minIni

Version 0.1

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

License

Trademarks

Copyright

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 the 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.

Notable incompatibilities of minIni are:
- `ini_gets` (replacement for `GetProfileString`) does not read all key names in a section when parameter `Key` is *NULL*. Similarly, `ini_gets` does not read all sections names when parameter `Section` is *NULL*.
- The length of a line is limited to a maximum in minIni. The maximum length is configurable (at compile-time, not at run-time) and it may be short on embedded systems.
- Quoted values are currently not supported. That is, if a value behind a key is enclosed in quotes, `ini_gets` will return that value with the quotes —it will not strip the quotes from the value, as the Microsoft Windows function `GetProfileString` does.

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).
- The colon is allowed as an alternative to the equal sign on keys. That is, the string "`Name: Value`" is equivalent to "`Name=Value`"
- Leading and trailing white space around key names and values is ignored.
- Section and key name comparisons are case insensitive (similar to Microsoft Windows).
- You can optionally set the line termination (for text files) that minIni will use.

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. 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 ("`=`").

INI files are commonly separated into sections —in minIni, this is optional. A section is a name between square brackets, like "`[Recipes]`".

There exists a few variations on INI files that minIni supports, and variations that minIni does not support. For example, minIni supports the colon ("`:`") as an alternative to the equal sign for the key/value delimiter; it also ignores spaces around the "`=`" or "`:`" delimitor, 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 bracked "`]`" of a section name.

It is commonly advised to start comments with a seminolon ("`;`") and to put any comments on a line by their own. minIni does not treat lines that start with a semicolon in a special way. That said, names of keys should not contain start a semicolon and a line that starts with a semicolon will thereby not match any valid key. Using a semicolon to add comments is therefore more a convention than part of the INI file format.

Using minIni

The first step in using minIni is building the library. To build the library, you need a configuration file (or "glue" file). 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.

After building the library, you can 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. The minIni package is minimalistic and does not support parameter substitution or "regular expression" searches. This also make the library easy to use.

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 the standard C/C^{++} library or from the operating system to perform the actual reading and writing. 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.

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.

Platform dependencies

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"
```

Function reference

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_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`