Vircon32

32-BIT VIRTUAL CONSOLE

System specification

Part 9: File formats

Document date 2022.11.13 Written by Carra

What is this?

This document is part number 9 of the Vircon32 system specification. This series of documents defines the Vircon32 system, and provides a full specification describing its features and behavior in detail.

The main goal of this specification is to define a standard for what a Vircon32 system is, and how a gaming system needs to be implemented in order to be considered compliant. Also, since Vircon32 is a virtual system, an important second goal of these documents is to provide anyone with the knowledge to create their own Vircon32 emulators.

About Vircon32

The Vircon32 project was created independently by Carra. The Vircon32 system and its associated materials (including documents, software, source code, art and any other related elements) are owned by the original author.

Vircon32 is a free, open source project in an effort to promote that anyone can play the console and create software for it. For more detailed information on this, read the license texts included in each of the available software.

About this document

Summary

Part 9 of the specification describes the file formats used for all information that a Vircon32 implementation needs to load and/or save during its operation. This includes file headers and layouts, data formats and basic validations for file correctness.

1 Introduction

For its operation, a Vircon32 system needs to be able to work with external information. Specifically it will need to read the contents of BIOS and cartridges, and to read and write the contents of memory cards. We will consider these external contents to be represented as “files”.

Every implementation may handle these files with different file systems and store them using different physical supports. For this reason we will abstract away these details: for our purposes a “file” consists merely of an ordered sequence of bytes. Optionally, a file can have a name associated to it. We also consider a file to be persistent: its contents will remain stored and accessible even if a particular Vircon32 system is turned off.

Any files that store external contents to be used by Vircon32 systems (again: BIOS, cartridges and memory cards) must be interchangeable between different Vircon32 consoles. To ensure compatibility across different implementations, this document will describe the formats that each of these file types will use to represent their information.

Note that the representation formats described in this document are only mandatory for information exchange (i.e. information that will cross the boundary of a Vircon32 system). However, implementations are still allowed to use other formats for their internal representations or processing. For instance, a hardware implementation may want to use a different internal representation for GPU textures to allow a simpler access to pixels from video hardware.

2 Types of Vircon32 files

We will first enumerate the different file types that are used by Vircon32 systems, and their basic characteristics. Each of these file types will be described in detail in the following sections.

ROM files

ROMs are the main files used by any Vircon32 system. A ROM is an executable file that packs a program together with its needed textures and sounds, all in a single file. Vircon32 uses file extension *.v32 for ROM files.

BIOS and cartridges are both executables, so both of them are stored in this same file format. The only difference is that, for a BIOS, the file signature changes and some additonal restrictions apply. This is detailed in section 7.

Asset files

As we stated, ROMs pack together 3 different types of assets for an executable: the program binary, GPU textures and SPU sounds. Each of this assets can also be represented individually as a stand-alone file, resulting in these 3 file formats:

- **Program binary files** (extension *.vbin)
  They contain a full binary (program + data) for a single program.

- **GPU texture files** (extension *.vtex)
  They contain a single texture usable by the graphics chip (GPU).

- **SPU sound files** (extension *.vsnd)
  They contain a single sound usable by the sound chip (SPU).

Note that these assets are only meant to be handled as stand-alone files by development tools or other support programs. A Vircon32 system itself will not normally use assets in this way. Still, they need to be described here because these formats will still be present as part of the ROM files themselves.

An important point to take into account about all these formats is that assets are always stored uncompressed. This results in larger files, but the trade-off is that reading/writing these files is much simpler and prevents dependencies from additional audio and video libraries to handle specific image or sound formats.

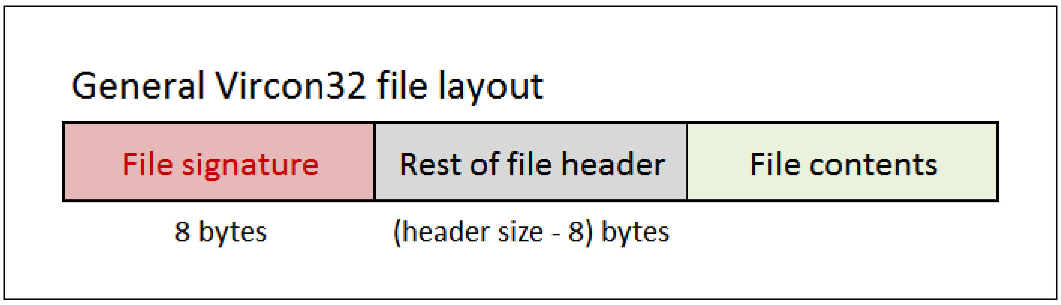
Memory card files

Memory cards are essentially memory regions that Vircon32 programs can read from and write to. After execution ends the contents of that memory are stored in a file. Vircon32 uses file extension *.memc for memory card files.

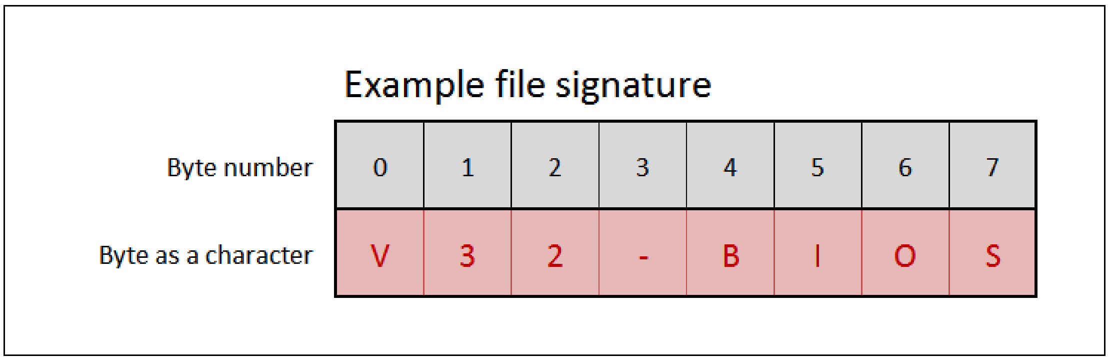
2.1 File Signatures

Using file extensions to distinguish between file types is convenient, but not reliable enough. Users could change file extensions, and some file systems might not support extensions at all. Because of this, the type of file needs to be identifiable from the file content itself.

To this end, all Vircon32 files include a known “file signature” in its first 8 bytes, to identify its content format. Next sections will show the different possible file signatures as part of their file headers. The file signature is always shown in red as the first field in all file headers.



File signatures are stored as a sequence of 8 characters encoded in standard ASCII. Still, since an exact match is required to recognise a particular signature, it is enough to perform a format-agnostic comparison of 64-bit values. For example, in a valid BIOS file we will expect the file's first 8 bytes to match this:



Example file signature

Byte number	0	1	2	3	4	5	6	7
Byte as a character	V	3	2	-	B	I	O	S

As a result of using file signatures, any file with size not greater than 8 bytes can be immediately discarded as an invalid Vircon32 file.

3 Used data formats

All these files contain information that is arranged in fields or sequences. Every field or sequence element will be encoded using one of the following data formats. Note that for all 4-byte data formats the bytes are arranged in little-endian.

3.1 Character

A character is a single byte encoded in ASCII. Whenever data fields in these file formats contain characters, they group them in fixed-capacity arrays of characters to form strings. Characters are used in 2 different types of strings:

- **File signatures**. They are not null-terminated because they are always exactly 8 characters. The characters used for valid file signatures are limited to standard ASCII: numeric values from 0 to 127.

- **ROM titles**. These have a variable length, so they are null-terminated: a byte with value zero is placed after the last used character to mark where the string ends. For ROM titles the use of extended ASCII is allowed, and it will be interpreted according to codepage 1252, also known as Latin-1.

3.2 Integer

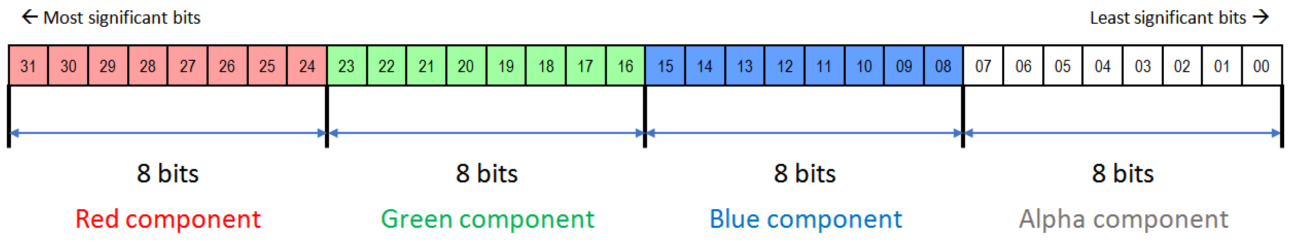
These files always represent integers using 4 bytes. Each of these integers is interpreted as a 32-bit unsigned integer. This is equivalent to the C data type "uint32_t".

3.3 CPU word

Files containing program binary data will represent each of their stored words as a single 32-bit binary value. During program execution the Vircon32 CPU may interpret each word as either an instruction or other data formats but, within the context of these files, stored words require no further interpretation than just a sequence of 32 bits.

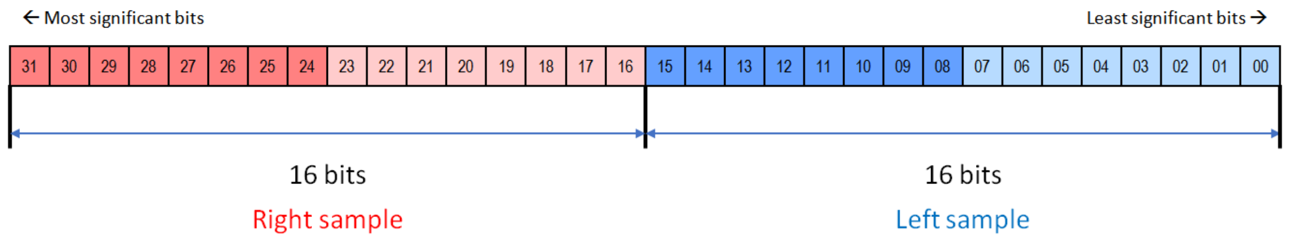
3.4 GPU pixel

Files containing GPU textures will represent each of their stored pixels as a single 32-bit RGBA color, as already documented in part 2 of the specification:



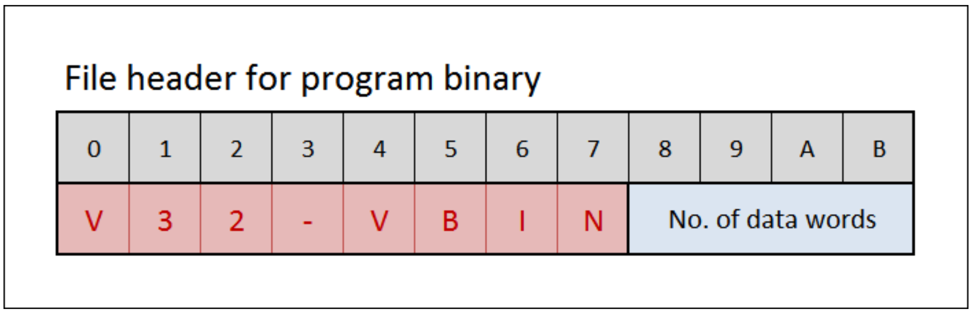
3.5 SPU sample

Files containing SPU sounds will represent each of their samples as a single 32-bit value that encodes 16-bit values for left and right channels, as already documented in part 2 of the specification:



4 Program binary files

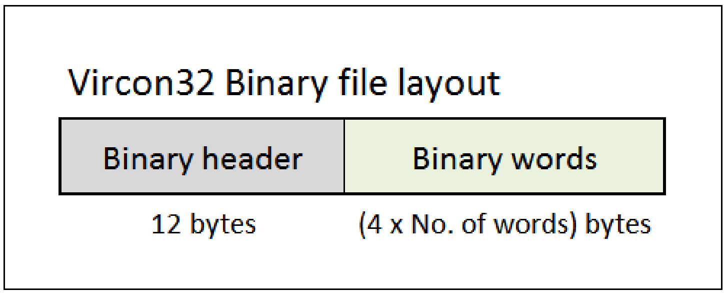
Aside from the file signature, the header of a program binary file only contains 1 integer field stating the number of 32-bit CPU words that compose this binary.



File header for program binary

0	1	2	3	4	5	6	7	8	9	A	B
V	3	2	-	V	B	I	N	No. of data words			

After the header, the file only contains all of those CPU words in sequence.



4.1 Validations for program binary files

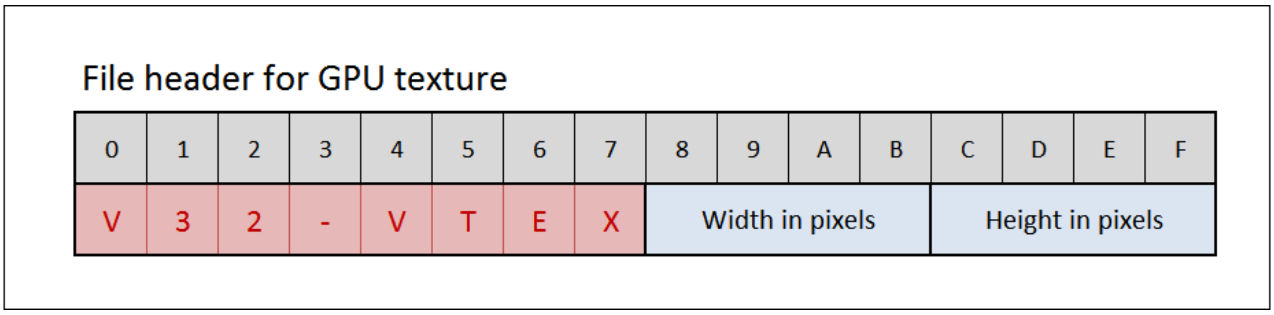
Aside from having the correct file signature, the following assertions can be tested to ensure that a program binary file is valid:

1) The declared number of words is valid. The range is from 1 up to the applicable program ROM size limit (see cartridge and BIOS limits on section 7).

2) The file size in bytes must be 12 + 4 x { No. of data words }.

5 GPU texture files

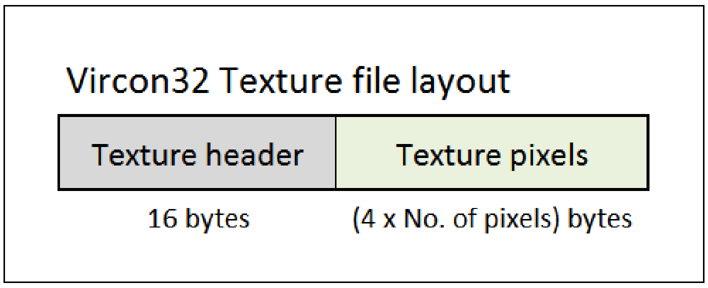
Although the internal size of a GPU texture is fixed to 1024 x 1024 pixels, most game textures will only use part of that area. For that reason texture files store only a rectangle of the specified width and height. So, aside from the file signature, the header of a GPU texture file only contains 2 integer fields stating the dimensions in pixels of the stored texture area.



File header for GPU texture

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
V	3	2	-	V	T	E	X	Width in pixels				Height in pixels			

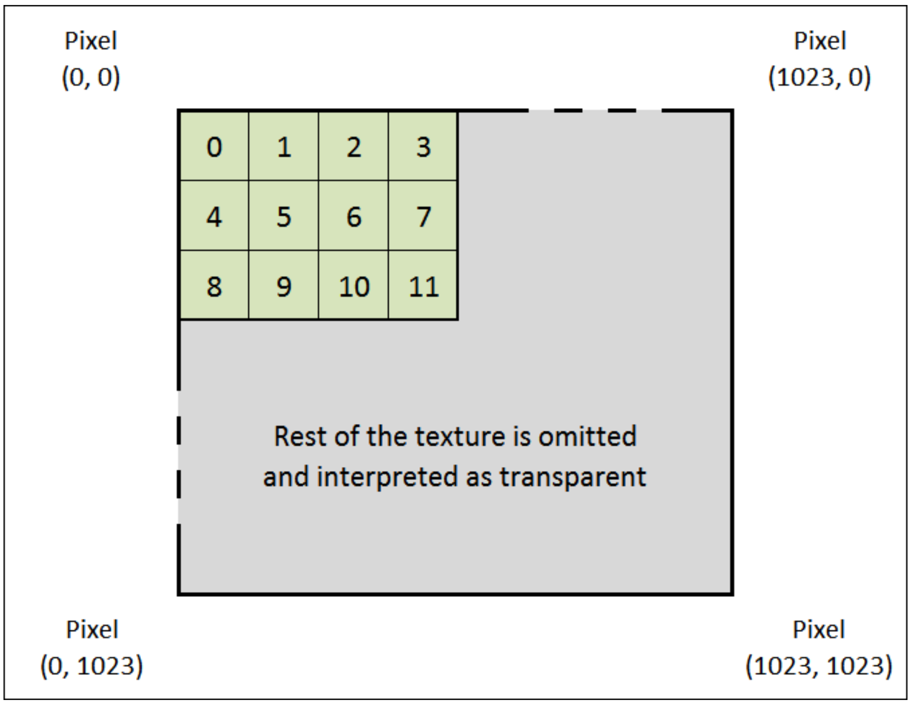
After the header, the file only contains all of those pixels in sequence.



5.1 Storage order for texture pixels

The stored texture area rectangle will be considered as the top-left part of the full texture. The rest of the texture pixels, up to the full size, are considered to have all 4 RGBA components equal to zero (i.e. fully transparent).

Pixels will be stored starting from the top-left corner, and advancing first left to right and then top to bottom. For example, for a stored texture of size 4 x 3 pixels, the storage order for those 12 pixels will be the one shown here.



5.2 Validations for GPU texture files

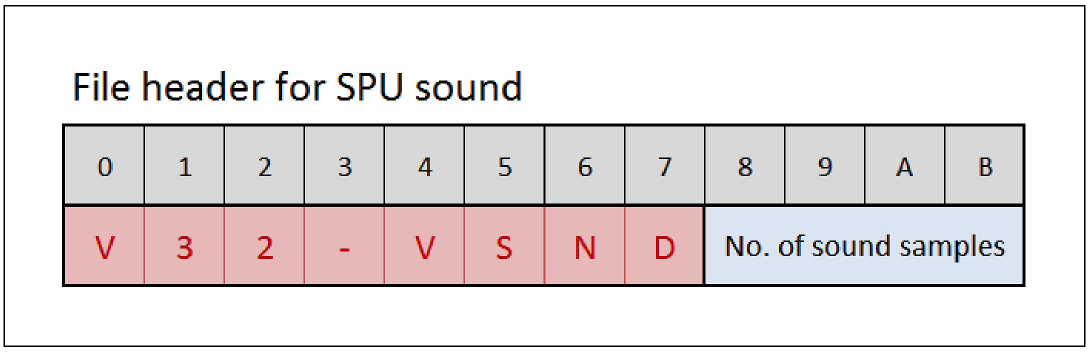
Aside from having the correct file signature, the following assertions can be tested to ensure that a GPU texture file is valid:

1) The declared texture size is valid. For both width and height, the range is from 1 up to 1024. Width and height don't have to be equal.

2) The file size in bytes must be 16 + 4 x { Width in pixels } x { Height in pixels }.

6 SPU sound files

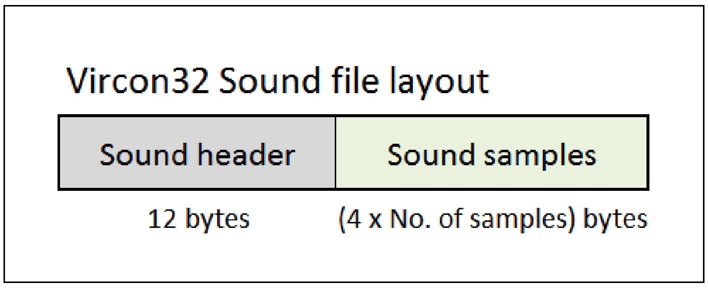
Aside from the file signature, the header of a SPU sound file only contains 1 integer field stating the number of 32-bit samples that compose this sound.



File header for SPU sound

0	1	2	3	4	5	6	7	8	9	A	B
V	3	2	-	V	S	N	D	No. of sound samples			

After the header, the file only contains all of those samples in sequence. As a reminder, Vircon32 will interpret all SPU samples as having a playback rate of 44100 samples/s.



Vircon32 Sound file layout

Sound header	Sound samples
12 bytes	(4 x No. of samples) bytes

6.1 Validations for SPU sound files

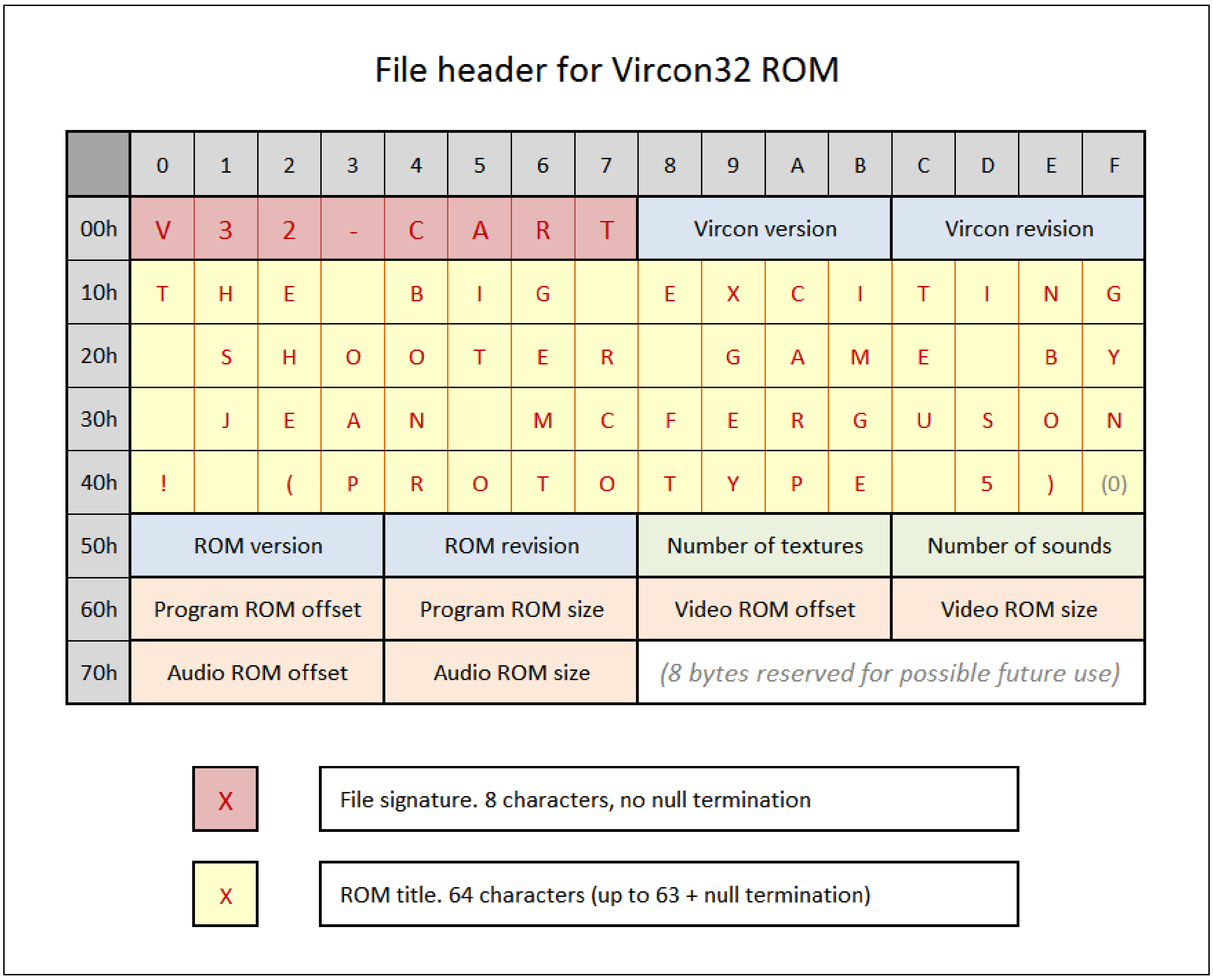
Aside from having the correct file signature, the following assertions can be tested to ensure that a SPU sound file is valid:

1) The declared number of samples is valid. The range is from 1 up to the applicable audio ROM size limit (see cartridge and BIOS limits on section 7).

2) The file size in bytes must be 12 + 4 x { Number of samples }.

7 Vircon32 ROM files

The structure of a ROM file is more complex than other formats. This is also reflected in its header. Its size is 128 bytes and it contains the following fields:



File header for Vircon32 ROM

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00h	V	3	2	-	C	A	R	T	Vircon version				Vircon revision			
10h	T	H	E		B	I	G		E	X	C	I	T	I	N	G
20h		S	H	O	O	T	E	R		G	A	M	E		B	Y
30h		J	E	A	N		M	C	F	E	R	G	U	S	O	N
40h	!		(	P	R	O	T	O	T	Y	P	E		5	)	(0)
50h	ROM version				ROM revision				Number of textures				Number of sounds			
60h	Program ROM offset				Program ROM size				Video ROM offset				Video ROM size			
70h	Audio ROM offset				Audio ROM size				*(8 bytes reserved for possible future use)*							

X	File signature. 8 characters, no null termination
X	ROM title. 64 characters (up to 63 + null termination)

Both cartridges and BIOS share this same file format. They are distinguished by their different file signatures: cartridges use **V32-CART**, while BIOS use **V32-BIOS**.

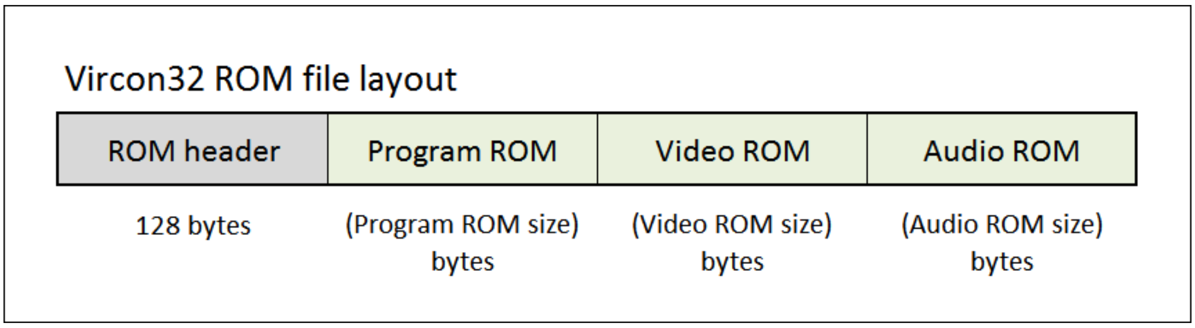
Fields colored in blue (versions and revisions) are pairs of integers to be interpreted together as a version number in the form <version . revision>. The current version of the Vircon32 standard is 1.0 (the first version, and therefore the only one supported).

ROM title and version are only provided as ROM metadata. Implementations are not required to process these informations, or even to read them at all.

Fields colored in orange (offsets and sizes) are also pairs of integers. They are used to delimit the 3 regions within the file that correspond to the 3 ROMs packed together in a

ROM file. These are the 3 components of an executable: Program ROM, Video ROM and Audio ROM.

The layout of a ROM file can be seen as just appending these 3 components to the header:



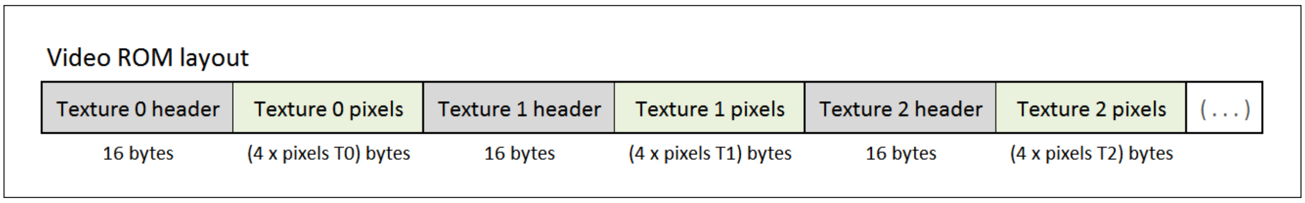
Offsets and sizes are all expressed in bytes and therefore should always be multiples of 4. Offsets are expressed as number of bytes from the beginning of the file. For instance, the Program ROM offset should always be 128 since it is located right after the header.

Note that, from these 3 components, the only mandatory one is the program ROM. Video and/or Audio ROMs will be absent if the declared number of textures and sounds are zero, respectively.

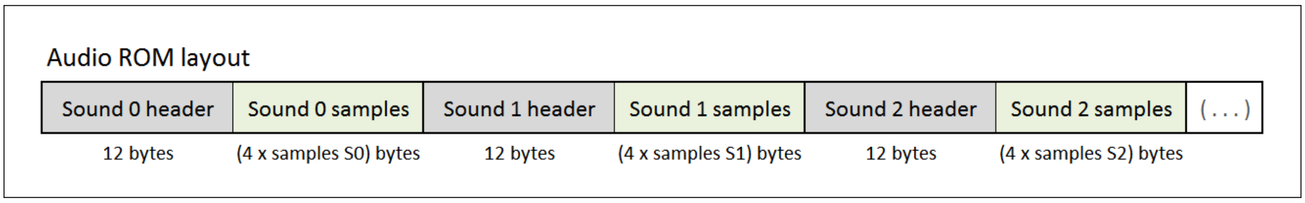
7.1 Layout of the 3 component ROMs

The 3 parts of a ROM file reuse the 3 file formats for assets described in previous sections. Here, each asset is stored simply by embedding its individual asset file into the ROM file. That way, the layout for the Program ROM will be the same as the layout of a program binary file (since there is always exactly 1 binary).

For the Video ROM, its layout will be a sequence of embedded GPU texture files stored one after another:



And similarly, the Audio ROM layout will consist of a sequence of embedded SPU sound files one after another:



7.2 Validations for cartridge ROM files

Aside from having the correct file signature, the following assertions can be tested to ensure that a cartridge ROM file is valid:

1) Stated Vircon version and revision must correspond to version 1.0.

2) The file size in bytes must be 128 + { Program ROM size } + { Video ROM size } + { Audio ROM size }

3) The declared number of textures must be in the range from 0 up to 256.

4) The declared number of samples must be in the range from 0 up to 1024.

5) All existing textures must have valid width and height, from 1 up to 1024 pixels.

6) The existing program binary must have a valid number of words, from 1 up to the cartridge program ROM limit of (128 x 1024 x 1024) words.

7) Each of the existing sounds must have a valid number of samples, from 1 up to the cartridge audio ROM limit of (256 x 1024 x 1024) samples.

8) The sum of samples for all of the existing sounds must not be greater than the cartridge audio ROM limit of (256 x 1024 x 1024) samples.

7.3 Validations for BIOS ROM files

Aside from having the correct file signature, the following assertions can be tested to ensure that a BIOS ROM file is valid:

1) Stated Vircon version and revision must correspond to version 1.0.

2) The file size in bytes must be 128 + { Program ROM size } + { Video ROM size } + { Audio ROM size }

3) The declared number of textures and sounds must both be 1.

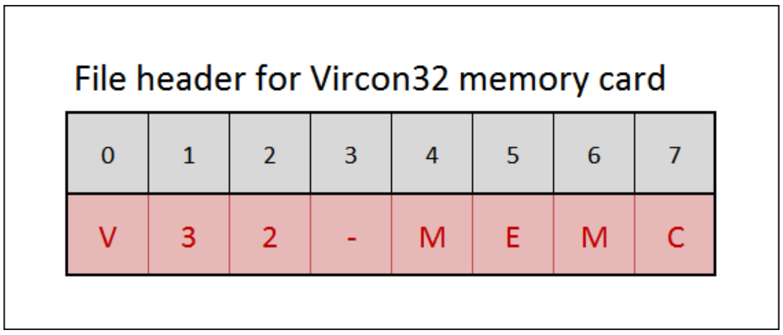
4) The existing texture must have valid width and height, from 1 up to 1024 pixels.

5) The number of words for the existing binary must be in the range from 1 up to the BIOS program ROM limit of (1024 x 1024) words.

6) The number of samples for the existing sound must be in the range from 1 up to the BIOS audio ROM limit of (1024 x 1024) samples.

8 Vircon32 memory card files

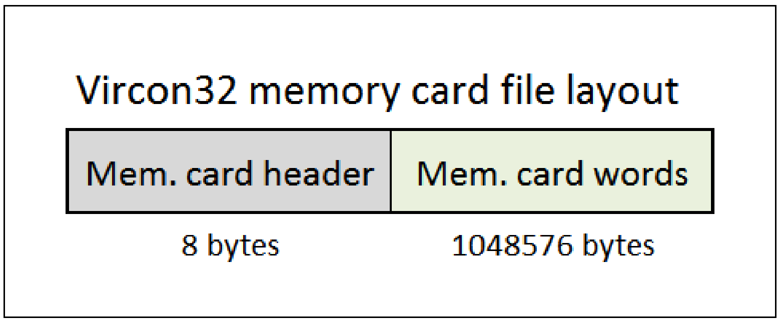
A Vircon32 system takes the contents of a memory card as just a usable range of memory positions, and its size is always exactly 256 KWords = 1MB. For that reason the header of a memory card file needs no more information than the file signature itself. The header for these files, then, is simply the following:



File header for Vircon32 memory card

0	1	2	3	4	5	6	7
V	3	2	-	M	E	M	C

After the header, the file only contains all of the card's 32-bit CPU words in sequence.



As a reminder, the first 20 words of the card's content are considered as the "game signature". Although these words are treated the same as the rest of the card, games will normally use them as a way to identify the contents of each card and prevent games from loading data intended for other games (with incompatible save formats).

8.1 Validations for memory card files

Aside from having the correct file signature, the following assertions can be tested to ensure that a memory card file is valid:

1) The file size in bytes must be 1048584.

(End of part 9)