Using ”profile” INI files

in the MapBasic environment

Jacques Paris

jacques@paris-pc-gis.com

August 2001

Contents

Because of the very specific way this document deals with INI files, it will include only the basic concepts necessary in this context. Only tested solutions will be offered; if alternatives exist, I will be happy to know about them in order to enhance this presentation.

Origins of the INI files

When I submitted my first draft on the “INI files” to Bill Thoen, he wrote me back: “These are actually called "profile files" by Microsoft, and they are based on the UNIX idea of .profile files; text files used to store environmental variables (and yes, in UNIX these files are called .profile with a period as the first character in the file name).” I noticed however that some MS help files referred to them also as “initialization” files. I will simply use the term “INI files”.

I have made a large use of them in the last two years, particularly within the scope of the Multi Lingual Compatibility project to hold language strings, and inserting in my code subs written by Bill to streamline their creation and access to them. I have found more recently a larger use for them in several MB applications and I would like to share my experiences and start a new knowledge base on the subject.

Several API functions deal with INI files. One set works only and directly on the WIN.INI file and the file name is not part of the arguments (GetProfileInt, GetProfileString and WriteProfileString). The functions that deal with user’s ini files contain “Private” in their names (GetPrivateProfileInt, GetPrivateProfileString and WritePrivateProfileString.

Finally, a pair of them are used to read/write only strings and to add a certain level of security by attaching a checksum to each key when it is written in; when a key is read, the checksum of the extracted value is compared to that stored in the ini file. GetPrivateProfileStruct and WritePrivateProfileStruct can be substituted to GetPrivateProfileString and WritePrivateProfileString without apparent restrictions.

One will notice that there are two read (Get…) functions for one write. If one can read directly an integer or a string, one can only write a string. Number reading is somewhat restricted because it cannot deal with float decimal numbers. I have thus chosen to limit the functions that I will use in this document to one write and one read, both …String, with the consequence of having to apply a val() conversion on the returned string when it represents a numerical value.

A special chapter will be devoted to the “delete” potentials of the write function. It is indeed that function that can delete entire sections, or selected keys of a given section. Its implementation in MapBasic requires some “trickery” that will be explained in detail.

Some precautions

As any topic new to you, it is a subject to handle with some care. There are some precautions to be taken and some limitations to be aware of. The main precaution to take results from the way changes to the ini files are carried: they are instantaneously written to the file (or at least to its image stored in memory as explained below). There is no automatic backing up potential, no undo or revert. The user may be wise to make a **backup of his ini file** before doing any edit on it, mainly deletes, before what is erased will be unretrievable.

One limitation is expertly presented by Bill Thoen as follows:

> “A subtle issue you should be aware of is that Windows profile files are completely read into memory when first referenced, and the entire block is written back to the file only when Windows thinks its necessary. The subtle issue is that you can make a write that does not get updated in the disk file immediately, and so it's possible that another read from a new process or from another section of code in a large application, will not receive the latest values of the keys.
>
> I have encountered this problem when I used a MapBasic app to create a front end for a C program's profile file. I'd set the key values and then launch the other program, and it did not get the new parameters. I had to actually end the MB program to force it to write the new values to disk. There are other events that cause a write (like Windows deciding to swap memory blocks), but the programmer can't force them (at least not to my knowledge.) This isn't a problem for applications that are the only one that accesses a profile's keys in a session, but watch out or this in multi-module applications because it causes very odd "bugs." “

INI file structure and contents definition.

An INI file is a simple text file, structured in SECTIONS, each containing KEYS.

There must be at least one section that is identified by a word between square brackets, e.g. [GENERAL]. One will use multiple sections to make handling of long lists of keys easier by breaking them down or when alternative key lists are necessary.

The beginning of a **section** is thus defined by its name written between [ ]. The order of the sections in the file is the order in which they have been written in the file originally and is not affected by posterior writes (addition of new sections, new keys or changes of value to existing keys). If a write to an ini file contains the name of a section not present in the file, that section will be added at the end of the file.

The order of the sections is irrelevant for reading.

No duplication of section names is allowed.

A **key** is made up of the key name followed by the equal sign and a value that must be written between double quote signs “ “ only if one wants to preserve whitespace at the end of the value. The order of the keys within a section is the order in which they have been inserted in the file and it is not affected by changes of value to existing keys. If a write to an ini file contains the name of a keynot present in the file, that key will be added at the end of the section.

The order of the keys is irrelevant for reading.

No duplication of key names is allowed within a section.

A paraphrase of these rules would be that a combination section name + key name must be unique in order to be addressed directly and without ambiguity.

An example:

```
                                        Comments (not in the file)

[SideBuffers]                           section name
Offset=5                                    key 1
Width=50                                    key 2
Ranges=3                                    …
Bruschem=2
Numschem=2
                                        Blank line(s) allowed
[SCHEME1]                               section name
nclasses=5                                  key 1
value1=WS                                   key 2
brush1=(19,0,-1)                            …
value2=WL
brush2=(20,0,-1)
value3=WM
brush3=(19,16711680,-1)
value4=WH
brush4=(20,16711680,-1)
value5=RV 1
brush5=(36,255,-1)

[SCHEME2]
nclasses=8
….
```

This type of file structure does not allow for **indexed variables** because the "read/write INI" functions deal only will a single line (section + key combination). Therefore, arrays cannot be accessed globally and each array member must be dealt with separately. The way to deal with them is to "compose" INI names by concatenating the "name" with a sequential number, as can be seen in this example for sections (SCHEME_) as well as for keys (value_ and brush_). I have omitted here the usual index parentheses to make code writing easier, but if anyone wanted to include them, he could do it without problem.

"Read and Write INI" functions.

The function that allow to write to an INI file or to read from it are contained in the Kernel32.DLL of Windows.

> If one needs to work on a 16-bit Windows platform, it would require the presence of Kernel.DLL. As the arguments are exactly the same, what will be said about the 32-bit functions will apply to the older platform; only the declare will differ. Details about the possibility of using either of the platforms are given below "Adjustments for using both 16/32 bit platforms".

They need to be "**declared**" in your application MB code.

```
Declare Function WritePrivateProfileString32 Lib KERNEL32 alias
                                                    “WritePrivateProfileStringA"
    (ByVal lpApplicationName As String,
    ByVal lpKeyName As String,
    ByVal lpString As String,
    ByVal lplFileName As String) As Smallint

Declare Function GetPrivateProfileString32 Lib KERNEL32 alias
                                                    "GetPrivateProfileStringA"
    (ByVal lpApplicationName As String,
    ByVal lpKeyName As String,
    ByVal lpDefault As String,
    lpReturnedString As String,
    ByVal nSize As Smallint,
    ByVal lpFileName As String) As Smallint
```

Argument definitions

	used in	
lpApplicationName	W/R	SECTION name
lpKeyName	W/R	KEY name
lpString	W	value to be written in INI file A string constant must be between “ “. A numeric constant does not require any special treatment. Any other situations, some conversion to a string is required (e.g. a logical constant must be converted to some string either 0/1, or “F” /“T”)
lplFileName	W/R	name of the INI file
lpDefault	R	value returned if the queried key is not located
lpReturnedString	R	value of the queried key as a string
nSize	R	size of the returned value in characters

All the arguments are defined “ByVal” with the exception of “lpReturnedString” that can be used only as an output from the read function, forbidding the passing of that argument as a “value”.

Returned values.

These functions return directly a single value that is the indication of the success/failure of the operation. The function value is zero if the operation was a failure, >0 in case of success.

Other values are also “returned” by the way of some of the arguments. It is particularly the case of the “lpReturnedString” of the read function, value that is the raison d’être of the function. Notice that “nSize” could be also a returned value but has no practical use in this document as an output

In a pragmatic way, the user is interested to know first if the function worked (return value of the function itself), then, in the particular case of the read function, whether the

queried string was located and its value found. In this last case, a specific value is attributed to “lpReturnedString”, that specified by “lpDefault”. (see below)

It is always preferable to terminate the application when one function fails one way or the other.

Default returned value

One of the arguments of the read function is “lpDefault”, the default string that will be assigned to the “lpReturnedString” if for some reason the function fails to locate the queried key. It should be something that one is sure not to find in the values that could be stored as such in the keys of the INI file. As that argument is passed BYVal, that value can be either stored in a variable used as the argument or a constant passed directly to the function or via a define statement.

```
        Dim vDef as string
        vDef = ”?”
        rStatus = GetPrivateProfileString32 ( , , vDef, , )
or
        rStatus = GetPrivateProfileString32 ( , , “?”, , )
or
        define RET_DEFAULT "?"
        rStatus = GetPrivateProfileString32 ( , , RET_DEFAULT, , )
```

If that argument is frequently used in the code, the middle solution is not recommended because if a change has to be made in the default string, the correction will have to be done too many times. The choice between the first and last solutions is a question of personal choice, because if the naming of the variable or of the constant is done judiciously (short and understandable), neither has a marked advantage.

Required code for using the read/write functions

Beside the declare statements, the MB application should contain some code elements necessary to answer to some specificities of the functions.

A smallint variable must be dim’ed to call the functions and its result used to give a clear answer on the success of the operation directly (writing) or via the returned string value (reading)

writing

```
Dim rStatus as SmallInt

rStatus = WritePrivateProfileString32 (…)
if rStatus = 0 then
        Note “Unable to write to file”
exit sub end if
```

reading

vDefault is the string that will be finally returned if the function fails or is unable to locate the queried key. (see “Default returned value”).

Val_Returned is the string that is the principal output of the function. As the API must write the key value to this string but cannot allocate the space on its own, that variable must be initialized to the size of the maximum expected in the INI file. This string variable will be first filled characters (255 of any sort) then passed to the function that will overwrite it with the contents of the key.
The size of the string must also be passed to the function (255) and will be returned as the size of the key string. The simplest way is to use for this argument the function “len(Val_Returned)”.

ReturnedString is the argument name used in calling the function (see lpReturnedString in “Argument definitions” above)

```
Dim rStatus as integer
Dim vDefault, Val_Returned as string
Vdefault = ”?”
Val_Returned = String$(255, "X")
rStatus = GetPrivateProfileString32 (…)
if rStatus = 0 then
        Val_Returned = vDefault
  else
        Val_Returned = ReturnedString
end if
if Val_Returned = vDefault then
        Note “Unable to read from file”
exit sub end if
```

Adjustments for using both 16/32 bit platforms

1 - add following function definitions

```
Declare Function WritePrivateProfileString16 Lib KERNEL alias WritePrivateProfileString"
    (ByVal lpApplicationName As String,
    ByVal lpKeyName As String,
    ByVal lpString As String,
    ByVal lplFileName As String) As Smallint

Declare Function GetPrivateProfileString16 Lib KERNEL alias "GetPrivateProfileString"
    (ByVal lpApplicationName As String,
    ByVal lpKeyName As String,
    ByVal lpDefault As String,
    lpReturnedString As String,
    ByVal nSize As Smallint,
    ByVal lpFileName As String) As Smallint
```

2 - replace the "if str$(systeminfo(14)... " statement by

```
if str$(systeminfo(14))>"2" then          'makes sure of the MI_Platform.
        note "This application runs only in a Windows environment."
        end sub
```

```
        end if
```

3 - for each read or write use a "do case" as follows

```
        do case SystemInfo(14)
         case 1
              nStatus = GetPrivateProfileString16 (Section, Key, sDefault, sRetVal,
                                                        Len(sRetVal), File)
'or           nStatus = WritePrivateProfileString16 (sSection,sKey, """" + sValue + """", sFile)
        case 2
              nStatus = GetPrivateProfileString32 (Section, Key, sDefault, sRetVal,
                                                        Len(sRetVal), File)
'or           nStatus = WritePrivateProfileString32 (sSection,sKey, """" + sValue + """", sFile)
        end case
```

How to delete sections and keys

The write function can also be used for deleting sections or keys. The argument list for deleting a section would be (“section_name”,NULL,””,inifile_name) and for a key (“section_name” ,”key_name”,NULL,inifile_name). The problem in MapBasic resides with the NULL value, which is by definition the character Zero, not the value zero. Here is again Bill Thoen explaining the situation:

> “It's a hard one to slip past MapInfo, since passing ""+Chr$(0) gets handled by MapInfo as "" BEFORE it passes it to the API. You see, in memory a string is actually an address that points to some allocated memory. That way the string is always visible at the same place in memory, but its contents can be redefined dynamically. The difference between an empty string and a null, is that for an empty string, the pointer in the string's main address points to a place in memory where only one byte (a string terminator) is found, thus it means that it is an empty string. For a NULL, the pointer in the string's address points to nowhere, i.e. it equals zero. MapInfo's strings ALWAYS point to somewhere, even if that somewhere is an empty string.”

Bill envisaged several alternatives of which I was able to implement the first one; “What might work is to declare an alias to the function and make that parameter an integer instead of a string.”

And here is the solution. First we must declare 2 new aliases because the required input for section and key deletions is slightly different. I renamed these aliases very clearly as PrivateProfileDeleteSection and …DeleteKey. Presenting them with the standard write function helps understanding where the changes are made (bold italics)

```
Declare Function WritePrivateProfileString Lib KERNEL32 alias "WritePrivateProfileStringA"
     (ByVal lpApplicationName As String,
     ByVal lpKeyName As String,
     ByVal lpString As String,
     ByVal lplFileName As String) As Smallint
Declare Function PrivateProfileDeleteSection Lib KERNEL32 alias "WritePrivateProfileStringA"
     (ByVal lpApplicationName As String,
     ByVal lpKeyName As integer,
```

```
    ByVal lpString As string,
    ByVal lplFileName As String) As Smallint
Declare Function PrivateProfileDeleteKey Lib KERNEL32 alias "WritePrivateProfileStringA"
    (ByVal lpApplicationName As String,
    ByVal lpKeyName As string,
    ByVal lpString As integer,
    ByVal lplFileName As String) As Smallint
```

For deleting a **section** the argument list would look like

```
nStatus = PrivateProfileDeleteSection (sSection , 0 , "" , sFile)
```

and for deleting a **key**

```
nStatus = PrivateProfileDeleteKey (sSection , sKey , 0 , sFile)
```

An actual use of it is made in Application 3 : Deleting a section from an INI file. The INI_DELETE.MBX uses both kinds of deletes to clean up INI files.

Application 1 : Building an INI file from some text files.

Requirements:

Use (TXT) text files as input
Lines are ending by a return/line feed. A line should not exceed 255 characters.

Register the first 10 lines of any file, or up to the end of file if less.
A simple consideration for speed of demo and size of ini file.

Register as many files are needed.

Structure of the ini file

The structure will contain a [GENERAL] section with only one key: the number of different texts stored in the file (NTexts), plus as many sections [TEXT_] (_ = from 1 to ntxt) containing each up to 10 line_ keys (_ = from 1 to Nlines) and that Nlines key holding the number of lines in that section,

Comments *in italics* by block of code lines

Block 1 *Required "Declare" and dim of variables*
Val_Ret must be defined as a string even because it is not passed ByVal to Read…
We have chosen the solution "variable" for the default returned value

```
Declare Function WritePrivateProfileString32 Lib KERNEL32 alias
                                                    "WritePrivateProfileStringA"
  (ByVal lpApplicationName As String,
    ByVal lpKeyName As String,
    ByVal lpString As String,
    ByVal lplFileName As String) As Smallint

Declare Function GetPrivateProfileString32 Lib KERNEL32 alias "GetPrivateProfileStringA"
    (ByVal lpApplicationName As String,
    ByVal lpKeyName As String,
    ByVal lpDefault As String,
    lpReturnedString As String,
    ByVal nSize As Smallint,
    ByVal lpFileName As String) As Smallint

dim inifile, filein, ain, Val_Ret, Ret_Def, Get_Val, a, b as string
dim rstat, nlines, ntxt as smallint

Ret_Def="?"
```

Block 2 *Identifying / initializing the ini file*
The INI file will be situated in the same directory as the application.

If the file exists, the number of existing "texts" is read.
Val_Ret is initialized
the read function is called
the function value is checked as well as the Ret_Val.

If a Ref_Def is found (here a ?) , a note with an error message with a clear note on where it occurred is open and the program stops.
the value is transformed in a numeric variable

If not, the first section (GENERAL) and its only key (Ntexts) set to zero is created. We prefer to create it right away even if the key will have to be updated to make sure that this section will be at the top of the file.
Note I the call to the write function that we used 4 " on each side of the variable ntxt to make sure that the value of that variable will be properly stored between two ".
The function value is checked for possible failure of the write.

```
inifile=ApplicationDirectory$()+"text_app.ini"
ntxt=0
if FileExists (inifile) then
     Val_Ret=String$(255,"X")
     rstat=GetPrivateProfileString32 ("GENERAL", "NTexts", Ret_Def, Val_Ret,
                                                                  Len(Val_Ret), iniFile)
     if rstat=0 then Get_Val=Ret_Def else Get_Val=Val_Ret end if
     if Get_Val=Ret_Def then
          note "Reading GENERAL - Ntexts from "+inifile+"
                                          failed"+chr$(13)+chr$(13)+"Program stops"
     exit sub end if
     ntxt=val(Get_Val)
  else
     rstat=WritePrivateProfileString32 ("GENERAL", "NTexts","""" + ntxt + """", inifile)
     if rstat=0 then
          note "Writing GENERAL - Ntexts  to "+inifile+"
                                  failed"+chr$(13)+chr$(13)+"Program stops. INI file incomplete"
     exit sub end if
end if
```

Block 3 *Identifying and opening the text file*

```
filein=fileopendlg("","","TXT","Select text file")
if filein="" then exit sub end if
open file filein for input as #1
```

Block 4 *Reading the text file, writing the inii file*
The section name is composed by concatenation (variable "a")
The number of lines processed is counted
In the loop, one line of the text file is read in,
the name of the key is composed by concatenation (variable "b"),
the line (variable "ain") is written to the ini file using the variables a and b for section and key names.
the function value is always checked for possible failure.

```
a="TEXT"+str$(ntxt+1)
nlines=0
     line input #1,ain
loop1:
     b="line"+str$(nlines+1)
     rstat=WritePrivateProfileString32 (a, b, """" + ain + """", inifile)
```

```
        if rstat=0 then Get_Val=Ret_Def else Get_Val=Val_Ret end if
        if Val_Ret=Ret_Def then
              note "Writing "+a+" - "+b+" to "+inifile+" failed"+chr$(13)+chr$(13)+"Program
                                                                    stops. INI file incomplete"
        exit sub end if
        nlines=nlines+1
        line input #1,ain
    if not eof(1) and nlines<10 then goto loop1 end if
```

Block 5 *Recording the number of lines in the new section and updating the number of texts in the GENERAL section*

```
    ntxt=ntxt+1
    rstat=WritePrivateProfileString32 (a,"Nlines", """" + nlines + """", inifile)
    if rstat=0 then
          note "Writing "+a+" - Nlines to "+inifile+" failed"+chr$(13)+chr$(13)+"Program stops.
                                                                    INI file incomplete"
    exit sub end if

    rstat=WritePrivateProfileString32 ("GENERAL", "NTexts", """" + ntxt + """", inifile)
    if rstat=0 then
          note "Writing GENERAL - Ntexts  to "+inifile+" failed"+chr$(13)+chr$(13)+"Program
                                                                    stops. INI file incomplete"
    exit sub end if
```

Application 2 : Thematic style changes based on a INI file.

Requirements:

A MI table made of regions only contains a column “class” that holds category labels.

An INI file holds one or more color scheme (pairs of category label - corresponding brush definition). Its structure is best described by the example.

The purpose is to “color the regions” using any specified scheme in the INI file according the contents of the “class” column of the region table.

Structure of the ini file (example)

```
[GENERAL]                 section
Numschem=2                number of schemes

[SCHEME1]                 section
nclasses=4                number of categories
value1=a                  label for category 1
brush1=(19,0,-1)          brush for same
value2=b                        …
brush2=(20,0,-1)
value3=c
brush3=(19,16711680,-1)
```

```
value4=d
brush4=(20,16711680,-1)

[SCHEME2]
nclasses=4
value1=d
brush1=(2,16776960,0)
value2=c
brush2=(2,16711935,0)
value3=b
brush3=(2,65280,0)
value4=a
brush4=(44,0,-1)
```

Comments *in italics* by block of code lines

Block 1 *Required "Declare" and dim of variables*

Val_Ret must be defined as a string even because it is not passed ByVal to Read…
We have chosen the solution "variable" for the default returned value

```
Declare Function GetPrivateProfileString32 Lib KERNEL32 alias "GetPrivateProfileStringA"
    (ByVal lpApplicationName As String,
    ByVal lpKeyName As String,
    ByVal lpDefault As String,
    lpReturnedString As String,
    ByVal nSize As Smallint,
    ByVal lpFileName As String) As Smallint

dim inifile,filein, Val_Ret, Ret_Def, Get_Val, a, b, klas(), brus(), k as string
dim i, j, rstat, nschem, selschem, ncla, nobj as smallint
dim brossori, bross(), nubro as brush
dim o as object

Ret_Def="?"
close all
```

Block 2 *Identifying the ini file and finding how many schemes it contains*

```
inifile=fileopendlg("","","INI","Select INI file")
if inifile="" then exit sub end if

Val_Ret=String$(255,"X")
rstat=GetPrivateProfileString32 ("GENERAL", "NumSchem", Ret_Def, Val_Ret,
Len(Val_Ret), iniFile)
if rstat=0 then Get_Val=Ret_Def else Get_Val=Val_Ret end if
if Get_Val=Ret_Def then
    note "Reading GENERAL - NumSchem from "+inifile+"
failed"+chr$(13)+chr$(13)+"Program stops"
exit sub end if
nschem=val(Get_Val)
```

Block 3 *If more than 1 scheme in INI, ask which one to use*

```
selschem=1
```

```
if nschem>1 then
    dialog title "Selecting scheme"
        control statictext title "You must select one of the defined schemes" position 5,5
        control statictext title "Enter a number from 1 to "+nschem position 5,15
        control edittext width 15 value 1 into selschem position 120,25
        control okbutton position 5,40
        control cancelbutton position 120,40
if not commandinfo(1) then exit sub end if
end if

filein=fileopendlg("","","TAB","Select region file")
if filein="" then exit sub end if
```

Block 4 *Extracting from the chosen scheme category labels and brush definition*

```
a="SCHEME"+str$(selschem)

Val_Ret=String$(255,"X")
rstat=GetPrivateProfileString32 (a, "Nclasses", Ret_Def, Val_Ret, Len(Val_Ret), iniFile)
if rstat=0 then Get_Val=Ret_Def else Get_Val=Val_Ret end if
if Get_Val=Ret_Def then
    note "Reading "+a+" - Nclasses from "+inifile+" failed"+chr$(13)+chr$(13)+
                                                                    "Program stops"
exit sub end if
ncla=val(Get_Val)

redim klas(ncla)
redim brus(ncla)
redim bross(ncla)

for i= 1 to ncla
b="value"+str$(i)
Val_Ret=String$(255,"X")
rstat=GetPrivateProfileString32 (a, b, Ret_Def, Val_Ret, Len(Val_Ret), iniFile)
if rstat=0 then Get_Val=Ret_Def else Get_Val=Val_Ret end if
if Get_Val=Ret_Def then
    note "Reading "+a+" - "+ b +" from "+inifile+" failed"+chr$(13)+chr$(13)+
                                                                    "Program stops"
exit sub end if
klas(i)=Get_Val

b="brush"+str$(i)
Val_Ret=String$(255,"X")
rstat=GetPrivateProfileString32 (a, b, Ret_Def, Val_Ret, Len(Val_Ret), iniFile)
if rstat=0 then Get_Val=Ret_Def else Get_Val=Val_Ret end if
if Get_Val=Ret_Def then
    note "Reading "+a+" - "+ b +" from "+inifile+" failed"+chr$(13)+chr$(13)+
                                                                    "Program stops"
exit sub end if
brus(i)=Get_Val
next
```

Block 5 *Recording present brush*
Preparing all necessary brushes

```
brossori=currentbrush()
for i=1 to ncla
    a="set style brush makebrush"+brus(i)
    run command a
    bross(i)=currentbrush()
next
```

Block 6 *Identifying and setting up the region table*

```
open table filein as filin
map from filin
set map window frontwindow() layer 1 editable on
```

Block 7 *Applying the brush corresponding to its category to each object in the table*
A default brush (1,0,-1) is assigned first to each object.
The original brsuh is restored

```
nobj=tableinfo(filin,8)
for i=1 to nobj
    fetch rec i from filin
    o=filin.obj
    k=filin.class
    nubro=makebrush(1,0,-1)
    for j=1 to ncla
        if k=klas(j) then nubro=bross(j) exit for end if
    next
    alter object o info 3,nubro
    update filin set obj=o where rowid=i
next
set style brush brossori
```

Application 3 : Deleting a section from an INI file.

Requirements:

The code should be used to delete an entire section from an INI file.

Comments *in italics* by block of code lines

Block 1 *Required "Declare" and dim of variables*
We are going to create 2 functions, GET_SECT for building a string of semicolon separated words to be used in a listbox and Nth_NAME for retrieving a name from such a list given its position in the list. These functions are pretty much standard and could be used as an alternative to those described already in the MB Resource center "Functions and Subs" section.

```
Declare Function PrivateProfileDeleteSection Lib KERNEL32 alias
                                                    "WritePrivateProfileStringA"
    (ByVal lpApplicationName As String,
    ByVal lpKeyName As integer,
    ByVal lpString As string,
    ByVal lplFileName As String) As Smallint
```

```
declare sub main
declare function get_sect(filin as string) as string
declare function nth_name(list as string, position as smallint) as string

'================================
sub main
dim fileini, a_sect, sect_name as string
dim isect, r_status as smallint
dim do_it as logical
```

Block 2 *Selecting INI file and choosing the section to delete*

```
fileini=fileopendlg("","","INI","Open ini file")
a_sect=get_sect(fileini)
dialog title "Deleting a section in an INI file"
control statictext title "Select the section you want to delete" position 5,5
control listbox title a_sect into isect position 20,20 width 95 height 50
control okbutton
control cancelbutton
if not commandinfo(1) then exit sub end if
```

Block 3 *Retrieving the section name and deleting it after confirmation*

```
sect_name=nth_name(a_sect,isect)
do_it=ask("Do you really want to delete the section "+sect_name+" ?","Yes","No")
if not do_it then exit sub end if
r_status=privateprofiledeletesection(sect_name, 0, "", fileini)
if r_status=0 then
      note "Deleting section "+sect_name+"  from "+fileini+
                  " failed"+chr$(13)+chr$(13)+"Program stops. INI file unchanged"
      exit sub
end if
note "Section "+sect_name+" succesfully deleted from "+fileini

end sub
```

Block 4 *Function that returns a semicolon delimited string of all the section names contained in an INI file.*

```
function get_sect(filin as string) as string
dim a, ain as string
a=""
open file filin for input as #1
line input #1, ain
boucle:
if left$(ain,1)="[" then
      a=a+mid$(ain,2,len(ain)-2)+";"
end if
line input #1, ain
if not eof(1) then goto boucle end if
get_sect=left$(a,len(a)-1)
close file #1
end function
```

Block 4 *Function that returns the name in a given position of a semicolon delimited string*

```
function nth_name(list as string, position as smallint) as string
dim ideb, iend, i as smallint
i=0
ideb=1
iend=1
boucle:
iend=instr(ideb,list,";")
if iend = 0 then
      nth_name=right$(list,len(list)-ideb+1)
      exit function
end if
i=i+1
if i=position then goto laststep end if
ideb=iend+1
goto boucle
laststep:
if ideb=1 then
      nth_name=left$(list,iend-1)
  else
      nth_name=mid$(list,ideb,iend-ideb)
end if
end function
```