Les Chroniques du Campagnol
Season 2 Episode 9
The return of the INI file

May 14, 2022

I like ini files. INI files are easy to use and understand, have limitations that force you not to overcomplicate things. But above all else, they are easily rewritable. An application can atomically do a change in an INI file without regard for any entry they do not know about. Also, INI files are remarkably idiot proof. It really takes a lot of creativity to make an INI file totally unrecoverable, while it takes a single missing character in Json or XML to completely confuse a parser.

Unfortunately, for mostly biased reasons, INI files have a bad reputation. And they are not supported on anything else than Windows.

1 User Needs

I want a cross-platform library that supports INI files mostly in the same way as the old Windows API did. Except that it does address limitations of the existing implementation, and avoids the pitfalls in which it fell.

I specifically want to address the following problems:

- Inconsistent handling of double quotes and leading spaces.
- Inconsistent handling of special characters inside values and names.

What I want to achieve is to make sure that any value written will be retrieved as is. The Microsoft implementation does not guarantee that.

2 Using a third party library

A solution is of course to use a third party library that represents the INI file internally as a dictionaries of dictionaries of strings.

A common problem with all those libraries is that they rewrite their view of the value mappings from the memory cache. By doing so, they usually lose any additional text that the user might have put in the file, especially comments.

I regularly make use of comments in INI files, especially when temporarily *commenting out* an entry:

```
[MySection]
Server=http://production
;Server=http://localhost
```

This way, it's super easy to switch from one to the other. The standard INI API from Windows does not lose those entries; most third party re-creations do.

Another issue with those libraries is that they treat the ini file as a document instead of a database: you load the ini file in memory, then make some changes, then rewrite the file from memory. If multiple applications try to update the file at the same time, race conditions will occur.

3 What I will do

At first, I'll make a very simple, maybe brutal implementation: read the file, make changes in memory, rewrite the file. And of course manage file locking so that no two applications can make concurrent changes, and no application can read the file while it is being modified. That is nice, because that is exactly what I will need to change the `hosts` file also.

4 How it will work

The system will work in a very simple and layered fashion: one layer manages the file locking, the other manages the contents.

4.1 Layer 1 - Managing the file

1. Lock the file for exclusive access.
2. Read the contents of the file line by line.
3. Pass the lines to layer 2 a filter function that will transform needed lines, remove some, etc.
4. Save the filtered lines in memory until the whole file has been processed.
5. Rewrite the file from the start.
6. Release the exclusive lock.

Nice, but that will fail if the file does not exist yet

Yes, this is another problem we need to solve, we can't lock a nonexistent file... There may be platform specific ways of creating a file and lock it atomically, but we still need to handle the case where two processes will try to create the file at the same time[1]. Therefore, instead of trying to be smart, let's try to be consistent: if the file does not exist, create an empty file. If the creation fails, then someone else has created the file. Once the file exists, we fall back to the previous case.

4.2 Layer 2 - The filter function

The second layer is the filter function. I chose to make the function operate on an enumeration of lines, and return an enumeration of lines. That way, I can write the function as a generator, and I avoid managing line endings in the filter.

What follows is an example implementation of the filter, with many details removed.

```
1   IEnumerable<string> Filter (IEnumerable<string> Input, SectionToChange,
        NameToChange, NewValue)
2   {
3       string CurrentSection = null;
4       foreach (string s in Input)
5           if (IsSectionHeader (s, out string NewSection))
6           {
7               if (CurrentSection == SectionToChange && NameToChange is not null)
8               {
9                   yield return $"{NameToChange}={NewValue}";
10                  NameToChange = null;
11              }
12              CurrentSection = NewSection;
13              yield return s;
14          }
15          else if (IsMappingEntry (s, out string Name, out string Value))
16          {
17              if (CurrentSection == SectionToChange && Name == NameToChange)
18              {
```

[1] This situation is likely to happen when an application consists of multiple cooperating processes. These are likely to start at the same time, and thus try to access their initialization parameters at the same time.

```
                    yield return $"{NameToChange}={NewValue}";
                    NameToChange = null;
                }
                else
                   yield return s;
            }
            else
                yield return s;
    }
```

One other nice thing of the layered approach is that I can make unit tests about the transformation function without having to actually write a file.

5 Future improvements

There are many things that need to be supported. The first is arbitrary strings (including binary data and/or data that looks like section headers, etc...) This can be solved quite easily by using escaped strings.

Another nice-to-have feature would be to automatically comment/uncomment lines. For example, if I have the following INI file:

```
[MySection]
Server=http://production
;Server=http://localhost
```

and the code calls:

```
WritePrivateProfileString ("MySection", "Server", "http://localhost")
```

What a ”normal” INI file handler will do is generate this:

```
[MySection]
Server=http://localhost
;Server=http://localhost
```

I would greatly prefer that the function detects that there is a commented-out line for that new value, then uncomment that line while commenting-out the current value:

```
[MySection]
;Server=http://production
Server=http://localhost
```

And finally, I would like to have an in-memory cache where so that I do not need to re-read the file each time if it has not changed.