

ini

COLLABORATORS

	TITLE : ini		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		July 27, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	ini	1
1.1	ini.doc	1
1.2	ini.library/--background--	1
1.3	ini.library/ini_ChangeString	5
1.4	ini.library/ini_CheckProtection	5
1.5	ini.library/ini_Delete	6
1.6	ini.library/ini_DeleteFilename	7
1.7	ini.library/ini_ErrorString	7
1.8	ini.library/ini_GetBool	8
1.9	ini.library/ini_GetHeader	9
1.10	ini.library/ini_GetInteger	9
1.11	ini.library/ini_GetString	10
1.12	ini.library/ini_GetVariable	11
1.13	ini.library/ini_LocalConfig	12
1.14	ini.library/ini_New	12
1.15	ini.library/ini_NewConfig	13
1.16	ini.library/ini_NewFilename	14
1.17	ini.library/ini_OpenFile	15
1.18	ini.library/ini_ProtectSection	16
1.19	ini.library/ini_RemoveVariable	17
1.20	ini.library/ini_Save	17
1.21	AUTHOR	18
1.22	DISTRIBUTION	18

Chapter 1

ini

1.1 ini.doc

```
--background--
ini_ChangeString()
ini_CheckProtection()
ini_Delete()
ini_DeleteFilename()
ini_ErrorString()
ini_GetBool()
ini_GetHeader()
ini_GetInteger()
ini_GetString()
ini_GetVariable()
ini_LocalConfig()
ini_New()
ini_NewConfig()
ini_NewFilename()
ini_OpenFile()
ini_ProtectSection()
ini_RemoveVariable()
ini_Save()
AUTHOR
DISTRIBUTION
```

1.2 ini.library/--background--

PURPOSE

The ini.library was written to give the amiga user a more consistent configuration file format, and programmers a much easier way of storing config or project information on disks.

Most programs have config files to store global settings and such. These files have all different structures people in the real world (the users) cannot examine or understand; and, worse yet, often differ from version to version. We all know the hassle with those 'old->new' configuration file converters...With the ini.library this changes: you have access to a standardized ini file format known from other

operating systems such as unix, windows and so on; it allows you to save your config stuff much easier (less code), and it enables the user to read and even modify configuration data with a simple text editor (or, a wide-range field for unemployed PD programmers: using an INI editor)

As the ini.library is I-WANT-TO-BE-FREE-WARE you are ENCOURAGED to use it - its free of charge! (go ahead, do it!) and it offers much more comfort to your applications. PD dealers, diskmag, everyone, I'd like you to push'n'hype the ini.library ;-).

INI files are plain line-oriented ASCII texts that can contain three possible types of lines : COMMENTS, HEADERS and VARIABLES. COMMENTS are lines that include additional information not used by the parser. You can use these comment lines to give the user ideas about what your variables mean and what reasonable changes s(he) is allowed to make to that variables. HEADERS are identifiers for sections and are enclosed in [] brackets. A SECTION is a group of zero to infinite (?) number of variables and provides a simple but efficient means of logically grouping data. VARIABLES are simple statements in the form "<variable>=<contents>" where both <variable> and <contents> could be virtually anything - you name it.

Each INI file must start with a sequence of 4 characters ";INI" otherwise the ini.library will return the INIERROR_INVALID_INI_FILE. This was included to prevent misuse of the ini.library: imagine your program has a 'load user configuration' menu and the user tries out some binary hack or an IFF picture (you know users, don't you?! ;-). If you watch out for INIERROR_INVALID_INI_FILE, you can send the user a message of complaint and ignore its wishes -> great. Since this is a cause for many first-time problems, from version 2.1 upwards this checking can be made optional; see ini_NewConfig() for details

COMMENTS

Comments can be placed anywhere [well, almost]. If you want to have a full-line comment, place a ';' or a '*' in the first column of that line. Anything after these two characters will be ignored. If you want to place a comment at the end of a line you should use the ';' sign ONLY. Also note that if you make a comment after a variable you should probably enclose its contents in quotation marks; otherwise the whitespaces between the last character of the contents and the comment start will be part of the contents. This is no problem if you are using integer or boolean variables; it might be a problem for SOME string variables, but doesn't necessarily have to - its up to your interpretation of the input data.

Examples:

```
;INI for MUIMon release version
```

```
* another piece of wisdom  
best part of munich=ULTRAWORLD ; 100% wild techno
```

SECTIONS

Sections are used to group variables logically. You should try to place

all your data in groups so that your inifiles have a more consistent look and both you and the user can find data easier. The concept of sections can have various usages, one is to handle logic groupings for one program, another is to handle data related to different programs accessing the same INI file, yet another is to just make your INI more readable. Of course, the ini.library is flexible enough to ignore sections altogether, but you shouldn't force it! Sections cost you nothing and make for a lot better programs.

HEADERS

Headers start a new section. Headers are strings enclosed in [] brackets and can be any string you like. Everything following a header is logically assigned to that header - up to the next section header or an end-of-file.

Examples:

```
[drivers]
pc0=storage:dosdrivers/pc0
cd=devs:CDROM

[Disko Lovers,International]
best ULTRAWORLD DJs=MONIKA,TIN-A-303,BLEEP & TRIPPLE R,DJ HELL
; hi to SMC all ravers around the globe. See you on MAYDAY V

[hermeneutic philosophy revisited]
zizek=no meta language exists
```

VARIABLES

Variables are basically strings that are assigned a name. The names is what you search for, the strings is what you get -it is as simple as that. For instance, in "TABS=8" the string "TABS" is the name (also called the variable), the string "8" is its value. The interpretation of the string value is up to you. The ini.library provides three basic types of interpretation : Strings, Integers (LONGs) and boolean Values (BOOLs). You can of course add your own interpretations as you like, and this is why the ini concept is so flexible: You can virtually store ALL information you need in such a way that it is represented as an ASCII string. If you want your strings to be of a certain length, you can enclose them in brackets, for example such as in

```
user="T\"N\"I and THE DREAM TEAM"
```

where you can assign the string >T"N"I AND THE DREAM TEAM< to the variable named >user<. Note that a sequence of \" is used as an escape character in the above example.

If you have ideas for additional datatypes of general interest the ini.library should understand, please contact me. If possible, I'll include them.

Examples:

```
-----
;INI

[GLOBAL]
```

```
user="T'N'I and The Dream Team" ;from INTENSE Records
tabs=8
indent=YES
autofold=FALSE
preferences=
```

```
[DIRECTORYS]
home=work:assembler/
; add multiple directorys by using '+' or ','
include=include:+dh1:more_includes/
libs=dh2:code/libs/asm/+lib:+disko:love
```

The above example shows you some ideas of inifiles: Two groups are part of this inifile: GLOBAL and DIRECTORYS. Each section can (but doesn't have to) have different variables, each variable can be any string enclosed in quotation marks or from the = sign to the end of the line. If you want to add comments, enclose the string in quotes, and add a comment like in the "user=" line above. Note that the feature described as "add multiple directorys by ..." is a description of an add-on datatype used by the application for this INI file and NOT provided by the ini.library itself.

Now you should be able to read inifiles. More complex features such as section protection and file location are described in the "programmers.guide" document which should be part of this distribution.

HINTS

If you decide to use the ini.library -WHICH I STRONGLY HOPE YOU DO- you should always check for error returns - you may never know when they come in handy for you.

If you save your files, make use of the grouping feature; that is, at least provide some GLOBAL section so that the user knows what he's up for. It makes INIfiles more readable and doesn't cost you anything.

LIMITATIONS

Yes, there are some.

- * currently no whitespaces before and after the "=" are allowed. This means that the two following lines are different for the ini.library parser functions

```
ultraworld=great
ultraworld = great
```

There is a simple workaround for this: just do your own variable and contents parsing (explicit datatypes, see above). Future versions WILL include a fix on that, I PROMISE.

- * the ini.library uses approximately twice as much memory as the ini file is in size (less for larger files). This is not a problem for most users, because inifiles seldom grow over a size of some 10-20 kb (even on windows I haven't seen a inifile larger than 25kb up to this day), and many applications not even reach a 5kb limit. (You can write a LOT variables in a 5 kb file).

However, this memory is needed only while your parser is active, (i.e. between calls to `ini_New()` and `ini_Delete()`). Afterwards, only the library code (5kb) is kept in memory.

1.3 ini.library/ini_ChangeString

NAME

`ini_ChangeString` -- change an existing string for a variable

SYNOPSIS

```
error = ini_ChangeString( parser, header, variable, value )
d0      A0      A1      A2      A3
```

```
INIPARSER ini_ChangeString(INIPARSER *,STRPTR,STRPTR,STRPTR);
```

FUNCTION

Changes the contents of a variable to a new value. If you have made changes to the data represented in an inifile (e.g. your internal configuration structure) you can use this function to fix those changes and then write back your file with `ini_Save()`. If the variable already existed, its contents will be replaced by the new contents; if it didn't exist, it will be appended to the section it is assigned to. Unless you use explicit saving of ini-files you **MUST** use `ini_ChangeString()` to change the contents of variables, ***NEVER*** try to do it on your own (its quite tricky, actually!).

INPUTS

`parser` - pointer to a valid `INIPARSER`
`header` - section header name
`variable` - section variable name
`value` - new contents

RESULTS

`error` - `INIPARSER` as defined in `libraries/ini.h`

NOTE

old end-of-line comments will be stripped by this function. You can explicitly set one by using contents such as

```
char *newContents = "\"blabla\" ;comment "; or
```

```
dc.b  '"blabla" ;comment',0
```

SEE ALSO

`ini_RemoveVariable`, `ini_Save`

1.4 ini.library/ini_CheckProtection

NAME

`ini_CheckProtection` -- check if a section has a valid protection

SYNOPSIS

```
error = ini_CheckProtection( header, password )
d0          A0          A1
```

```
INIEERROR ini_CheckProtection( INILINEINFO *, STRPTR )
```

FUNCTION

This function checks if it can find a "\$code" variable in the given section and if that code matches the password given as an argument. The password is case sensitive and must match *EXACTLY* the one used to code that particular section (including whitespaces and so on). It is up to the caller to decide what to do if a section is not properly coded : you can make it a fatal problem or just ignore it - whatever you like. This means, the ini.library will load files with or without invalid passwords - YOU have the "power" to handle this situation (actually, this was done because it is not a feature of the normal inifile "standard", it's a addon bonus of the ini.library)

INPUTS

header - pointer to a INILINEINFO structure for a section header
(i.e. ili_Flags must be INIFLAG_HEADER)
password - pointer to the password (assumably) used to protect this section

RESULTS

error - INIEERROR as defined in libraries/ini.h; you should look out for INIEERROR_NONE = good, INIEERROR_INVALID_PASSWORD = bad

SEE ALSO

ini_ProtectSection

1.5 ini.library/ini_Delete

NAME

ini_Delete -- delete INI parser

SYNOPSIS

```
error = ini_Delete( parser )
D0          A0
```

```
INIEERROR ini_Delete( INIPARSER * );
```

FUNCTION

Free all memory used by an INI parser and reset all accompanying internal data structures. You should ALWAYS call this function after you have finished using the INI file so that you don't lose any precious memory. You must call ini_Delete() even if ini_New() wasn't able to create the parser for you (there still might be some memory somewhere only ini_Delete can take care of)

WARNING

This function does not arbitrate for access to the parser. The calling task must be the owner of the involved parser.

INPUTS

parser - a pointer to a parser structure initialized with ini_New()

RESULTS

error - INIERROR as defined in libraries/ini.h

SEE ALSO

ini_New

1.6 ini.library/ini_DeleteFilename

NAME

ini_DeleteFilename() -- destructor for ini_NewFilename()

SYNOPSIS

```
ini_DeleteFilename( filename )
    DO
```

```
void ini_DeleteFilename( STRPTR )
```

FUNCTION

This function will deallocate memory associated with a filename generated by ini_NewFilename()

INPUTS

filename - a filename returned from ini_NewFilename

NOTE

after calling ini_DeleteFilename() your "filename" variable is invalid : DO NOT USE IT ANY LONGER!

RESULTS

none

SEE ALSO

ini_NewFilename()

1.7 ini.library/ini_ErrorString

NAME

ini_ErrorString -- find error message to an INIERROR

SYNOPSIS

```
message = ini_ErrorString( error )
    DO          DO
```

```
STRPTR ini_ErrorString( INIERROR )
```

FUNCTION

Used to find an ini.library error message. Currently the error strings are in english but a localized version is planned for the next update of this library, so you should use ini_ErrorString() when printing warnings instead of your own error strings. Also, if

you use `ini_ErrorString()` you automagically can handle errors that result from higher library versions.

INPUTS

`error` - some `INIEERROR` code returned by one of the `ini.library` functions

RESULTS

`message` - the error message for that error code (soon to be localized)

If there was no error message (because of `INIEERROR_NONE` or an invalid error code) `ini_ErrorString()` will return an EMPTY string (`" "`) and not `NULL`, so you can always do something like

```
printf("%s\n", ini_ErrorString( ini_Whatever(...) ) );.
```

NOTE

You are NOT allowed to modify this string, if you need to do so, make a copy of the string and use that copy instead.

1.8 ini.library/ini_GetBool

NAME

`ini_GetBool` -- find a boolean variable

SYNOPSIS

```
error = ini_GetBool(parser,head,var,defaultBool,target)
d0      A0      A1      A2      D0      A3
```

```
INIEERROR ini_GetBool(INIPARSER *,STRPTR,STRPTR,BOOL,BOOL *);
```

FUNCTION

Used to find a boolean in an inifile. If the value cannot be found for any reason, a default value is used. The variable should be

```
"YES",
"Y",
"ON" or
"ACTIVE"
```

if the returned value is to be `TRUE`, otherwise `FALSE` will be returned. Future Versions of the `ini.library` will probably include locale-dependant IDs in addition to those above. You can use `ini_GetBool()` very good to handle boolean flags in your configuration - and it is a lot more readable than other solutions.

INPUTS

`parser` - pointer to an `INIPARSER` opened by `ini_New()`

`head` - section header name

`var` - section variable name

`defaultBool` - default value if boolean value not found

`target` - adress of target Bool

RESULTS

`error` - `INIEERROR` as defined in `libraries/ini.h`. Actually, you can normally ignore this error code, because you must provide a

default value anyway.

SEE ALSO

ini_GetInteger, ini_GetString

1.9 ini.library/ini_GetHeader

NAME

ini_GetHeader -- find a section header

SYNOPSIS

```
header = ini_GetHeader( parser, headerString )
D0      A0  A1
```

```
INILINEINFO *ini_GetHeader( INIPARSER *, STRPTR )
```

FUNCTION

This function is used to locate a section in an INI file by looking for the section header. If you are looking for a specific variable, ini_GetString() is probably the better choice, but you can do your own parsing of INI sections, e.g. if you have 'intelligent' sections; that is sections with flexible concepts or whatever. You could use ini_GetVariable() in that case.

INPUTS

parser - a pointer to a parser structure initialized with ini_New()
headerString - name of the section header you are looking for, without enclosing brackets (ie. "global")

RESULTS

header - pointer to an INILINEINFO or NULL if that header could not be found. This can be the case either if the INI file does not include that particular section, or (worse) the INI parser wasn't able to complete its job (as you know, all ini.library functions are save to use). If this is NON-NULL, you can use it as an argument to ini_GetVariable() or you can do your own section looping (see "advanced concepts" in the "programmers guide")

SEE ALSO

ini_GetVariable, ini_GetString, ini_New

1.10 ini.library/ini_GetInteger

NAME

ini_GetInteger -- find a long integer variable

SYNOPSIS

```
error = ini_GetInteger(parser,head,var,defaultLong,target)
d0      A0      A1  A2  D0      A3
```

```
INIEERROR ini_GetString(INIPARSER *,STRPTR,STRPTR, LONG, LONG *);
```

FUNCTION

Used to find an integer in an inifile. If the integer cannot be found for any reason, a default value is used. The ini section should contain a string representing a long decimal value. If the string starts with a \$ sign, a hexadecimal value is expected, % indicates binary digits and finally @~sets an octal number. Therefor, the following four entries will all return the same number

```
decimal=123
octal=@173
binary=%1111011
hex=$7B
```

The prefixes are compatible to most assemblers (now you know which language the ini.library was written in ;-)) but a future update will probably include C-style prefixes (0x and so on).

INPUTS

parser - pointer to an INIPARSER opened by ini_New()
 head - section header name
 var - section variable name
 defaultLong - default value if integer not found
 target - adress of target long

RESULTS

error - INIERROR as defined in libraries/ini.h. Actually, you can normally ignore this error code, because you must provide a default value anyway.

SEE ALSO

ini_GetBool, ini_GetString

1.11 ini.library/ini_GetString

NAME

ini_GetString -- find a string variable

SYNOPSIS

```
error = ini_GetString(parser,head,var,defaultStr,target,sizeofTarget)
D0          A0      A1      A2      A3          D0      A4
```

```
INIERROR ini_GetString(INIPARSER *,STRPTR,STRPTR,STRPTR,STRPTR,WORD);
```

FUNCTION

Used to find a string in an inifile. If the string cannot be found for any reason, a default string is used; so unless you desperately rely on a string to exist as part of an inifile, you can call this function without even checking for error returns. You can add a series of ini_GetString() statements for all strings you need to check without having to worry if they exist or not.

INPUTS

parser - pointer to an INIPARSER opened by ini_New()
 head - section header name

var - section variable name
 defaultStr - default string to use (NULL is possible)
 target - target string
 sizeofTarget - size of target string. ini_GetString() promises not to overwrite its bounds AND to keep a zero byte at the end, so actually a maximum of (sizeofTarget-1) characters will be written.

RESULTS

error - INIERROR as defined in libraries/ini.h. Actually, you can normally ignore this error code, because you must provide a default value anyway.

SEE ALSO

ini_GetBool, ini_GetInteger

1.12 ini.library/ini_GetVariable

NAME

ini_GetVariable -- find a section variable

SYNOPSIS

```
variable = ini_GetVariable( header, variableName )
D0          A0          A1
```

```
INILINEINFO *ini_GetVariable( INILINEINFO *, STRPTR )
```

FUNCTION

This function is used to locate a variable following a given header (or, for that matter, any given INILINEINFO). You can use this function if you want to manually parse a section, if you want to find out if there are more variables of a given name following a specific variable, or for anything else you could think of. Note that you cannot pass the INIPARSER as function arguments, use (parser->table.ln_Head) if you have to do so.

INPUTS

parser - a pointer to an INILINEINFO structure (from your INIPARSER.table) or -preferably- as returned by ini_GetHeader()
 varibaleString - name of the variable you are looking for, without enclosing brackets and "=" (ie. "variable")

RESULTS

variable - pointer to an INILINEINFO or NULL if that variable could not be found. This can be the case either if the INI file does not include that particular section, or (worse) the INI parser wasn't able to complete its job (as you know, all ini.library functions are save to use).

NOTE

ini_GetVariable() doesn't promise the variable to be a member of the section you specified with your header. For example,

```
head = ini_GetHeader( parser, "HEAD ONE" );
info = ini_GetVariable( head, "VARIABLE" );
```

would find the "VARIABLE=FOUND" entry in the "HEAD TWO" section if the inifile looked something like this :

```
[HEAD ONE]
size=3551

[HEAD TWO]
VARIABLE=FOUND
```

This is A FEATURE, NOT A BUG ;-), yes, because you normally will use `ini_GetVariable()` to loop through the complete inifile.

SEE ALSO
`ini_GetVariable`, `ini_GetString`, `ini_New`, `ini_ParseSection`

1.13 ini.library/ini_LocalConfig

NAME
`ini_LocalConfig()` -- get the local configuration

SYNOPSIS
`config = ini_LocalConfig()`

INICONFIG `*ini_LocalConfig(void)`

FUNCTION
This function will return the local configuration if one is defined for the current task. Local configurations are "task sensitive", which means that one task can have one local config. `ini_LocalConfig()` will NOT fall back to the global config, so if you want to read the current config you would probably write something like

```
if( ( config = ini_LocalConfig() ) == NULL )
    config = IniBase->GlobalConfig;
```

INPUTS
none

RESULTS
config - local configuration or NULL if none exists

SEE ALSO
`ini_NewConfig()`, `ini_DeleteConfig()`

1.14 ini.library/ini_New

NAME
`ini_New` -- open INI parser

SYNOPSIS

```
error = ini_New( fileName, parser )
D0      A0      A1
```

```
INIEERROR ini_New( STRPTR, INIPARSER * );
```

FUNCTION

Open an inifile and parse its contents. After successfully calling an inifile with this function you can examine its contents using the functions provided (eg.ini_GetString()) or manually. You must call this function before calls to any other ini.library functions. Don't even think about creating an INIPARSER yourself.

WARNING

This function does not arbitrate for access to the parser. The calling task must be the owner of the involved parser. Use semaphores if you want to share parser information with other tasks.

You should always call ini_Delete() on the parser EVEN IF ini_New() failed. Otherwise some internal memory might get lost somewhere

INPUTS

fileName - name of the input file. This is currently a plain filename with or without pathname; however you should try to use only plain filenames ("test.ini" instead of "dhl:somewhere/test.ini") so that the ini.library can fix inifiles to S:, INI: or wherever the user wants them located. See "locating files" in the "programmers guide" for details.

parser - pointer to an empty INIPARSER structure. C programmers simply add the address of an INIPARSER variable, ASM programmers can use a (DS.B ip_SIZEOF) statement (See example TEST.ASM)

RESULTS

error - INIEERROR as defined in libraries/ini.h. Proceed ONLY if error = INIEERROR_NONE, else you can forget about it.

SEE ALSO

ini_Delete

1.15 ini.library/ini_NewConfig

NAME

ini_NewConfig() -- setup a local configuration

SYNOPSIS

```
error = ini_LocalConfig( tagList )
d0      A0
```

```
INIEERROR ini_LocalConfig( struct TagItem * )
```

FUNCTION

This function creates a local configuration for your program. If one already exists it will be modified to use the new values. See "ini.introduction" for more details on the concept of local and global ini.library configurations

NOTE

Sorry, no vararg-stub for C-programmers (yet?); I didn't want to have to write a linklib for only one function...

INPUTS

tagList - a list of tag items. Currently defined tags are :

ICFG_ENABLE_S

ti_Data is BOOL, default = TRUE

ICFG_ENABLE_INI

ti_Data is BOOL default = TRUE

ICFG_ENABLE_ENV

ti_Data is BOOL, default = TRUE

ICFG_ENABLE_USER

ti_Data is BOOL, default = TRUE

ICFG_INIMASK

ti_Data is BOOL, default = FALSE. If set to TRUE, the ini.library no longer requires the ";INI" at the start of an input file.

ICFG_ASSIGN

user-assign directory, ti_Data is STRPTR to a static memory! (i.e. the ini.library will not copy name name of the user-assign so you MUST keep it in memory yourself!!!!!! this may change for future releases but right now its a requirement)

RESULTS

error - INIERROR as defined in libraries/ini.h;

SEE ALSO

ini_LocalConfig(), ini_DeleteConfig()

1.16 ini.library/ini_NewFilename

NAME

ini_NewFilename -- build a config-sensitive INI filename

SYNOPSIS

```
newname = ini_NewFilename( filename )
d0          A0
```

```
STRPTR ini_NewFilename( STRPTR )
```

FUNCTION

This function builds a complete path+filename for a simple filename based upon the local (or global) configuration. The problem is simple: a file named "application.ini" could be located in S:, in INI:, in ENV:, in a user-defined directory or the current directory. BASED UPON THE CURRENT CONFIGURATION this function will build a correct filename

for you. Example:

```
ini_NewFilename( "application.ini" )
```

could return

```
"s:application.ini"
"ini:application.ini"
"env:application.ini"
"<user>:application.ini" or just
"application.ini"
```

You must call `ini_DeleteFilename()` on a filename returned by this function.

INPUTS

filename - the basic filename; should NOT contain any path information (except for subdirectories: if you are using multiple INIs grouped in a subdirectory (i.e. "tool/users.ini" -> "ini:tool/users.ini"))

RESULTS

newname - path + filename, or NULL for OUT_OF_MEMORY error

SEE ALSO

`ini_DeleteFilename()`

1.17 ini.library/ini_OpenFile

NAME

`ini_OpenFile()` - open an INI-file

SYNOPSIS

```
file = ini_OpenFile( filename, mode, parser )
d0          d1          d2      a0
```

```
BPTR ini_OpenFile(STRPTR,WORD,INIPARSER *);
```

FUNCTION

This function opens an INI-File just like `dos/Open()` would do. However, it does a little bit more: if you use it in `MODE_NEWFILE` and have a valid parser structure, `ini_OpenFile()` will open the file in that directory where the original parser was located. The problem is this: because your program doesn't know where the INI file was loaded from (it could be `INI:`, it could be `S:`, it could be anywhere in the known galaxy) you wouldn't normally know where to save it back to (unless of course you don't need explicit saving and use `ini_Save()` instead of all the fuss).

INPUTS

filename - filename without path or anything
mode - just like `dos/Open()`, `MODE_NEWFILE`, `MODE_OLDFILE` etc.
parser - parser structure. Because the parser has to be initialized, this probably restricts use of `ini_OpenFile()` for the general public to the situation described above: explicitly saving an ini-file

RESULTS

file - just like dos/Open() a pointer to a filehandle you can Read() or Write() to (or whatever). You must Close() it yourself!

1.18 ini.library/ini_ProtectSection

NAME

ini_ProtectSection -- protect an INI file section

SYNOPSIS

```
error = ini_ProtectSection( parser, header, password )
d0                A0        A1        A2
```

```
INIEERROR ini_ProtectSection( INIPARSER *, INILINEINFO *, STRPTR )
```

FUNCTION

This function protects a INI file section by adding a new (or updating an existing) "\$code" variable to contain a coded value. The coding mechanism uses all data given in variables of a section plus a special password known only to the caller of ini_ProtectSection. Thus, if a user a) changes contents of variables or b) tries to use this function with an invalid password the "\$code" variable will become invalid. You can use ini_CheckProtection to see if the "\$code" variable is valid, so there you have your protection mechanism!

In general, if you intend to protect a section, you should do the following :

- * call ini_CheckProtection() : if the section is invalid, send some error message (or do-what-you-like)
- * edit the protected section
- * call ini_ProtectSection()
- * call ini_Save()

Of course, you should probably skip ini_CheckProtection() during the development of your application, since you may find the need to quickly change var contents while debugging and so on. Once working you can use protected sections to contain "read-only" data you still want to make public (such as for debugging purposes, or for pure fun, or for making people jealous and encouraging hackers to take a deeper look at your code)

The concept for protected sections is an add-on to the INI standard made by the ini.library so you shouldn't expect it to work if you port code to other OS such as Windows.

NOTE

The password is case sensitive and must match EXACTLY for calls to ini_ProtectSection()/ini_CheckProtection(); i.e.

```
"ultraworld" != "Ultraworld" != "ultraworld "
                ^                ^
```

INPUTS

parser - pointer to the INIPARSER structure
 header - pointer to a INILINEINFO structure for a section header
 (i.e. ili_Flags must be INIFLAG_HEADER)

password - pointer to the password you want to use for protection

RESULTS

error - INIERROR as defined in libraries/ini.h; you should look out for INIERROR_NONE = good, INIERROR_INVALID_PASSWORD = bad

SEE ALSO

ini_CheckProtection

1.19 ini.library/ini_RemoveVariable

NAME

ini_RemoveVariable() -- remove an existing variable

SYNOPSIS

```
error = ini_RemoveVariable( parser, header, variable )
d0          A0          A1          A2
```

```
INIERROR ini_RemoveVariable( INIPARSER *,STRPTR,STRPTR );
```

FUNCTION

This function removes an existing variable from an INIPARSER. You can use this if you don't need a particular variable anymore; however you don't have to kill unused variables, because they are ignored anyway based upon your read requests. DON'T ever remove a variable manually.

INPUTS

parser - pointer to an INIPARSER
header - section header name
variable - section variable name

RESULTS

error - INIERROR as defined in libraries/ini.h

SEE ALSO

ini_Save, ini_ChangeString

1.20 ini.library/ini_Save

NAME

ini_Save -- save an inifile back to disk

SYNOPSIS

```
error = ini_Save( fileName, parser )
d0          A0          A1
```

```
INIERROR ini_Save( STRPTR, INIPARSER * )
```

FUNCTION

Save a parser structure back to an inifile. If you have made changes to an INIPARSER (e.g. by calling ini_ChangeString()), you

can save the complete inifile to disk using this function. If you have to make lots of changes on the input data, for example if you save project file information there, you could explicitly save the file yourself: open it using `ini_OpenFile(..,MODE_NEWFILE,..)`, then `dos/FPrintf()` all the contents and finally `dos/Close()` it. You have to decide yourself what you prefer; making changes with `ini_ChangeString()` and `ini_Save()` makes more readable code and is a lot safer (and preserves any comments the user has made); exporting your internal data as inifiles is probably more straight-forward but not quite as bullet-proof. Whatever.

INPUTS

`fileName` - name of inifile to create. No checking is done if the file already exists; so you must make some provision for this yourself if you think this could be important.
`parser` - pointer to an `INIPARSER` to write back to disk

RESULTS

`error` - `INIEERROR` as defined in `libraries/ini.h`

SEE ALSO

`ini_ChangeString`, `ini_RemoveVariable`

1.21 AUTHOR

Send your bug reports, ideas, love letters, fresh'n'phunky vinyl to

Gerson Kurz
Karl Köglspurger Str.7 / A303
(yes, its APPARTMENT 303, not Roland TB 303 ;-)
80939 München
West Germany

This tool is dedicated to the ****ULTRAWORLD**** (Fuck the Sperrstunde!)

1.22 DISTRIBUTION

**** The INI.LIBRARY is I-WANT-TO-BE-FREE-WARE ****
dedicated to the global house & techno scene
written by Gerson Kurz

The `ini.library` may be used by and **FREELY** distributed with any application be it commercial or public domain. There are no pagan users fees, Trump-esque licenses, or other forms of rabid capitalist trickery associated with using this library and the example files. You do not even have to acknowledge the secret of your superb and efficient config handling you gain by using the `ini.library`. The only limitation is that you may not alter the actual executable of the `ini.library`, nor sell the product and its examples as a distinct product (i.e. represent it as such). I don't give any guarantee for the fitness of the `ini.library` for any purpose : use at own risk.
