)) == 0)

*Example glue ﬁles* — 26

### FatFs [(http://elm-chan.org/](http://elm-chan.org/)))

#define INI\_BUFFERSIZE 256 /\* maximum line length, maximum path length \*/

/\* You must set \_USE\_STRFUNC to 1 or 2 in the include file ff.h (or tff.h)

\* to enable the "string functions" fgets() and fputs().

\*/

#include "ff.h" /\* include tff.h for Tiny-FatFs \*/

#define INI\_FILETYPE FIL

\

#define ini\_openread(filename,file) (f\_open((file), (filename),

FA\_READ+FA\_OPEN\_EXISTING) == FR\_OK)

\

#define ini\_openwrite(filename,file) (f\_open((file), (filename),

FA\_WRITE+FA\_CREATE\_ALWAYS) == FR\_OK)

#define ini\_close(file) (f\_close(file) == FR\_OK) #define ini\_read(buffer,size,file) f\_gets((buffer), (size),(file)) #define ini\_write(buffer,file) f\_puts((buffer), (file)) #define ini\_remove(filename) (f\_unlink(filename) == FR\_OK)

#define INI\_FILEPOS DWORD

#define ini\_tell(file,pos) (\*(pos) = f\_tell((file)))

#define ini\_seek(file,pos) (f\_lseek((file), \*(pos)) == FR\_OK)

static int ini\_rename(TCHAR \*source, const TCHAR \*dest)

{

/\* Function f\_rename() does not allow drive letters in the destination file \*/ char \*drive = strchr(dest, ':');

drive = (drive == NULL) ? dest : drive + 1; return (f\_rename(source, drive) == FR\_OK);

}

### “Memory Disk Drive” ﬁle system (Microchip)

#define INI\_BUFFERSIZE 256 /\* maximum line length, maximum path length \*/ #include "MDD File System\fsio.h"

#include <string.h>

#define INI\_FILETYPE FSFILE\*

#define ini\_openread(filename,file) ((\*(file) = FSfopen((filename),FS\_READ)) != NULL)

#define ini\_openwrite(filename,file) ((\*(file) = FSfopen((filename),FS\_WRITE)) != NULL)

#define ini\_openrewrite(filename,file) ((\*(file) = fopen((filename),FS\_READPLUS)) != NULL)

#define ini\_close(file) (FSfclose(\*(file)) == 0)

#define ini\_write(buffer,file) (FSfwrite((buffer), 1, strlen(buffer), (\*file))

> 0)

#define ini\_remove(filename) (FSremove((filename)) == 0)

#define INI\_FILEPOS long int

#define ini\_tell(file,pos) (\*(pos) = FSftell(\*(file)))

#define ini\_seek(file,pos) (FSfseek(\*(file), \*(pos), SEEK\_SET) == 0)

/\* Since the Memory Disk Drive file system library reads only blocks of files,

* the function to read a text line does so by "over-reading" a block of the
* of the maximum size and truncating it behind the end-of-line.

\*/

static int ini\_read(char \*buffer, int size, INI\_FILETYPE \*file)

{

27 — *Example glue ﬁles*

size\_t numread = size; char \*eol;

if ((numread = FSfread(buffer, 1, size, \*file)) == 0) return 0; /\* at EOF \*/

if ((eol = strchr(buffer, '\n')) == NULL) eol = strchr(buffer, '\r');

if (eol != NULL) {

/\* terminate the buffer \*/

\*++eol = '\0';

/\* "unread" the data that was read too much \*/

FSfseek(\*file, - (int)(numread - (size\_t)(eol - buffer)), SEEK\_CUR);

} /\* if \*/ return 1;

}

#ifndef INI\_READONLY

static int ini\_rename(const char \*source, const char \*dest)

{

FSFILE\* ftmp = FSfopen((source), FS\_READ); FSrename((dest), ftmp);

return FSfclose(ftmp) == 0;

}

#endif

License

The software product “MININI” is copyright ⃝c

*License* — 28

APPENDIX B

2008–2020 by CompuPhase.

MININI is distributed under the “Apache License” version 2.0, which is re- produced below, plus an exception clause to explicitly allow static linking.

**EXCEPTION TO THE APACHE 2.0 LICENSE**

As a special exception to the Apache License 2.0 (and referring to the definitions in Section 1 of this license), you may link, statically or dynamically, the “Work” to other modules to produce an executable file containing portions of the “Work”, and distribute that executable file in “Object” form under the terms of your choice, without any of the additional requirements listed in Section 4 of the Apache License

2.0. This exception applies only to redistributions in “Object” form (not “Source” form) and only if no modifications have been made to the “Work”.

**TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION**

1. **Definitions**

“License” shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

“Licensor” shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

“Legal Entity” shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, “control” means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

“You” (or “Your”) shall mean an individual or Legal Entity exercising permissions granted by this License.

“Source” form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

“Object” form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

“Work” shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work.

“Derivative Works” shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

“Contribution” shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, “submitted” means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as “Not a Contribution.”

“Contributor” shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

29 — *License*

1. **Grant of Copyright License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sub-license, and distribute the Work and such Derivative Works in Source or Object form.
2. **Grant of Patent License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, oﬀer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counter-claim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
3. **Redistribution.** You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
   1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
   2. You must cause any modified files to carry prominent notices stating that You changed the files; and
   3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
   4. If the Work includes a “NOTICE” text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or diﬀerent license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

1. **Submission of Contributions.** Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
2. **Trademarks.** This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
3. **Disclaimer of Warranty.** Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an “AS IS” BASIS, WITHOUT WAR- RANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of us- ing or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
4. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work

*License* — 30

stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

1. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to oﬀer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

31 — *Index*

# **索引**

Names of persons or companies (not products) are in *italics*.

⋄

Function names, constants and compiler reserved words are in typewriter font.

⋄

A Advisory lock, [11](#_bookmark25), *see also* File lock ASCII, [8](#_bookmark17), [9](#_bookmark19)

assert, [4](#_bookmark8)

B Binary mode, [6](#_bookmark12), [24](#_bookmark52)

C C++ class, [13](#_bookmark30)

Case-insensitive (comparisons), [2](#_bookmark5)

*CCS Inc.*, [24](#_bookmark52)

Comments

INI ﬁle ~, [1](#_bookmark3), [3](#_bookmark6) Compilation

release build, [4](#_bookmark8)

Constructor, [13](#_bookmark30)

D Delete (key or section), [14](#_bookmark32)

I INI ﬁle comments, [1](#_bookmark3), [3](#_bookmark6)

ini browse, [13](#_bookmark30)

ini getbool, [14](#_bookmark32)

ini getf, [15](#_bookmark33)

ini getkey, [16](#_bookmark35)

ini getl, [17](#_bookmark37)

ini gets, [18](#_bookmark39)

ini getsection, [19](#_bookmark41)

ini haskey, [20](#_bookmark43)

ini hassection, [20](#_bookmark43)

ini putf, [21](#_bookmark46)

ini putl, [21](#_bookmark46)

ini puts, [22](#_bookmark49)

K Key enumeration, *see* Enumeration

L License, [29](#_bookmark57)

Limitations, [1](#_bookmark3), [4](#_bookmark8)

Line termination, [6](#_bookmark12), [9](#_bookmark19)

Linux, [4](#_bookmark8), [8](#_bookmark17), [9](#_bookmark19), [11](#_bookmark25)

E EFSL (ﬁle system), [25](#_bookmark53)

Enumeration, [1](#_bookmark3), [12](#_bookmark28), [16](#_bookmark35), [19](#_bookmark41)

F Fat Filing System, [26](#_bookmark54) FatFs, [27](#_bookmark55)

File lock, [10](#_bookmark22)

File system, [24](#_bookmark52), *see also* Glue ﬁle Fixed point, [7](#_bookmark14), [16](#_bookmark35), [21](#_bookmark46)

Flash memory, [1](#_bookmark3), [7](#_bookmark14)

Floating point, [7](#_bookmark14), [16](#_bookmark35), [21](#_bookmark46)

G GCC, [5](#_bookmark10)

GetProfileString, [1](#_bookmark3), [12](#_bookmark28)

Glibc, [5](#_bookmark10)

Glue ﬁle, [2](#_bookmark5), [4](#_bookmark8), [24](#_bookmark52)

GNU GCC, [4](#_bookmark8)

*Graf, Joseph J.* ~, [ii](#_bookmark1)

H Hexadecimal, [18](#_bookmark39)

Hierarchy, [3](#_bookmark6)

M MDD (ﬁle system), [27](#_bookmark55)

Memory footprint, [i](#_bookmark0), [1](#_bookmark3), [10](#_bookmark22)

*Microchip Technology Inc.*, [27](#_bookmark55)

minGlue.h, [4](#_bookmark8), [24](#_bookmark52)

MinGW, [4](#_bookmark8)

minIni::browse, [13](#_bookmark30)

minIni::del, [14](#_bookmark32)

minIni::getbool, [14](#_bookmark32)

minIni::getf, [15](#_bookmark33)

minIni::getkey, [16](#_bookmark35)

minIni::getl, [17](#_bookmark37)

minIni::gets, [18](#_bookmark39)

minIni::getsection, [19](#_bookmark41)

minIni::haskey, [20](#_bookmark43)

minIni::hassection, [20](#_bookmark43)

minIni::minIni, [13](#_bookmark30)

minIni::put, [21](#_bookmark46)

Multi-tasking, [10](#_bookmark22)

Mutex, [10](#_bookmark22)

N NDEBUG, [4](#_bookmark8)

*Nil, Michael de* ~, [25](#_bookmark53)

*Index* — 32

P Petit FatFs, [27](#_bookmark55)

R Rational numbers, [1](#_bookmark3), [7](#_bookmark14), [16](#_bookmark35), [21](#_bookmark46)

Read-only support, [7](#_bookmark14), [14](#_bookmark32), [21–23](#_bookmark46)

Real number, *see* Rational number Release build, [4](#_bookmark8)

S Section (INI ﬁle), [2](#_bookmark5)

Thread-safe, [10](#_bookmark22)

Tiny-FatFs, *see* Petit FatFs

U Unicode, [8](#_bookmark17)

Unix, [4](#_bookmark8), [9](#_bookmark19), [11](#_bookmark25)

UTF-8, [8](#_bookmark17)

V *Voras, Ivan* ~, [8](#_bookmark17)

Section enumeration, *see* Enumeration

Semaphore protection, [10](#_bookmark22), [11](#_bookmark25)

Spaces (characters), [2](#_bookmark5)

std::string, [13](#_bookmark30)

strncasecmp, [4](#_bookmark8)

strnicmp, [4](#_bookmark8), [10](#_bookmark22)

T tchar.h, [8](#_bookmark17)

Temporary ﬁle, [7](#_bookmark14)

Text mode, [6](#_bookmark12)

W White space, [2](#_bookmark5)

Write access, *see* Read-only support

Write cache, [1](#_bookmark3), [7](#_bookmark14)

WriteProfileString, [1](#_bookmark3)

wxString, [13](#_bookmark30)

wxWidgets, [13](#_bookmark30)

Y *Ysboodt, Lennart* ~, [25](#_bookmark53)