Freelancer's Binary Initialization File Format *("BINI")* Specification

The Initialization File *("INI")*

The purpose of an initialization file is to store values used to initialize a program. These values are stored under named entries. An example entry could be the entry `color` with the value `blue`. These entries are grouped in *sections*.

The plain text initialization file

The most common initiaization file format is the plain text ini:

```
[section]
an-entry = a-text-value, 1234
a-floating-point-value = 1.2345
number = 12344
another-entry = 1.00, 123, foo, bar

[another-section]
name = another-section
just = another, entry
```

An entry is represented by its name, followed by a `=` and the comma-seperated values. Grouped under sections. These are represented by their name in between of brackets (`[`, `]`).

The plaintext ini format is easy to read, edit and maintain for a human, ideal for development. The drawback is that a plain text INI is hard to read for a computer program. Another drawback is that data isn't stored efficiently. A 9 numbered digit would fit when binary encoded in just 4 characters!

The BINI format

The BINI format is used for these advantages above the normal plain text format:

- **Binary encoding**, numbers are stored binary, which makes them a lot smaller, and faster to load.
- **String table**, strings (pieces of text) are stored in one table, which prevent duplicates: when `name` is used 5 times in a plain text ini, there will be 1 `name` in the string table which is referenced 5 times.

- **Smaller**, the BINI's are smaller for they store data more efficiently using stringtables and binary encoded primitives.
- **Faster**, the BINI's load a lot faster for they are smaller but most important can be loaded efficiently.

Structure

A bini consists of 3 parts: the *header*, the *main data* and the *strintable*.

The header

The header stores the location in the BINI where the string table starts, and filetype recognition part, so that programs are able to regognize the file type and version.

Length	Type	Purpose
4	string	File type (`BINI` default)
4	int32	Format version (`0x00000001` default)
4	int32	String table offset

The main data

The main data section relies directly after the *header*. It contains the sections, entries and values. It consists of one or more *section* structures. These *section structures* themselves contain zero or more *entry structures*, which contain zero or more *value structures*.

Section structure

The section structure (representing a section as the name suggests) contains the name and entries of the section represented:

Length	Type	Purpose
2	int16 stp*	Name of the section
2	int16	Number of entries in this section

These 4 bytes are directly followed by the entry structures of that section.

* stp = **s**tring **t**able **p**ointer. A stringtablepointer is a number which respresends a string in the stringtable by its offset in the stringtable. eg. The first string in the stringtable would be referenced by the

number 0. The second string in the stringtable can start at the 7th byte of the stringtable and therefor would be pointed to by the number 7.

Entry structure

The entry structure (representing an entry) contains the name, and values in a similar way as the section structures contain their own names and entries:

Length	Type	Purpose
2	int16 stp	Name of the entry
1	byte	Number of values in this entry

The value structures directly follow these 3 bytes.

Value structure

A value consists out of 2 parts, the type of value, and the actual data of the value:

Length	Type	Purpose
1	byte enum	Value type 0x01: Integer 0x02: Floating point 0x03: String
4	int32/ single/ int32 stp	The data (type specified in the previous byte enum)

String table

The string table is a lot of null terminated strings, which are refered to from the main data section, by their offset relative to the start of the string table. (see example above.)

Notes

There is no 'number of sections' field in a BINI. There are as many sections as fit in the space between the end of the header (0x12) and the stringtableoffset. For section structures do not have a fixed length it is impossible to random access a given section without having read all the previous sections.

Only the value stringtable pointer uses a 32bit integer, the other stringtable pointers use 16bit integers. When creating a BINI you should make sure that the section and entry name strings are put at the start of the stringtable or otherwise the size of the bini will be fairly limited. (65536 (0xFFFF) bytes of entry/section name string affected stringtable).

An example parser, and compressor

(© 2004 - Bas Westerbaan, Intrepid)
Note that a unprovided abstract class is inherited. This code is only provided as an example way to parse and compress a freelancer INI. This code may not be used without explicit permission of Intrepid Software.

```
using System;
using System.IO;
using System.Text;
using System.Collections;

namespace Intrepid.Corelib.Data.Ini
{
        /// <summary>
        /// A freelancer BINI file
        /// </summary>
        public class FlIniFile : IniFile
        {
                /// <summary>
                /// Initializes a new instance of TextIniFile
                /// </summary>
                /// <param name="iniFile"></param>
                public FlIniFile(IniFile iniFile)
                {
                        this._Sections = iniFile.Sections;
                }

                /// <summary>
                /// Class used to generate stringtable
                /// </summary>
                protected class StringTable
                {
                        /// <summary>
                        /// An stringtable item
                        /// </summary>
                        struct Item
                        {
                                /// <summary>
                                /// Initializes a new instance of item
                                /// </summary>
                                /// <param name="index">The index</param>
                                /// <param name="string">The
string</param>
```

```
                        public Item(int index, string @string)
                        {
                                this.Index = index;
                                this.String = @string;
                        }

                        /// <summary>
                        /// The index of the item
                        /// </summary>
                        public int Index;
                        /// <summary>
                        /// The string
                        /// </summary>
                        public string String;
                }

                /// <summary>
                /// The items
                /// </summary>
                ArrayList items = new ArrayList();
                /// <summary>
                /// Index counter
                /// </summary>
                int counter = 0;

                /// <summary>
                /// Adds a string
                /// </summary>
                /// <param name="String"></param>
                /// <returns></returns>
                public int Add(string String)
                {
                        foreach(Item i in items)
                        {
                                if(i.String.Equals(String))
                                        return i.Index;
                        }

                        Item ni = new Item(counter, String);
                        counter += String.Length + 1;
                        items.Add(ni);
                        return ni.Index;
                }

                /// <summary>
                /// Emits the stringtable
                /// </summary>
                /// <returns></returns>
                public byte[] Emit()
                {
                        byte[] ret = new byte[counter];
                        int i2 = 0;

                        foreach(Item i in items)
                        {
                                Encoding.ASCII.GetBytes(i.String,
0, i.String.Length, ret, i2);
```

```
                                        i2 += i.String.Length + 1;
                                }
                                return ret;
                        }
                }

                /// <summary>
                /// Initializes a new instance of FlIniFile
                /// </summary>
                public FlIniFile()
                {
                }

                /// <summary>
                /// Gets a string from a string table
                /// </summary>
                /// <param name="StringTable"></param>
                /// <param name="Offset"></param>
                /// <returns></returns>
                private string GetStringTableString(byte[] StringTable,
int Offset)
                {
                        int pos = Offset;

                        while(pos < StringTable.Length &&
StringTable[pos] != 0)
                                pos++;

                        return Encoding.ASCII.GetString(StringTable,
Offset, pos - Offset);
                }

                /// <summary>
                /// Checks the ini stream to check whether this is a
freelancer INI
                /// </summary>
                /// <param name="stream">The stream to check</param>
                /// <returns>Whether it is a freelancer ini</returns>
                public static bool IsFlIni(Stream stream)
                {
                        byte[] buffer = new byte[4];

                        if(stream.Read(buffer, 0, 4) != 4)
                        {
                                return false;
                        }
                        else
                        {
                                if(buffer[0] == 'B' && buffer[1] == 'I'
&& buffer[2] == 'N' && buffer[3] == 'I')
                                {
                                        return true;
                                }
                                else
                                {
                                        return false;
                                }
```

```
                }
        }

        /// <summary>
        /// Checks the ini file to check whether this is a
freelancer INI
        /// </summary>
        /// <param name="fileName">The file to check</param>
        /// <returns>Whether it is a freelancer ini</returns>
        public static bool IsFlIni(string fileName)
        {
                FileStream fs = new FileStream(fileName,
FileMode.Open, FileAccess.Read, FileShare.ReadWrite);
                bool b = IsFlIni(fs);
                fs.Close();
                return b;
        }

        /// <summary>
        /// Saves a freelancer binary ini to a stream
        /// </summary>
        /// <param name="stream">The stream to save to</param>
        public override void Save(Stream stream)
        {
                BinaryWriter bw = new BinaryWriter(stream,
Encoding.ASCII);
                byte[] header = new byte[8]{0x42, 0x49, 0x4e,
0x49, 0x01, 0x00, 0x00, 0x00};
                bw.Write(header, 0, 8);
                StringTable st = new StringTable();
                int offset = 12;

                foreach(IniSection s in _Sections)
                {
                        offset += 4;
                        st.Add(s.Name);

                        foreach(IniEntry e in s.Entries)
                        {
                                offset += 3 + e.Values.Count * 5;
                                st.Add(e.Name);
                        }
                }

                bw.Write(offset);

                foreach(IniSection s in _Sections)
                {
                        bw.Write((short)st.Add(s.Name));
                        bw.Write((short)s.Entries.Count);

                        foreach(IniEntry e in s.Entries)
                        {
             bw.Write((short)st.Add(e.Name));
                                bw.Write((byte)e.Values.Count);

                                foreach(IniValue v in e.Values)
```

```
                                        {
                                                if(v.ValueType ==
IniValueType.Float)
                                                {
                                                        bw.Write((byte)2);

        bw.Write((float)v.Value);
                                                }
                                                else if(v.ValueType ==
IniValueType.Integer)
                                                {
                                                        bw.Write((byte)1);

        bw.Write((int)v.Value);
                                                }
                                                else if(v.ValueType ==
IniValueType.String)
                                                {
                                                        bw.Write((byte)3);

        bw.Write((int)st.Add((string)v.Value));
                                                }
                                        }
                                }
                        }

                        byte[] stb = st.Emit();
                        bw.Write(stb, 0, stb.Length);
                        bw.Flush();
                }

                /// <summary>
                /// Loads a freelancer binary ini from a stream
                /// </summary>
                /// <param name="stream">The stream from which to
load</param>
                public override void Load(Stream stream)
                {
                        Clear();
                        BinaryReader br = new BinaryReader(stream,
Encoding.ASCII);
                        char[] header = new char[8];
                        br.Read(header, 0, 8);

                        int stringTableOffset = br.ReadInt32();
                        int stringTableLength = (int)br.BaseStream.Length
- stringTableOffset;
                        br.BaseStream.Seek(stringTableOffset,
SeekOrigin.Begin);
                        byte[] stringTable = new byte[stringTableLength];
                        br.Read(stringTable, 0, stringTableLength);
                        br.BaseStream.Seek(12, SeekOrigin.Begin);

                        while(br.BaseStream.Position + 4 <=
stringTableOffset)
                        {
                                short nameOffset = br.ReadInt16();
```

```
                                short entryCount = br.ReadInt16();

                                IniSection s = new
IniSection(GetStringTableString(stringTable, nameOffset),
                                        new ArrayList(entryCount));

                                _Sections.Add(s);

                                for(int i = 0; i < entryCount; i++)
                                {
                                        nameOffset = br.ReadInt16();
                                        byte valueCount = br.ReadByte();

                                        IniEntry e = new
IniEntry(GetStringTableString(stringTable, nameOffset),
                                                new
ArrayList(valueCount));

                                        s.Entries.Add(e);

                                        for(int j = 0; j < valueCount;
j++)
                                        {
                                                byte vt = br.ReadByte();
                                                IniValueType valueType =
0;
                                                object val = null;

                                                if(vt == 1)
                                                {
                                                        valueType =
IniValueType.Integer;
                                                        val =
br.ReadInt32();
                                                }
                                                else if(vt == 2)
                                                {
                                                        valueType =
IniValueType.Float;
                                                        val =
br.ReadSingle();
                                                }
                                                else if(vt == 3)
                                                {
                                                        valueType =
IniValueType.String;
                                                        val =
GetStringTableString(stringTable, br.ReadInt32());
                                                }

                                                e.Values.Add(new
IniValue(valueType, val));
                                        }
                                }
                        }
                }
        }
```

```
}
```

About

Version 1 of the document, released 29-10-2004.
By Bas Westerbaan, bas.westerbaan at w-nz dot com.
Released on Modsplanet