A little diddy about binary file formats

by Kalid Azad · 28 comments Tweet

Understanding the nature of file formats and escape characters has been an itch of mine. I recently found a few useful explanations (http://www.faqs.org/docs/artu/ch05s02.html#id2901882) that inspired me to write my understanding of binary files.

How computers represent data

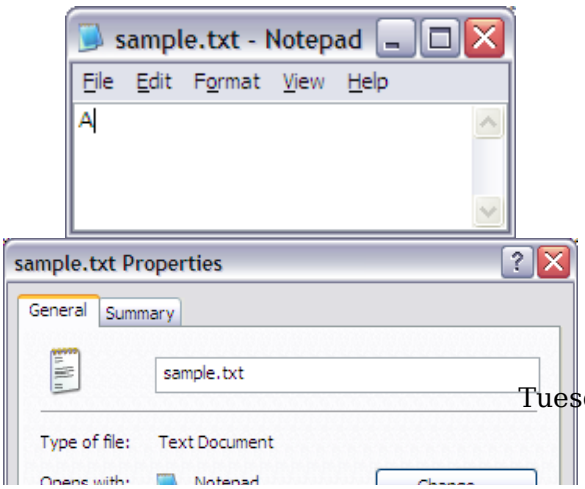
Everything is bits and bytes, 1's and 0's to the computer. Humans understand text, so we have programs that convert a series of 1's and 0's into something we can understand.

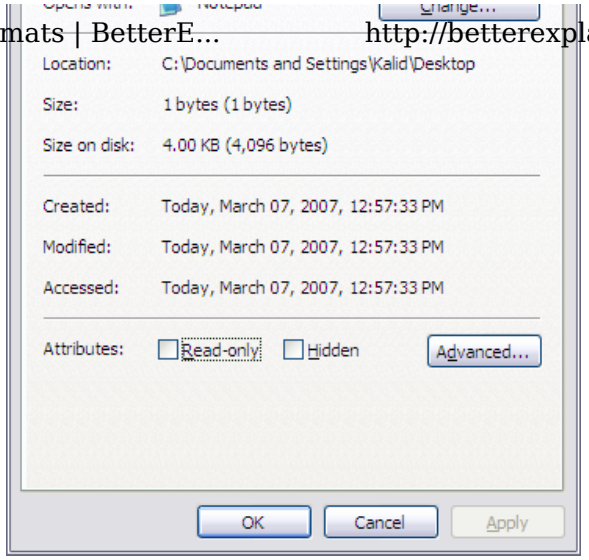
In the ASCII character scheme, a single byte (a sequence of eight 1's or 0's, or a number from 0-255) can be converted into a character. For example, the character 'A' is the number 65 in decimal, 41 in hex, or 01000001 in binary. 'B' is the number 66 in decimal, and so on (see a full chart (http://www.lookuptables.com/)).

Don't believe me? Mini-example time.

Create a file in notepad with the single letter "A" (any filename will do — "sample.txt").

Save the file, right-click and look the properties — it should be 1 byte: notepad stores characters in ASCII, with one byte per character. The "size on disk" may be larger because the computer allocates space in fixed blocks (of 4 kilobytes, for example).

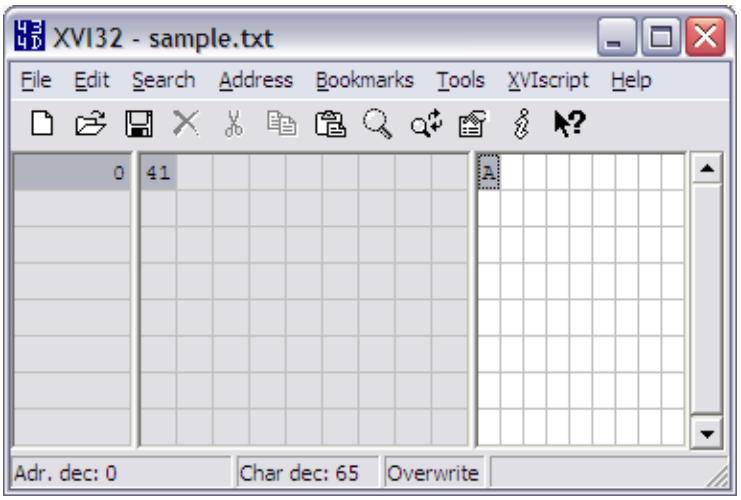




(http://betterexplained.com/wp-content/uploads/binary/notepad_A_size.png)

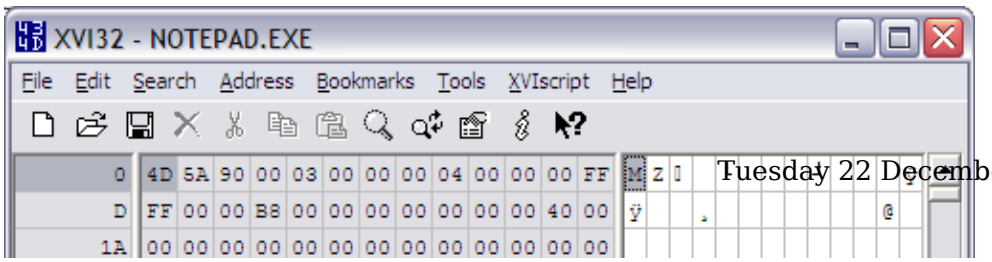
Find a hex editor (here's a free one (http://www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm#download)) and open the file you just saved. (On Linux/Unix, use "od -x sample.txt").

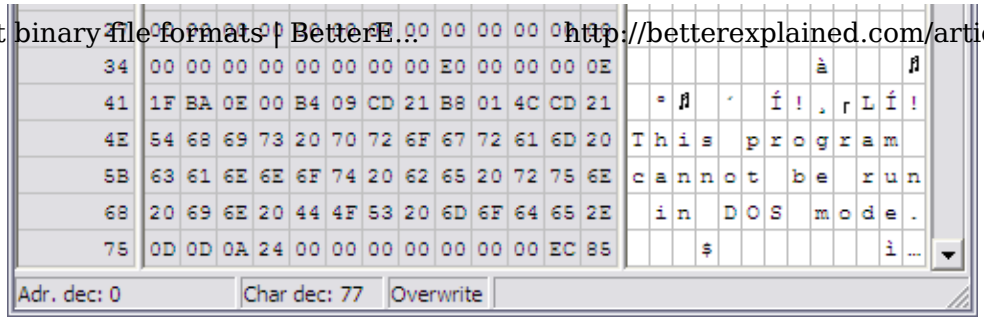
You'll see only the single number "41" in hexadecimal (65 in decimal), and the hex editor may show the character "A" on a side screen (the ASCII representation of the byte you are examining). The "0" on the left is the address of the byte — programmers love counting from zero.



The hex editor displays all data as ASCII text, which it is in our case. If you open up a non-ASCII file, the data inside will be displayed as ASCII characters, though it may not always make sense.

Try opening a random .exe to see what ASCII strings are embedded inside — you can usually find a few in the beginning portions of the file. All DOS executables start with the header "MZ", the initials of the programmer (http://en.wikipedia.org/wiki/Mark_Zbikowski) who came up with the file format.





Cool, eh? These headers or "magic numbers" are one way for a program to determine what type of file it's seeing. If you open a PNG image you'll see the PNG header (http://en.wikipedia.org/wiki/PNG), which includes the ASCII letters "PNG".

What's going on?

Inside the memory of the computer, only '65' (41 in hex or 01000001 in binary) is stored in sample.txt. Given the context of the information (i.e., notepad is expecting a text file) the computer knows to display the ASCII character 'A' on the screen.

Now consider how a human would store the actual *numeric value* of 65 if you told them to write it down. As humans, we would write it as two characters, a '6' and then a '5', which takes 2 ASCII characters or 2 bytes (again, the "letter" 6 can be stored in ASCII).

A computer would store the number "65" as 65 in binary, the same as 'a'. Except this time, software would know that the '65' was not the code for a letter, it was actually the number itself.

Now, suppose we wanted to store the number 4,000,000,000 (4 billion). As humans, we would write it as 4000000000, or 10 ASCII characters (10 bytes). How would a computer do it?

A single byte has 8 bits, or 2^8 (256) possible values. 4 bytes gives us 2^32 bits, or roughly 4 billion values (http://betterexplained.com/articles/mental-math-shortcuts/). So, we could store the number 4 billion in only 4 bytes.

As you can see, storing numeric data in the computer's format saves space. It also saves computational effort — the computer does not have to convert a number between binary and ASCII.

So, why not use binary formats?

If binary formats are more efficient, why not use them all the time?

- **Binary files are difficult for humans to read.** When a person sees a sequence of 4 bytes, he has no idea what it means (it could be a 4-letter word stored in ASCII). If he sees the 10 ASCII letters 4000000000, he knows it is a number.
- **Binary files are difficult to edit.** In the same manner, if a person wants to change 4

Billion to 2 billion, he needs to know the binary representation. With the ASCII representation, he can simply put in a "2" instead of the "4".

- **Binary files are difficult to manipulate.** The UNIX tradition has several simple, elegant tools to manipulate text. By storing files in the standard text format, you get the power of these tools without having to create special editors to modify your binary file.
- **Binary files can get confusing.** Problems happen when computers have different ways of reading data. There's something called the "NUXI" or byte-order problem (http://betterexplained.com/articles/understanding-big-and-little-endian-byte-order/), which happens when 2 computers with different architectures (PowerPC Macs and x86 PCs, for example) try to transfer binary data. Regular text stored in single bytes is unambiguous, but be careful with unicode (http://betterexplained.com/articles/unicode/).
- **The efficiency gain usually isn't tremendous.** Representing numbers in binary can ideally save you a factor of 3 (a 4 byte number can represent 10 bytes of text). However, this assumes that the numbers you are representing are large (a 3-digit number like 999 is better represented in ASCII than as a 4-byte number). Lastly, ASCII actually only uses 7 bits per byte, so you an theoretically pack ASCII together to get an 1/8 or 12% gain. However, storing text in this way is typically not worth the hassle.

One reason binary files are efficient is because they can use all 8 bits in a byte, while most text is constrained to certain fixed patterns, leaving unused space. However, by **compressing your text data** you can reduce the amount of space used and make text more efficient.

Marshalling and Unmarshalling Data

Aside: Marshalling always makes me thinks of Sheriff Marshals and thus cowboys. Cowboys have nothing to do with the CS meaning of "marshal".

Sometimes computers have complex internal data structures, with chains of linked items that need to be stored in a file. *Marshalling* is the process of taking the internal data of a program and saving it to a flat, linear file. *Unmarshalling* is the process of reading that that linear data and recreating the complex internal data structure the computer originally had.

Notepad has it easy – it just needs to store the raw text so no marshalling is needed. Microsoft Word, however, must store the text along with other document information (page margins, font sizes, embedded images, styles, etc.) in a single, linear file. Later, it must read that file and recreate the original setup the user had.

You can marshal data into a binary or text format — the word "marshal" does not indicate how the data is stored.

So when are binary file formats useful?

There are situations where you may want to use binary file formats. PNG images use a binary format because efficiency is important in creating small image files. However, PNG does binary formats right: it specifies byte orders and word lengths to avoid the NUXI problem.

There are often business reasons to use binary formats. The main reason is that they are more difficult to reverse engineer (humans have to guess how the computer is storing its data), which can help maintain a competitive advantage.

Posted in Programming (http://betterexplained.com/articles/category/programming/)

Questions & Contributions

Ask a Question Contribute an Insight

Have a question? Ask away.

Ask Question

Name (optional)

Have feedback? Just enter it above. I'm making a curated set of questions and insights for the article. Thanks!

Continue Discussion 28 replies

Jan '07

L

laila

Wow. I am loving this! You are awesome for putting stuff like this up. Thanks.

Jan '07

A

Anonymous_User

I just found this, really informative. Thanks.

Jan '07

kalid **Founder**

Awesome, I'm glad you found the articles useful!

Mar '07

A

anton

Thanks Kalid for the wonderful explanation. I wasn't sure that the funny characters represented next to hex values were their ascii equivalents and that they are not part of the actually file.

Mar '07

kalid **Founder**

Thanks Anton! I've just added a few screenshots to make the examples even clearer.

Sep '07

A

Anonymous_User

Thanks for the article. Your was the only one that I found through google that was relevant to the topic.

Sep '07

In This Series

About The Site

BetterExplained helps 400k monthly readers with clear, insightful lessons.

Calculus Course

Calculus, Better Explained (/calculus)Calculus insights within minutes. Free to read online (/calculus).

Math, Better Explained

Math, Better Explained (/ebook/math) A dozen math essentials (/ebook/math). Amazon bestseller.

"If you can't explain it simply, you don't understand it well enough." —Einstein (more (/philosophy/))