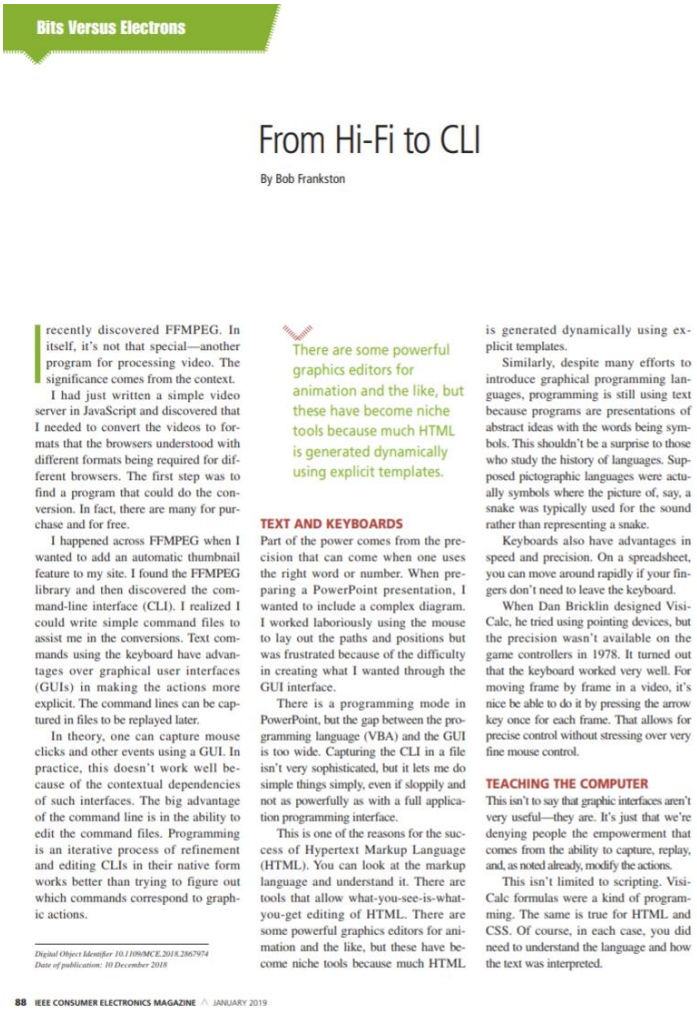
From Hi-Fi to CLI

Online Version

This is [this](#) is the version as it appeared in the magazine:



Bits Versus Electrons

From Hi-Fi to CLI

By Bob Frankston

I recently discovered FFMPEG. In itself, it's not that special—another program for processing video. The significance comes from the context.

I had just written a simple video server in JavaScript and discovered that I needed to convert the videos to formats that the browsers understood with different formats being required for different browsers. The first step was to find a program that could do the conversion. In fact, there are many for purchase and for free.

I happened across FFMPEG when I wanted to add an automatic thumbnail feature to my site. I found the FFMPEG library and then discovered the command-line interface (CLI). I realized I could write simple command files to assist me in the conversions. Text commands using the keyboard have advantages over graphical user interfaces (GUIs) in making the actions more explicit. The command lines can be captured in files to be replayed later.

In theory, one can capture mouse clicks and other events using a GUI. In practice, this doesn't work well because of the contextual dependencies of such interfaces. The big advantage of the command line is in the ability to edit the command files. Programming is an iterative process of refinement and editing CLIs in their native form works better than trying to figure out which commands correspond to graphic actions.

Digital Object Identifier 10.1109/MCE.2018.2867974
Date of publication: 10 December 2018

There are some powerful graphics editors for animation and the like, but these have become niche tools because much HTML is generated dynamically using explicit templates.

TEXT AND KEYBOARDS

Part of the power comes from the precision that can come when one uses the right word or number. When preparing a PowerPoint presentation, I wanted to include a complex diagram. I worked laboriously using the mouse to lay out the paths and positions but was frustrated because of the difficulty in creating what I wanted through the GUI interface.

There is a programming mode in PowerPoint, but the gap between the programming language (VBA) and the GUI is too wide. Capturing the CLI in a file isn't very sophisticated, but it lets me do simple things simply, even if sloppily and not as powerfully as with a full application programming interface.

This is one of the reasons for the success of Hypertext Markup Language (HTML). You can look at the markup language and understand it. There are tools that allow what-you-see-is-what-you-get editing of HTML. There are some powerful graphics editors for animation and the like, but these have become niche tools because much HTML is generated dynamically using explicit templates.

Similarly, despite many efforts to introduce graphical programming languages, programming is still using text because programs are presentations of abstract ideas with the words being symbols. This shouldn't be a surprise to those who study the history of languages. Supposed pictographic languages were actually symbols where the picture of, say, a snake was typically used for the sound rather than representing a snake.

Keyboards also have advantages in speed and precision. On a spreadsheet, you can move around rapidly if your fingers don't need to leave the keyboard.

When Dan Bricklin designed VisiCalc, he tried using pointing devices, but the precision wasn't available on the game controllers in 1978. It turned out that the keyboard worked very well. For moving frame by frame in a video, it's nice be able to do it by pressing the arrow key once for each frame. That allows for precise control without stressing over very fine mouse control.

TEACHING THE COMPUTER

This isn't to say that graphic interfaces aren't very useful—they are. It's just that we're denying people the empowerment that comes from the ability to capture, replay, and, as noted already, modify the actions.

This isn't limited to scripting. VisiCalc formulas were a kind of programming. The same is true for HTML and CSS. Of course, in each case, you did need to understand the language and how the text was interpreted.

88 IEEE CONSUMER ELECTRONICS MAGAZINE JANUARY 2019

It is also available online in [HTML](#) format.

Back to CLI

I recently discovered `FFMPEG`. In itself, it's not that special – another program for processing video. The significance comes from the context.

I had just written a simple video server in JavaScript and discovered that I needed to convert them to formats that the browsers understood with different formats for different browsers. The first step was to find a program that could do the conversion. In fact, there are many for purchase and for free.

I happened across `FFMPEG` when I wanted to add an automatic thumbnail feature to my site. I found the FFMPEG library and then discovered the command line interface (CLI). I realized I could write simple command files to assist me in the conversions. Text commands using the keyboard have advantages over graphical user interfaces (GUIs) in making the actions more explicit. The command lines can be captured in files to be replayed later.

In theory, one can capture mouse clicks and other events using a GUI. In practice, this doesn't work well because of the contextual dependencies of such interfaces. The big advantage of the command line is in the ability to edit the command files. Programming is an iterative process of refinement and editing CLIs in their native form works better than trying to figure out which commands correspond to graphic actions.

Text and Keyboards

Part of the power comes from the precision that can come when I use the right word or number. When preparing a PowerPoint presentation, I wanted to do a complex diagram. I worked laboriously using the mouse to lay out the paths and positions but was frustrated because of the difficulty in creating what I wanted through the GUI interface.

There is a programming mode in PowerPoint, but the gap between the programming language (AKA the Macro Language) and the GUI is too wide. Capturing the CLI in a file isn't very sophisticated, but it lets me do simple things simply, even if sloppily and not as powerfully as with a full API.

This is one of the reasons for the success of HTML. You can look at the markup language and understand it. There are tools that allow WYSIWYG editing of HTML. There are some powerful graphics editors for animation and the like, but these have become niche tools as much HTML is generated dynamically using explicit templates.

Similarly, despite many efforts to introduce graphical programming languages, programming is still using text because programs are presentations of abstract ideas with the words being symbols. This shouldn't be a surprise to those who study the history of languages. Supposed pictographic

languages were actually symbols where the picture of, say a snake, was typically used for the sound rather than representing a snake.[i]

Keyboards also have advantages in speed and precision. On a spreadsheet, you can move around rapidly if your fingers don't need to leave the keyboard. When Dan Bricklin designed VisiCalc[ii], he did try using pointing devices, but the precision wasn't available on the game controllers in 1978. It turned out the keyboard worked very well.

For moving frame by frame in video, it's nice be able to do it by pressing the arrow key once for each frame. That allows for precise control without stressing over *very* fine mouse control.

Teaching the Computer

This isn't to say that graphic interfaces aren't very useful – they are. It's just that we're denying people the empowerment that comes from the ability to capture, replay and, as noted above, modify the actions.

This isn't limited to scripting. VisiCalc formulas were a kind of programming that was nonlinear but could still be understood by looking at the text. The same is true for HTML and CSS. Of course, in each case, you did need to understand the language and how the text was interpreted.

When we take a command line and put it in a file, we are, in essence, teaching the computer how to repeat the action. Yes, it is programming, and there is no need to be frightened. Anyone who writes down a recipe or gives directions is doing that kind of programming.

Which brings us to Hi-Fi or High-Fidelity audio that was a major advance in consumer electronics in the 1950's along with television. The consumer programmed the system by wiring together components using a simple cable with a standard connector. At the time the wide market simply bought phonographs to play records at 45RPM. But the sophisticated users bought components and assembled them and used quality speakers. These are the users who drove innovation in the market just like games do today.

Sloppy is Good

In Unix (or the Mac) the command line interface is the "terminal" program reflecting its roots from the days of teletypes. In Windows you "`cmd`" (it's also called the DOS box). Windows has other more, sophisticated command windows or shells including the full Unix terminal, but I still use the familiar command interface.

```
for %i in (*.jpg) do echo %i.jpg
```

In Unix and on the Mac, you can write

```
for i in *.jpg; do echo $i; done
```

That CLI line will type out the names of the jpg files in a directory. This form allows me to call command files that don't support wildcards. Note the term "type" for presenting the names on the screen. There is a lot of legacy terminology just like in real languages. Fortunately, you can do a lot with only a few basic commands and learn more as you want. And there is so much more. The NETSH command in Windows allows for sophisticated management of network functions.

In creating the scripting file for converting the video files I couldn't quite get it to do what I wanted without further programming. I settled for a script that did most of what I wanted and then I could manually rename the file.

(As an aside, as part of the process, I also learned more about video files and codecs … such learning is a nice by-product of exploring new directions).

Sloppy is OK because I'm doing something for my own use and not creating a production tool for millions of users. This reflects the real world where good enough is often the best approach. The fallacy of perfection leads us to embrace the notion of home automation as if you can just script one's life. Even more problematic is that those scripts also control others' environments. Hence the frustrations with today's smart home market that tries to do me too many favors instead of giving me tools to do simple things simply.

Helping Others

One of the big advantages of the command line is that you can explain it to a friend without having to see the screen. In Windows "`R cmd`" will create a command window. If you want to find the network settings, just have your friend type

```
ipconfig
```

and then tell you what it says. (Unix has ifconfig but may need to be installed).

This is far easier than trying to guide one through an unseen graphical interface.

Empowering

Consumer Electronics isn't just about delivering canned solutions to users. In the same way, that component audio (Hi-Fi) allowed users without engineering degrees to build electronic systems software gives us all access to be creators and tap into the power of programming without having to be "programmers".

The power of the web came from giving people the ability to create rather than just consume. Facebook and Mobile-First may be the face of the present, but we must enable the future by giving consumers the ability to do more than click hither and yon.

Enabling Innovation

One final takeaway that came from writing my video server was discovering that I didn't really have to write a video server. I simply had to respond to a request for a range of a file. The rest is done client-side. This is an important lesson. It's not that video is trivial but if we effectively architect a system we can get tremendous leverage and enable rapid innovation. There isn't any one single proper architecture nor one perfect solution.

By providing adequate enabling technologies we enable thousands of early adopters to empower millions more to innovate and create rather than simply consume.

[i] Coulmas, Florian – Writing Systems of the World.

[ii] http://VisiCalc.us or you can just substitute “Excel” to understand the point.