minIni

Version 0.5
September 2008

minIni is a programmer's library to read and write "INI" files in embedded systems. minIni takes little resources and can be configured for various kinds of file I/O libraries.

The principal purpose for minIni is to be used on embedded systems that run on an RTOS (or even without any operating system). minIni requires that such a system provides a kind of storeage and file I/O system, but it does not require that this file I/O system is compatible with the standard C/C^{++} library —indeed, the standard library is often too big and resource-hungry for embedded systems.

Contents

The minIni library was derived in part from the article "Multiplatform .INI Files" by Joseph J. Graf in the March 1994 issue of Dr. Dobb's Journal.

The examples and programs in this manual have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose.

Introduction

minIni is a library to read and write simple configuration files with a format compatible with "ini" files. The minIni library features a small code size footprint and it requires little resources (e.g. RAM). It is therefore suitable for use in (small) embedded systems.

INI files are best known from Microsoft Windows, where functions like `GetProfileString` and `WriteProfileString` read from and write to INI files. The functions in minIni are modelled after the functions of the Windows SDK, but they are not fully compatible with them.

Although the main feature of minIni is that it is small and minimal, it has a few other features:

◇ minIni supports reading keys that are outside a section, and it thereby supports configuration files that do not use sections (but that are otherwise compatible with ini files).
◇ You may use a colon to separate key and value; the colon is equivalent to the equal sign. That is, the strings "`Name:~Value`" and "`Name=Value`" are the same.
◇ The hash character ("`#`") is an alternative for the semicolon to start a comment line.
◇ Leading and trailing white space around key names and values is ignored.
◇ Section and key name comparisons are case insensitive (similar to Microsoft Windows).
◇ Section and key enumeration are supported.
◇ You can optionally set the line termination (for text files) that minIni will use. (This is a compile-time setting, not a run-time setting.)

INI file syntax

ini files have a simple syntax with name/value pairs in a plain text file. The name must be unique (per section) and the value must fit on a single line. ini files are commonly separated into sections —in minIni, this is optional. A section is a name between square brackets, like "`[Network]`" in the example below.

Listing: **Example INI file**

```
[Network]
hostname = My Computer
address = dhcp
dns = 192.168.1.1
```

In the API and in this documentation, the "name" for a setting is denoted as the *key* for the setting. The key and the value are separated by an equal sign ("`=`"). minIni supports the colon ("`:`") as an alternative to the equal sign for the key/value delimiter.

Leading a trailing spaces around values or key names are removed. If you need to include leading and/or trailing spaces in a value, put the value between double quotes. The `ini_gets` function strips off the double quotes from the returned value.

minIni ignores spaces around the "`=`" or "`:`" delimiters, but it does not ignore spaces between the brackets in a section name. In other words, it is best not to put spaces behind the opening bracket "`[`" or before the closing bracket "`]`" of a section name.

Comments in the ini must start with a semicolon ("`;`") or a hash character ("`#`"), and run to the end of the line.

Using minIni

The first step in using minIni is making sure that it compiles. The library consists of only one C file and two header files, so the amount of configuration to do is minimal. If you cannot use the standard C/C^{++} library, there is, however, a configuration file (or "glue" file) that you must make or customize. 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 two embedded file system libaries for embedded systems —see the appendix 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 four functions that the minIni library provides to read text and values from INI files and to write text and values to an INI file. See the function reference for details.

minIni uses string functions from the standard C/C^{++} library, including one function that is not in the ANSI C standard: `strnicmp`. If your compiler does not provide this function, you can use a portable implementation in minIni by defining the macro `PORTABLE_STRNICMP` in the glue file (see below) 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.

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.

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 functions) 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`".

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 systems, 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.

If you are not using the standard C/C^{++} file I/O library, chances are that you need a different handle or "structure" to identify the storeage than the ubiquitous "`FILE*`" type. If this is the case, you must define the type that minIni uses in the glue file.

```
#define INI_FILETYPE        HANDLE
```

The minIni functions will declare variables of this type and pass these variables to sub-functions (including the glue interface functions) by reference.

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 example, `minGlue.h` could contain the definition:

```
#define INI_BUFFERSIZE      512
```

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

The temporary file is only used when writing to INI files. The minIni routines copy/change the INI file to a temporary file and then rename that temporary file to the original file. This approach uses the least amount of memory. The path name of the temporary file is the same as the input file, but with the last character set to a tilde ("~").

Unicode

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.

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.

Line termination

On Microsoft 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, the line terminator is only the LF character and on the Macintosh, it is only the CR 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 termination 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 library already handles the platform-dependent line termination for you.

However, if you wish to read and adjust the INI files with other 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"
```

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 implementation, 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 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.

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 Windows API. To list all sections, call function `ini_getsection` with an incremental "index" number until it fails. Similarly, to list all keys in a section, call `ini_getkey` 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 */
```

Function reference

ini_getkey

Enumerate keys

`ini_getkey` reads the name of an indexed key inside a given section.

Syntax: `int ini_getkey(const char *Section, int idx, char *Buffer, int BufferSize, const char *Filename)`

`Section`	The name of the section. If this parameter is *NULL* or an empty string, the keys outside any section are enumerated.
`idx`	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 pointed at by parameter `Buffer`. This is the maximum number of characters that will be read and stored.
`Filename`	The full path name of the INI file.

Returns: The number of characters read into parameter `Buffer`, or zero if no (more) keys are present in the specified section.

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_getsection`

ini_getl

Read a numeric value

`ini_getl` returns the numeric value that is found in the given section and at the given key.

Syntax: `long ini_getl(const char *Section, const char *Key, long DefValue, const char *Filename)`

`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 full path name of the INI file.

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

Notes: The number must be in decimal format. If the key is present, but it does not represent a decimal number, this function may return zero or an incorrect value.

See also: `ini_gets`, `ini_putl`

---

ini_gets Read a string

`ini_gets` reads the textual value that is found in the given section and at the given key.

Syntax:

```
int ini_gets(const char *Section, const char *Key,
             const char *DefValue, char *Buffer, int BufferSize,
             const char *Filename)
```

`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 parameter `Buffer`) if the key is not present in the INI file.
`Buffer`	The buffer into which this function will store the data read.
`BufferSize`	The size of the buffer pointed at by parameter `Buffer`. This is the maximum number of characters that will be read and stored.
`Filename`	The full path name of the INI file.

Returns: The number of characters read into parameter `Buffer`.

See also: `ini_getl`, `ini_puts`

---

ini_getsection Enumerate sections

`ini_getsection` reads the name of an indexed section.

Syntax:

```
int ini_getsection(int idx, char *Buffer, int BufferSize, const char
                   *Filename)
```

`idx`	The zero-based index of the section to return.
`Buffer`	The buffer into which this function will store the section name.
`BufferSize`	The size of the buffer pointed at by parameter `Buffer`. This is the maximum number of characters that will be read and stored.
`Filename`	The full path name of the INI file.

Returns: The number of characters read into parameter `Buffer`, or zero if no (more) sections are present in the INI file.

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

```
int s;
char name[20];
for (s = 0; ini_getsection(s, name, 20, "config.ini") > 0; s++)
    printf("%s\n", name);
```

See also: `ini_getkey`

ini_putl

Store a numeric value

`ini_putl` stores the numeric value that in the given section and at the given key.

Syntax:

```
int ini_putl(const char *Section, const char *Key, long Value,
             const char *Filename)
```

`Section`	The name of the section. If this parameter is *NULL* or an empty string, the `Key` is stored outside any section (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 full path name of the INI file.

Returns: 1 on success, 0 on failure.

See also: `ini_getl`, `ini_puts`

ini_puts

Store a string

`ini_puts` stores the text parameter that in the given section and at the given key.

Syntax:

```
int ini_puts(const char *Section, const char *Key,
             const char *Value, const char *Filename)
```

`Section`	The name of the section. If this parameter is *NULL* or an empty string, the `Key` is stored outside any section (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 any line breaking characters, such as carriage-return or line-feed characters. If this parameter is *NULL*, the function erases the key/value pair.
`Filename`	The full path name of the INI file.

Returns: 1 on success, 0 on failure.

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

See also: `ini_gets`, `ini_putl`

Appendix — example glue files

stdio (standard C/C^{++} library)

```
/*  Glue functions for the minIni library, based on the C/C++ stdio library
 *
 *  Or better said: this file contains macros that maps the function interface
 *  used by minIni to the standard C/C++ file I/O functions.
 *
 *  Copyright (c) ITB CompuPhase, 2008
 *
 *  This software is provided "as-is", without any express or implied warranty.
 *  In no event will the authors be held liable for any damages arising from
 *  the use of this software.
 *
 *  Permission is granted to anyone to use this software for any purpose,
 *  including commercial applications, and to alter it and redistribute it
 *  freely, subject to the following restrictions:
 *
 *  1.  The origin of this software must not be misrepresented; you must not
 *      claim that you wrote the original software. If you use this software in
 *      a product, an acknowledgment in the product documentation would be
 *      appreciated but is not required.
 *  2.  Altered source versions must be plainly marked as such, and must not be
 *      misrepresented as being the original software.
 *  3.  This notice may not be removed or altered from any source distribution.
 *
 *  Version: $Id: minGlue-stdio.h 3 2008-04-28 08:29:33Z thiadmer.riemersma $
 */

#define INI_BUFFERSIZE  512       /* maximum line length, maximum path length */

/* map required file I/O to the standard C library */
#include <stdio.h>
#define ini_openread(filename,file)   ((*(file) = fopen((filename),"rt")) != NULL)
#define ini_openwrite(filename,file)  ((*(file) = fopen((filename),"wt")) != NULL)
#define ini_close(file)               fclose(*(file))
#define ini_read(buffer,size,file)    fgets((buffer),(size),*(file))
#define ini_write(buffer,file)        fputs((buffer),*(file))
#define ini_rename(source,dest)       rename((source),(dest))
#define ini_remove(filename)          remove(filename)
```

EFSL (http://www.efsl.be/)

```
/*  Glue functions for the minIni library, based on the EFS Library, see
 *  http://www.efsl.be/
 *
 *  Copyright (c) ITB CompuPhase, 2008
 *
 *  This software is provided "as-is", without any express or implied warranty.
 *  In no event will the authors be held liable for any damages arising from
 *  the use of this software.
 *
 *  Permission is granted to anyone to use this software for any purpose,
 *  including commercial applications, and to alter it and redistribute it
 *  freely, subject to the following restrictions:
 *
 *  1.  The origin of this software must not be misrepresented; you must not
 *      claim that you wrote the original software. If you use this software in
 *      a product, an acknowledgment in the product documentation would be
 *      appreciated but is not required.
 *  2.  Altered source versions must be plainly marked as such, and must not be
 *      misrepresented as being the original software.
 *  3.  This notice may not be removed or altered from any source distribution.
 *
 *  Version: $Id: minGlue-efsl.h 3 2008-04-28 08:29:33Z thiadmer.riemersma $
 */

#define INI_BUFFERSIZE  256       /* maximum line length, maximum path length */

#include "efs.h"
#define INI_FILETYPE    EmbeddedFile
#define INI_LINETERM    "\r\n"    /* set line termination explicitly */


extern EmbeddedFileSystem g_efs;

#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))
#define ini_remove(filename)          rmfile(&g_efs.myFs,(char*)(filename))

/* 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)
{
  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;
}
```

FatFs & Tiny-FatFs (http://elm-chan.org/)

```
/*  Glue functions for the minIni library, based on the FatFs and Tiny-FatFs
 *  libraries, see http://elm-chan.org/fsw/ff/00index_e.html
 *
 *
 *  Copyright (c) ITB CompuPhase, 2008
 *
 *  This software is provided "as-is", without any express or implied warranty.
 *  In no event will the authors be held liable for any damages arising from
 *  the use of this software.
 *
 *  Permission is granted to anyone to use this software for any purpose,
 *  including commercial applications, and to alter it and redistribute it
 *  freely, subject to the following restrictions:
```

```
 *
 *  1.  The origin of this software must not be misrepresented; you must not
 *      claim that you wrote the original software. If you use this software in
 *      a product, an acknowledgment in the product documentation would be
 *      appreciated but is not required.
 *  2.  Altered source versions must be plainly marked as such, and must not be
 *      misrepresented as being the original software.
 *  3.  This notice may not be removed or altered from any source distribution.
 *
 *  Version: $Id: minGlue-FatFs.h 3 2008-04-28 08:29:33Z thiadmer.riemersma $
 */

#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) == 0)
#define ini_openwrite(filename,file)  (f_open((file),(filename), \
                                        FA_WRITE+FA_CREATE_ALWAYS) == 0)
#define ini_close(file)               f_close(file)
#define ini_read(buffer,size,file)    fgets((buffer),(size),(file))
#define ini_write(buffer,file)        fputs((buffer),(file))
#define ini_rename(source,dest)       f_rename((source),(dest))
#define ini_remove(filename)          f_unlink(filename)
```

Index

◇ Names of persons or companies (not products) are in *italics*.
◇ Function names, constants and compiler reserved words are in `typewriter font`.