*## Version icon: 2.x*
*## Platform icon: DOS, Windows*
*## Companion Disk icon: 03_HENTZ.ZIP*

# Read and Write Your Own .INI files Inside FoxPro with SetIni() and GetIni()

*Whil Hentzen*

**With the release of Visual FoxPro and Windows 95, the preferred configuration file methodology is to use the Windows registry. However, this is not always an option, and you may wish to fall back on the "INI" file mechanism. This pair of FoxPro 2.x functions will painlessly read and write settings to standard .INI text files.**

With the advent of Windows 3.1 at the beginning of this decade (yes, it's been *that* long!), developers started mimicking the use of "INI" text files for the use of configuring an application. This technique had several advantages. First, it was easy to make changes to a text file at the site of the application's installation; a task not quite as easy if you kept the configuration information in a .DBF and the application was running off of a run-time version of FoxPro. Second, to an experienced user or system administrator, the "INI" file concept was familiar, so you could direct them to make changes remotely. And third, INI files were platform independent - they worked equally well with DOS, Windows, Mac and Unix.

As Microsoft shifts its weight behind Windows and continues to encourage the use of OLE, the preferred method of storing configuration data is through the Windows registry file. However, as we've seen above, this isn't always going to work. If you're still working on non-Windows versions of applications, the registry is not accessible. And you may find that you're uncomfortable with the use of this new beast, and would prefer to stick with a mechanism that's more familiar.

Still, while the concept of reading and writing to a text file to store the application's settings is simple, the work of writing the code that will manage the creation of the file, the creation and updating of sections inside the INI file, and the handling of the various values under each section - well, it's really pretty close to busy work. There are more interesting things you could do with your time.

Fortunately, Felix H. Gonzalez has written a pair of functions that you can just plug into your applications that take care of this for you. Spend 30 seconds with SetIni() and GetIni() and you've just saved yourself several hours of drudgery. These functions aren't rocket science, but saving the better part of a day is pretty cool too, wouldn't you say?

## How to Use SetIni() and GetIni()

An INI file has the format

```
;MUPPET.INI

[Miss Piggy's Parameters]
Kermit = Boyfriend
Ralph The Dog = My Best Pal
```

Any line beginning with a semi-colon is ignored, the section heading is delimited with brackets, and the variables for that section are separated from their values with an equals sign. SetIni() and GetIni() allow you to maintain both the section heading and the variables and their values.

As you've noticed, I particularly like Cool Tools that can be used without reading the documentation, or, if you have to read the instructions, it only takes a minute to do so. The SetIni() and GetIni() functions meet this criteria. They use the same syntax:

```
m.lWasSuccessful = SetIni( "INI file name", ;
   "Section Heading", "Variable Name", "Value")

m.cReturnValue = GetIni( "INI file name", ;
   "Section Heading", "Variable Name")
```

SetIni() returns a logical true or false, depending on whether the value was able to be written. I've only found one case where the function fails - when the INI to be written is marked as read-only. Note that GetIni() doesn't require a fourth parameter since it's purpose is to return the value of a variable in the INI file. The "fourth parameter" is actually the return value.

```
m.lWorked = SetIni("muppet", "Miss Piggy's Parameters",;
    "Ralph The Dog", "My Best Pal")
? m.lWorked
.T.

m.cRetVal = GetIni("muppet", "Miss Piggy's Parameters", "Kermit")
? m.cRetVal
Boyfriend
```

As you can tell from these two function calls, using them is completely painless. If the INI file doesn't exist, it's created; if it does exist, the current one is saved as a BAK file and a new version is created. If you don't include a ".INI" extension, it will be created for you. If you don't include a path, the INI file is assumed to be (or is created) in the current directory.

SetIni() will create section and variable names if they don't exist. GetIni() will return an empty string if the section or variable doesn't exist. Note that the section and variable names are case sensitive, but you can include spaces and other delimiters in both.

I love stuff like this - saves me a bunch of time and is easy to use. How could it get any better?

### Where to Find SetIni() and GetIni()

Both functions are included on this month's Companion Disk. You can also find them in library 13 of CompuServe's FoxForum. They're freeware - use them as you wish. Just the .FXPs are provided; the source code is $12 and you can contact the author, Felix H Gonzalez at 75404,1716 for details. However, in my experience, even if you don't want the source code, be sure to drop him a note telling him that you like his work.

# Book of the Month

The last few months, I covered a series of books that can help you become a better programmer. This month, the book will help you become a better manager of programmers - and this advice is indispensable even if the only programmer you manage is yourself.

"Peopleware" by Tom DeMarco and Timothy Lister (Dorset House Publishing, ISBN 0-932633-05-6) is one of the industry's classic tomes. The entire book can be summed by the closing sentences of Chapter 6 (intriguingly titled "Laetrile). The chapter discussed the futile search for the "Silver Bullet" that all software managers hope for - the magic tool that will double their department's productivity overnight. But the answer isn't in technology - it's in the application of more effective ways of handling people, modifying the workplace, and changing the corporate culture. "Sharon knew what all good instinctive managers know: The manager's function is not to make people work, but to make it possible for people to work."

Peopleware is a quick read - 190 pages of large type in a oversized paperback, but the hypothesis and the answers ring true in a way that 10,000 pages of research printed in 6 point type of dense type couldn't provide. The length and writing style just make the story that DeMarco and Lister tell immediately accessible.

Their fundamental theory is that people want to do a better job, if only the company, rules and environment would let them. Supposing this, they then set out to design the office environment, selecting the right people, and creating productive teams so that software developers can do what they want to do - design, code and implement high quality software.

One of the first things that strikes you when you walk through the halls of RDI Software Technologies in Chicago is that virtually everyone in the company has a private office with a door that closes. Study after study has shown that the single most important factor in programmer productivity is long periods of uninterrupted time. How can that factor evaluate to anything but a negative number when the workplace

consists of a bullpen with five foot cubicles and a company paging system that blares out unintelligible nonsense every 47 seconds? But your company wouldn't allow it? DeMarco and Lister detail example after example of how to 'skirt the system' so that, at the very least, your most important project teams get into an environment that works for, not against, their goals.

What's another issue in creating a high value of what the authors call the Environmental (or "E") Factor: Uninterrupted hours/Body-Present Hours? The ability to silence that miracle of modern engineering - the telephone. They relate the tale of the manager that rebuked an entire department of highly paid software engineers: "You guys have to answer your own phones. If you all forward your phones to the secretaries, they'll never get any work done!" At my company, we've taken this idea to heart and now have "telephone free zones" each morning and each afternoon. The longest a customer will wait for a call to be returned is two hours - should they call at the very beginning of one of the zones - but they have four hours in the day where they can get through directly, and the resulting effect of having four hours of uninterrupted programming is a significant increase in our productivity - and morale!

If these ideas are of interest to you, then you'll want to pick up Peopleware - because I don't have space to go through the other 98 excellent suggestions.