INI file

From Wikipedia, the free encyclopedia

The **INI file** format is an informal standard for configuration files for some platforms or software. INI files are simple text files with a basic structure composed of sections, properties, and values.[1]

In MS-DOS and 16-bit Windows platforms up through Windows ME, the INI file served as the primary mechanism to configure operating system and installed applications features, such as device drivers, fonts, startup launchers, and things that needed to be initialized in booting Windows. INI files were also generally used by applications to store their individual settings.[2]

In Windows NT Microsoft introduced the registry, and began to steer developers away from using INI files for configuration. All subsequent versions of Windows have used the Windows Registry for system configuration, and applications built on the .NET Framework use special XML .config files. The APIs still exist in Windows, however, and developers may still use them.

The name "INI file" comes from the commonly used filename extension `.INI`, which stands for "initialization". Other common initialization file extensions are `.CFG`, `.conf`,[3] and `.TXT`,[4] especially 'config.txt' occurrences.

Linux and Unix systems also use a similar file format for system configuration. In addition, platform-agnostic software may use this file format for configuration. It is human-readable and simple to parse, so it is a usable format for configuration files that do not require much greater complexity.

Contents

Format

Keys (properties)

The basic element contained in an INI file is the *key* or *property*. Every key has a *name* and a *value*, delimited by an equals sign (=). The name appears to the left of the equals sign.

```
name=value
```

Sections

Keys may (but need not) be grouped into arbitrarily named *sections*. The section name appears on a line by itself, in square brackets ([and]). All keys after the section declaration are associated with that section. There is no explicit "end of section" delimiter; sections end at the next section declaration, or the end of the file. Sections may not be nested.

```
[section]
a=a
b=b
```

Case insensitivity

Section and property names are not case sensitive in the Windows implementation.[5]

Comments

Semicolons (;) at the beginning of the line indicate a comment. Comment lines are ignored.

```
; comment text
```

Varying features

The INI file format is not well defined. Many programs support features beyond the basics described above. The following is a list of some common features, which may or may not be implemented in any given program.

Blank lines

Some rudimentary programs do not allow blank lines. Every line must therefore be a section head, a property, or a comment.

Comments

Some software supports the use of the number sign (`#`) as an alternative to the semicolon for indicating comments. Practically speaking, using it to begin a line may effectively change a variable name on that line. For instance, the following line creates a variable named "#var", but not one named "var"; this is sometimes used to create a pseudo-implementation of a comment.

```
#var=a
```

More generally, use of the number sign is unpredictable, as in these following lines (note the space after the number sign in the second line). For this reason, the number sign character should not be used to begin comments.

```
#[section]
# var=a
```

In some implementations, a comment may begin anywhere on a line, including on the same line after properties or section declarations. In others, including the WinAPI function `GetPrivateProfileString`, comments must occur on lines by themselves.

Duplicate names

Most implementations only support having one property with a given name in a section. The second occurrence of a property name may cause an abort, it may be ignored (and the value discarded), or it may override the first occurrence (with the first value discarded). Some programs use duplicate property names to implement multi-valued properties.

Interpretation of multiple section declarations with the same name also varies. In some implementations, duplicate sections simply merge their properties together, as if they occurred contiguously. Others may abort, or ignore some aspect of the INI file.

Escape characters

Some implementations also offer varying support for an escape character, typically with the backslash (`\`). Some support "line continuation", where a backslash followed immediately by EOL (end-of-line) causes the line break to be ignored, and the "logical line" to be continued on the next actual line from the INI file. Implementation of various "special characters" with sequences escapes is also seen.

Common escape sequences

Sequence	Meaning
`\\`	\ (a single backslash, escaping the escape character)
`\0`	Null character
`\a`	Bell/Alert/Audible
`\b`	Backspace, Bell character for some applications
`\t`	Tab character
`\r`	Carriage return
`\n`	Line feed
`\;`	Semicolon
`\#`	Number sign
`\=`	Equals sign
`\:`	Colon
`\x????`	Unicode character with hexadecimal code point corresponding to *????*

Global properties

Optional "global" properties may also be allowed, that are declared before any section is declared.[6]

Hierarchy

Most commonly, INI files have no hierarchy of sections within sections. Some files appear to have a hierarchical naming convention, however. For section A, subsection B, sub-subsection C, property P and value V, they may accept entries

such as `[A.B.C]` and `P=V` (Windows' `xstart.ini`), `[A\B\C]` and `P=V` (the IBM Windows driver file `devlist.ini`), or `[A]` and `B,C,P = V` (Microsoft Visual Studio file `AEMANAGR.INI`).

It is unclear whether these are simply naming conventions that an application happens to use in order to give the *appearance* of a hierarchy, or whether the file is being read by a module that actually presents this hierarchy to the application programmer.

Name/value delimiter

Some implementations allow a colon (`:`) as the name/value delimiter (instead of the equals sign).

Quoted values

Some implementations allow values to be quoted, typically using double quotes and/or apostrophes. This allows for explicit declaration of whitespace, and/or for quoting of special characters (equals, semicolon, etc.). The standard Windows function GetPrivateProfileString (http://msdn2.microsoft.com/en-us/library/ms724353.aspx) supports this, and will remove quotation marks that surround the values.

Whitespace

Interpretation of whitespace varies. Most implementations ignore leading and trailing whitespace around the outside of the property name. Some even ignore whitespace within values (for example, making "host name" and "hostname" equivalent). Some implementations also ignore leading and trailing whitespace around the property value; others consider all characters following the equals sign (including whitespace) to be part of the value.

Order of sections and properties

In most cases the order of properties in a section and the order of sections in a file is irrelevant, but implementations may vary.

Example

Following is an example INI file for an imaginary program. It has two sections: one for the owner of the software, and one for a payroll database connection. Comments note who modified the file last and why an IP address is used instead of a DNS name.

```
; last modified 1 April 2001 by John Doe
```

```
[owner]
name=John Doe
organization=Acme Widgets Inc.

[database]
; use IP address in case network name resolution is not working
server=192.0.2.62
port=143
file="payroll.dat"
```

Accessing INI files

Under Windows, the *Profile API* is the programming interface used to read and write settings from classic Windows .ini files. For example, the GetPrivateProfileString (http://msdn.microsoft.com/en-us/library/ms724353(VS.85).aspx) function retrieves a string from the specified section in an initialization file.

The following sample C program demonstrates reading property values from the above sample INI file (Let the name of configuration file be `dbsettings.ini`)

```
#include <windows.h>

int main(int argc, _TCHAR *argv[])
{
  _TCHAR dbserver[1000];
  int dbport;
  GetPrivateProfileString("database", "server", "127.0.0.1", dbserver, sizeof(dbserver) / sizeof(dbserve
  dbport = GetPrivateProfileInt("database", "port", 143, ".\\dbsettings.ini");
  // N.B. WritePrivateProfileInt() does not exist
  return 0;
}
```

File mapping

Initialization File Mapping[7][8] creates a mapping between an INI file and the Registry. It was introduced with Windows NT and Windows 95 as a way to migrate from storing settings in classic .ini files to the new Windows Registry. File mapping traps the Profile API calls and, using settings from the `IniFileMapping` Registry section, directs reads and writes to appropriate places in the Registry.

Using the Example above, a string call could be made to fetch the *name* key from the *owner* section from a settings file called, say, *dbsettings.ini*. The returned value should be the string "John Doe":

```
GetPrivateProfileString("owner", "name", ... , "c:\\programs\\oldprogram\\dbsettings.ini");
```

INI mapping takes this Profile API call, ignores any path in the given filename and

checks to see if there is a Registry key matching the filename under:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\
   CurrentVersion\IniFileMapping
```

If this exists, it looks for an entry name matching the requested section. If an entry is found, INI mapping uses its value as a pointer to another part of the Registry. It then looks up the requested INI setting in that part of the Registry.

If no matching entry name is found and there is an entry under the `(Default)` entry name, INI mapping uses that instead. Thus each section name does not need its own entry.

HKEY_LOCAL_MACHINE\Software\...\IniFileMapping\dbsettings.ini	
`(Default)`	@USR:Software\oldprogs\inisettings\all
`database`	USR:Software\oldprogs\inisettings\db

So, in this case the profile call for the [owner] section is mapped through to:

HKEY_CURRENT_USER\Software\oldprogs\inisettings\all	
`name`	John Doe
`organization`	Acme Products

where the "`name`" Registry entry name is found to match the requested INI key. The value of "John Doe" is then returned to the Profile call. In this case, the @ prefix on the default prevents any reads from going to the dbsettings.ini file on disk. The result is that any settings not found in the Registry are not looked for in the INI file.

The "`database`" Registry entry does not have the @ prefix on the value; thus, for the [database] section *only*, settings in the Registry are taken first followed by settings in the dbsettings.ini file on disk.

Alternatives

Starting with Windows 95, Microsoft began strongly promoting the use of Windows registry over the INI file.[9] INI files are typically limited to two levels (sections and properties) and do not handle binary data well

Later XML-based configuration files became a popular choice for encoding configuration in text files. XML allows arbitrarily complex levels and nesting, and has standard mechanisms for encoding binary data.

More recently, data serialization formats, such as JSON and YAML can serve as

configuration formats. These three alternative formats can nest arbitrarily, but have a more heavyweight syntax than the INI file.

See also

- BOOT.INI
- MSConfig
- Sysedit
- SYSTEM.INI
- WIN.INI
- Amiga's IFF files

References

1. Microsoft TechNet: Configure an Ini File Item (http://technet.microsoft.com/en-us/library/cc731332.aspx)
2. Microsoft: Windows NT Workstation Resource Kit (https://www.microsoft.com/resources/documentation/windowsnt/4/workstation/reskit/en-us/26_ini.mspx)
3. .conf initialization files (http://www.fileinfo.com/extension/conf)
4. See Trainz which uses config.txt for virtually all data base assets. (Trainsoptions.txt is that app's version of a .ini file)
5. "GetPrivateProfileString function". *Microsoft Developer Network*. Microsoft. Retrieved 2012-06-02.
6. Apache Documentation for org.apache.commons.configuration2.INIConfiguration (http://commons.apache.org/proper/commons-configuration/apidocs/org/apache/commons/configuration2/INIConfiguration.html), The Apache Software Foundation
7. Initialization Files and the Registry (http://technet.microsoft.com/en-ie/library/cc722567(en-us).aspx), *Windows NT Workstation Resource Kit*, Microsoft TechNet
8. Administering the NT Registry (http://oreilly.com/catalog/manwinreg/chapter/ch08.html), *Managing the Windows NT Registry*, Paul Robichaux, O'Reilly Media
9. The System Registry (https://msdn.microsoft.com/en-us/library/ms970651.aspx)

External links

- Easy to use INI files (http://wiki.freepascal.org/Using_INI_Files)
- RudeConfig™ Open Source C++ Config File Library (http://rudeserver.com/config/index.html): INI parser written in C++.
- C# INI Reader/Writer (https://github.com/DeanReynolds/C--INI-Parser-Writer/blob/master/INI.cs): INI Reader/Writer written in C#.
- qLibc - General C Library (https://github.com/wolkykim/qlibc): INI parser written in C.
- Simple .INI file parser in C (https://github.com/benhoyt/inih): INI parser written in C.
- clinicap (CLiki) (http://www.cliki.net/clinicap): INI parser/generator written in Common Lisp.
- A very simple data file metaformat (http://octodecillion.com/blog/very-simple-

data-file-format): INI parser tutorial in Groovy.
- Cloanto Implementation of INI File Format (https://cloanto.com/specs/ini/): The particular syntax allowed by a parser implemented by Cloanto.
- Bohr: File format (https://github.com/chazomaticus/bohr/blob/master/doc/Ni/FileFormat.txt): The particular syntax with hierarchical extensions allowed by a parser implementation called Nickel in a project called Bohr.

Categories: Computer file formats | Configuration files
