Initialization Files
an alternative to using the registry
originally published in Liberty BASIC Newsletter #102
-
Alyce

STORING PROGRAM INITIALIZATION DATA

Did you ever wonder how a program can remember the list of your most recently opened files, or your font preference, or that you are a registered user? There are several ways this can be done. One way is to write a simple text file, called an "ini" file.

INI FILES

"Ini" is short for "initialization". Liberty BASIC has an ini file that contains your font preference, your list of recent files, your list of externals, and other preferences and records. It is called lbasicxxx.ini and you will find it in the same folder that contains your copy of Liberty BASIC. When Liberty BASIC starts up it reads the information in this file and uses it to set your prefered font, list your recent files in the File menu, and so on. You can open this file in a texteditor and read it. The gui designer that comes with Liberty BASIC, Freeform, also has an ini file.

WRITING TO THE REGISTRY

Windows has a repository of information about the way it works. This is called The Registry. Windows looks in the registry to see which shortcuts should be displayed on your desktop, to record your list of recently opened files, to record your choice of desktop color scheme, display resolution, and all kinds of other information. Applications can write to the registry. This is a very bad idea for novice programmers! If the registry is corrupted Windows may not function properly, or some applications may not function properly. In the worst case the computer may not be usable at all! Considering the risks involved, it makes little sense to store information in the registry when something like an ini file works just fine.

USING THE API TO CREATE AN INI FILE

Earlier versions of Windows did not have a registry. All initialization data was stored in "Ini" files.

The two kernel32 functions, **WritePrivateProfileStringA** and **GetPrivateProfileStringA** provide an easy and precise way to write to and read from initialization files. These "ini" files are simple text files. Use this method as an alternative to writing information to the Windows registry. Tampering with the registry can have dire results, even causing Windows to become unusable. Writing to a private ini file can only have an impact on the single program referenced.

You might want to use ini files to record a user's list of recently opened files, or his preferences for use of the program, like syntax color "on" or "off", or even whether he is a registered user of your program.

IMPORTANT NOTE ABOUT NULL VALUES

Some parameters can be passed as strings or as null pointers in the functions explained below. Choose the correct type as needed in your program.

```
string$ as ptr, _  'passed as string

_NULL as ulong, _  'passed as null pointer
```

WRITING AN INI FILE BY API

WritePrivateProfileStringA

For information regarding this function on MSDN, see: WritePrivateProfileString

```
C++

BOOL WINAPI WritePrivateProfileString(
  _In_ LPCTSTR lpAppName,
  _In_ LPCTSTR lpKeyName,
  _In_ LPCTSTR lpString,
  _In_ LPCTSTR lpFileName
);
```

First, we'll examine writing to a private ini file. Here is the syntax of the function, which is a part of kernel32.dll:

```
calldll #kernel32, "WritePrivateProfileStringA", _
```

```
Section$ as ptr, _    'section name
Entry$ as ptr, _      'entry name
String$ as ptr, _     'actual entry
FileName$ as ptr, _   'name of ini file
result as long
```

Section$
Points to a null-terminated string that specifies the section name to which a string will be copied. Liberty BASIC automatically adds the necessary single null terminator, which is chr$(0), so we don't need to add it. If the section does not exist, it will be created. The name of the section is case-independent; the string can be any combination of uppercase and lowercase letters. If you were to read the file in a texteditor, this entry would look like this -- the section name, contained in square brackets:

[section]

If the section were called "user", it would look like this:
[user]

Entry$
Points to a null-terminated string containing the name of the key to be associated with the string. If the entry does not exist it will be created.

If you were to read the file in a texteditor this entry would look like this -- the Entry$ followed by an = sign, followed by the String$

Entry=String

If the Entry was called "name" and string$ was "Carl Gundel", it would look like this:
name=Carl Gundel

If this is _NULL (=0) and passed as type ulong instead of ptr, the entire section is deleted.

```
_NULL as ulong, _  'section is null, delete section$
```

String$
Points to the null-terminated string value to be written to the file.
Entry=String

If the Entry was called "name", and the String$ was "Carl Gundel",it would look like this:
name=Carl Gundel

If this is _NULL (=0) and passed as type ulong instead of ptr, the entry is deleted.

```
_NULL as ulong, _  'entry is null, delete entry$
```

FileName$
Points to a null-terminated string that names the INI file. If this name is a fully-qualified path and file name, it will be used. If there is no path, Windows searches for the file in the Windows directory. If it doesn't exist, it will be created.

result
Returns zero if it fails, or nonzero if it is successful.

DEMO

Here is a demo. After running this small routine, look in the Windows folder for the file "testme.ini". Open it in Notepad to see the result. Notice that your program does not need to "open" or "close" the file as you would normally do when writing to a file in Liberty BASIC.

```
Section$="User"
Entry$="Name"
String$="Carl Gundel"
FileName$="testme.ini"

CallDLL #kernel32, "WritePrivateProfileStringA", _
Section$ As ptr, _
Entry$ As ptr, _
String$ As ptr, _
FileName$ As ptr, _
result As long
```

I ran the program then looked in my Windows folder for the file, "testme.ini" and this is what it contained:

[User]
Name=Carl Gundel

We can use this method to record many kinds of information.

READING AN INI FILE BY API

GetPrivateProfileStringA
This function reads a specific ini file.

For all information regarding this function, see the MSDN:GetPrivateProfileString

```
C++
DWORD WINAPI GetPrivateProfileString(
  _In_  LPCTSTR lpAppName,
  _In_  LPCTSTR lpKeyName,
  _In_  LPCTSTR lpDefault,
  _Out_ LPTSTR  lpReturnedString,
  _In_  DWORD   nSize,
  _In_  LPCTSTR lpFileName
);
```

Section$
The name of the section containing the key name. If this parameter is _NULL, the GetPrivateProfileString function copies all section names in the file to the supplied buffer.

Entry$
The name of the key whose associated string is to be retrieved. If this parameter is _NULL, all key names in the section specified by the lpAppName(Section$) parameter are copied to the buffer specified by the lpReturnedString (ReturnString$) parameter.

Default$
A default string. If the lpKeyName key cannot be found in the initialization file, GetPrivateProfileString copies the default string to the lpReturnedString (ReturnString$) buffer. If this parameter is _NULL, the default is an empty string, "".
Avoid specifying a default string with trailing blank characters. The function inserts a null character in the lpReturnedString (ReturnString$) buffer to strip any trailing blanks.

```
Default$ = "no name given"
```

FileName$
The name of the initialization file. If this parameter does not contain a full path to the file, the system searches for the file in the Windows directory.

ReturnString$
Points to a null-terminated string that will contain the key value stored in the ini file. If the key value cannot be found, this buffer will contain the contents of the Default$ value. There is an important difference in the use of this string argument in GetPrivateProfileStringA. We will be reading data that is placed into this variables by the API function. This means that we must pass this argument "By Reference". Passing it this way means that we are not just passing the value of the string, we are passing its memory address so that the function can alter the data contained at that address. To let Liberty BASIC know that we want to pass the argument "By Reference", we include a null-termination on the string, like this:

ReturnString$ = space$(100) + chr$(0)

The addition of a null terminator to signal "by reference" no longer appears to be needed by Liberty BASIC. Strings appear to be passed "by reference" automatically.

Above, we are creating a buffer, or location in memory that is large enough to contain the data that will be placed there by the function.

SizeString
This parameter specifies the length in characters of the buffer, ReturnString$

result
The return value specifies the number of bytes copied to the specified buffer, not including the terminating null character. This means that it specifies the length of the text that the function placed into the buffer. We can use this information to read the value in the ini file without extraneous characters that might be tacked onto the end.

DEMO TWO

Here is a working example, that reads the values written to "testme.ini" that were produced in the routine above:

```
Section$="User"
Entry$="Name"
FileName$="testme.ini"
Default$ = "no name" + Chr$(0)
SizeString=100
ReturnString$=Space$(SizeString)+Chr$(0)

CallDLL #kernel32, "GetPrivateProfileStringA", _
Section$ As ptr, _
Entry$ As ptr, _
Default$ As ptr, _
ReturnString$ As ptr, _
SizeString As long, _
FileName$ As ptr, _
result As long

Print "The key is "
Print Left$(ReturnString$,result)
```

```
End
```

If you were now to use Notepad to open "testme.ini" found in the Windows directory, it would look like this:
[User]
Name=Carl Gundel

DEMO THREE

Here is a small demo that checks the ini file to see if the user has registered. You would, of course, put your own code to handle the situation. The demo simply gives a notice telling if the user is registered when the program starts. There is a button that the user can click to enter his password. Run the program for the first time, and you will get a "Not registered." notice. Click the button and type the proper password. Close the program. Run the program again, and you should receive notice that you are a "Registered user."

This demo is similar to the earlier demos, but it uses a different section and entry name:

```
Section$="Register"
Entry$="Password"

nomainwin
button #1, "Register",[reg],UL,10,10,120,26
statictext #1, "Password = 'Official'",10,50,200,30
open "Register My App" for window_nf as #1
print #1, "trapclose [quit]"

gosub [readReg] 'see if user has registered

if key$<>"Official" then
notice "Not registered."
else
notice "Registered user."
end if

wait

[quit]
close #1:end

[reg]
```

```
prompt "Enter password";pw$
if pw$="" then
notice "Not a valid password."
end if

Section$="Register"
Entry$="Password"
String$=pw$
FileName$="testme.ini"

CallDLL #kernel32, "WritePrivateProfileStringA", _
Section$ As ptr, _
Entry$ As ptr, _
String$ As ptr, _
FileName$ As ptr, _
result As long
wait

[readReg]
Section$="Register"
Entry$="Password"
FileName$="testme.ini"
Default$ = "no password" + Chr$(0)
SizeString=100
ReturnString$=Space$(SizeString)+Chr$(0)

CallDLL #kernel32, "GetPrivateProfileStringA", _
Section$ As ptr, _
Entry$ As ptr, _
Default$ As ptr, _
ReturnString$ As ptr, _
SizeString As long, _
FileName$ As ptr, _
result As long

key$=Left$(ReturnString$,result)
return
```

USING STRUCTS

Analogous functions allow you to write and read the information in structs rather than strings. The following snippets of code are not full programs and cannot be run as is.

WritePrivateProfileStructA looks like this:

```
'code by Richard Russell
struct config, soundon as long, fullscreen as long
 config.soundon.struct = 1
 config.fullscreen.struct = 0

 size = len(config.struct)
 calldll #kernel32, "WritePrivateProfileStructA", _
  "settings" as ptr, _  ' Section name
  "config" as ptr, _    ' Key name
  config as struct, _   ' Structure
  size as ulong, _      ' Size of structure
  inifile$ as ptr, ret as long
```

The data is then read with **GetPrivateProfileStructA**

```
'code by Richard Russell
 size = len(config.struct)
 calldll #kernel32, "GetPrivateProfileStructA", _
  "settings" as ptr, _  ' Section name
  "config" as ptr, _    ' Key name
  config as struct, _   ' Structure
  size as ulong, _      ' Size of structure
  inifile$ as ptr, ret as long
```

To retrieve the information from the struct:

```
   SoundOn = config.soundon.struct
   FullScreen = config.fullscreen.struct
```

For a working demonstration of WritePrivateProfileStructA by - tsh73

WritePrivateProfileStructA Full Demo

STORING PROGRAM INITIALIZATION DATA | INI FILES | WRITING TO THE REGISTRY | USING THE API TO CREATE AN INI FILE | IMPORTANT NOTE ABOUT NULL VALUES | WRITING AN INI FILE BY API | DEMO | READING AN INI FILE BY API | DEMO TWO | DEMO THREE | USING STRUCTS