An Apple ///

Guide for Humans

Alan Anderson

Apple's official company line about the Apple /// is that the /// is the most powerful personal computer in its class. The truth, of course, is that the Apple /// is the most powerful personal computer in its class. But if you've just bought an Apple /// for your business, how is that power translated into benefits that you can see? While it's true that the /// has something called a Sophisticated Operating System, or SOS, what does that mean in the real world?

In many ways, the Apple /// is not remarkably different from the personal computers which came before it. It still runs the traditional high-level languages, BASIC and Pascal. It still talks to you via a screen or printer, and it is still talked to with a typewriter keyboard. However, those sneaky folks at Apple hid the real power of this thing deep inside. The most significant real advantage of the Apple /// is the way the computer manages its resources, that is, the operating system. Don't discount the importance of that one point, though. It can make all the difference in the world, and that's what this article is about.

One of the single most important things that SOS does is make it easier for programmers to create more sophisticated, easier to use software. This becomes more obvious as the programmer gets closer to the computer itself (*i.e.*, Assembly language), and is less obvious in the high level languages, such as Pascal and BASIC. Since not very many folks have poked around in Assembly language on the ///, the significance of SOS in simplifying programming has not been all that well-reported.

Does SOS have any importance if you're not interested in doing any programming at all, just in using off-the-shelf software? The answer to that is an emphatic "Yes!" In addition to making life easier for programmers, SOS does a number of things to increase the applications software user's control over the system. These things fall mainly into two categories: configurability and device independence.

Now that I've thrown out a couple of buzzwords, let's work on translating them into English. The first is configurability. This simply means that you have the ability to add new devices, change others, and all the time (this is the significant part) retain compatibility with your software. What does that mean? It means that, in general, more software will work with more hardware. The Apple ///'s SOS lets you plug in devices like printers, mass storage devices, modems, and anything else, and handles much of the necessary translation between the computer and the external device, so the software author doesn't have to worry about it.

Does this system work? Yes, it does. As an example, consider the release last fall of the ProFile hard disk drive. As soon as this drive was released, it was instantly usable for data files by virtually all Apple /// software. While this may not seem terribly remarkable if you haven't been around this business for a long time, the fact is that the release of a new disk drive generally requires some "patches" or modifications to the applications software before it can be used. A living example of this is the IBM Personal Computer's recent introduction of larger-capacity disk drives. While the drives are there, the software can't do anything with them until it is rewritten.

The reason SOS is able to perform this feat is through the magic of complex little programs called device drivers. A device driver is a sort of translator between the device itself and the application program. As an example, the device driver for the floppy disk drives translates your request to load your budget file into VisiCalc into the proper series of head movements and motor controls feeded to retrieve the information. This means that almost any device can be connected to the Apple ///, if a device driver is written for it.

The other important end-user benefit of SOS is something called device independence. This item is related somewhat to configurability, but it moves one step further. Device independence means that all the things plugged into the Apple ///, such as keyboard, screen, disk drives, printers, modems, voice synthesizers, magnetic card readers, and anything else real or imagined, are all treated equally under one big umbrella called devices.

What does that do for you? Let's say you're about to print out your VisiCalc budget. You give VisiCalc the "/**P**" command, and it says "Print: File or Printer". Normally, you just press **P** for printer, and the report shows up at the printer, but what happens if you choose the other option, File? Since SOS treats all devices equally, you can also "print" your budget somewhere else — like to a file on disk, for example. Think about this for a minute. When you print your file, all that happens is that each character to be printed is shipped off, one by one, to the designated device. That device then processes it however it wishes. If the device is a printer, the process involves making the printer create the character on a piece of paper; if it's a disk drive, it stores that character in a disk file; if it's a modem, it sends the character across the phone line; if it's a voice synthesizer, it speaks the character; you get the idea.

The reason this becomes useful is that it allows you to create on the disk an exact "picture" of a VisiCalc report that you can then load into your favorite word processor and edit, enhance, or add a report to. It's not just VisiCalc files, of course. Almost any program which prints reports and lets you specify where the report is going can be used the same way: Quickfile and PFS, for example, can do the same thing.

The Path to Success

I've discussed the fact that the Apple ///'s SOS presents you with a world of devices: disk drives, printers, modems, and more. In order to get the most out of your system, it's important to know how to communicate with all these guys. The fundamental rule is simple: all devices have a name, and in order to cause information to come from or go to a certain device, you need only know its name. In this section, we'll talk about those names, where they come from, and how to use them.

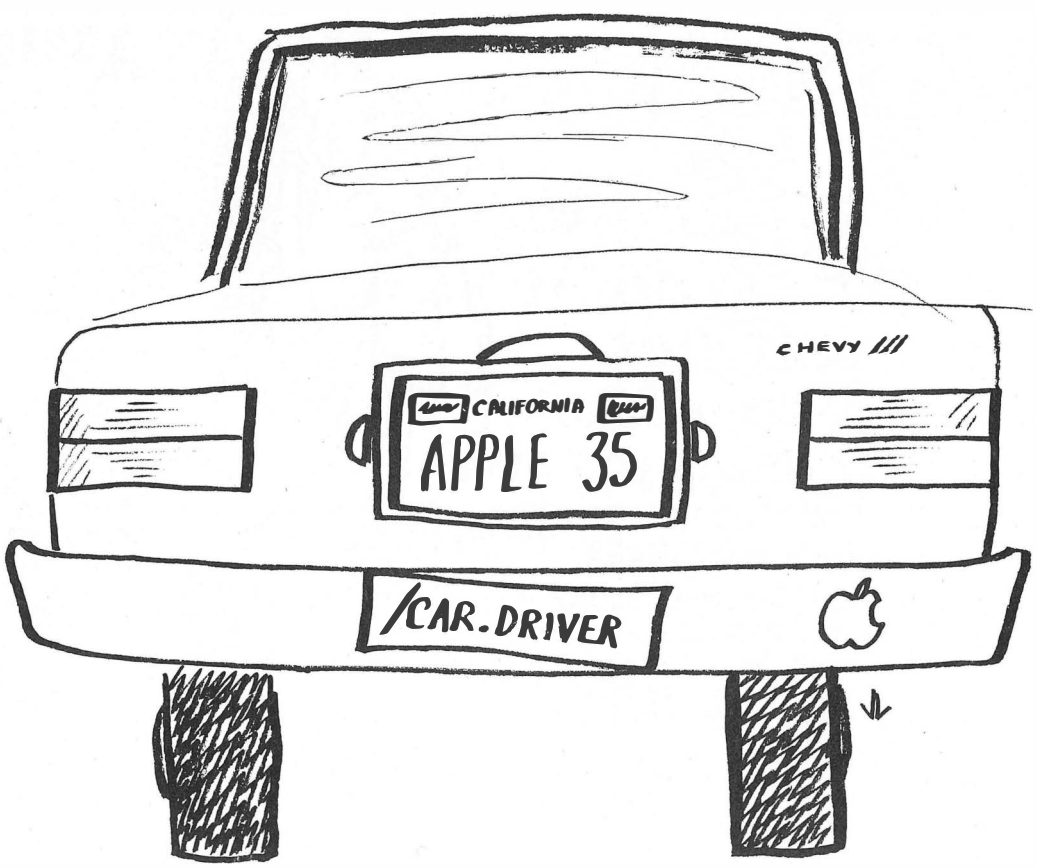
Earlier, I mentioned that every device is associated with a program called a device driver. This is the source of the device's name. The name is written into the driver and can be changed with the System Configuration Program (SCP) on the System Utilities diskette. There are a few rules for these names: they must begin with a "." (called "dot" and not "period" in the jargon); the dot must be followed by one to 14 more characters (so the whole name is between two and 15 characters long); the second character must be alphabetic, and the remaining characters must be alphabetic or numeric.

The Apple /// has a few special devices, special mainly because of their importance. These include: **.D1**, **.D2**, **.D3**, and **.D4**, which are the drivers for the four possible floppy disk drives; **.CONSOLE**, which drives the keyboard and screen as one device; **.PRINTER**, which drives a printer through the built-in RS-232 interface; **.GRAFIX**, which drives the graphics screen, treated as a separate device from the .CONSOLE screen, which displays text; **.RS232**, which is the driver for a modem hooked up through the built-in RS-232 port; and **.AUDIO**, which is the driver for the speaker.

The driver for the Silentype printer is usually called **.SILENTYPE**. I say "usually" because, as noted earlier, you can change device names with the System Configuration Program. So, **.SILENTYPE** may occasionally be called **.PRINTER**, as it is in VisiCalc ///, for example. How do you know what's going on? You must rely on the documentation or your local dealer, or you must investigate with SCP to see which driver is really which. Actually, this problem isn't as bad as it sounds, since most device drivers hold on to their standard names all the time. About the only confusion that exists is with different printer drivers acquiring the .PRINTER name. If you have only one printer, you should probably name it .PRINTER, and you'll never have to worry.

Some devices, like printers, modems, and the console, either take in or put out one character at a time. These devices are called **character devices** (clever name). There is another group of devices, however, which can handle (1) large chunks of information at one time, and (2) many different groups of information, or files, at one time. These are called **block devices**, and they are, of course, disk drives. This distinction is important because, when referring to the printer, **.PRINTER** tells the data where to go. But if you want to record your VisiCalc printout on the disk, just entering **.D1** or **.PROFILE** won't do; you must add a File Name to the device name.

Let's talk about file names. Their rules are similar to device names, but without the dot: one to 15 characters long, the first character must be alphabetic, the rest must be alphabetic, numeric, or dots. So, when you want VisiCalc to print a report to a disk file, you could direct it to **.D1/REPORT**. The "slash" character separates the device name from the file name. The entire specification for a file, any file, is called a



pathname, since it specifies the path that must be taken to get to the file.

Every diskette has a name, too. The rules for diskette names are exactly the same as file names: one to 15 characters long, the first character must be alphabetic, the rest must be alphabetic, numeric, or dots. Actually, these are not called diskette names, but are more correctly called **volume names** (think of disk *volumes* as in a *library*), since the name applies to any medium in a block device, such as a hard disk drive, tape drive, or even a high-speed cassette drive. This name is assigned by the System Utilities diskette when the volume is formatted, and can be changed with the RENAME command the same way a file name is changed.

Let's review some of what we've gone over so far: every physical device has a name, like **.PRINTER**, **.PROFILE**, or **.D1**; every mass storage volume has a name; and every mass storage volume can have many files, each with its own name. If you have a diskette named "ROGER" and you place it in the built-in disk drive, do you address it as .D1 or ROGER? You can use *either* name. Note that if you address ROGER, the Apple will valiantly search for a volume called ROGER, wherever it may be. If you use .D1, the computer will go to the built-in drive, no matter what diskette is there, as long as it's a valid SOS diskette.

One phenomenon that occurs when using a hard disk drive like ProFile is that it begins to accumulate a remarkable number of files, just like you used to accumulate diskettes before you had the ProFile. Under most operating systems, having a few hundred files on a hard disk drive was a massive organizational pain. However, SOS provides for something called subdirectories, which are ways to carve your ProFile up into smaller, more manageable pieces, each piece with a name, just like any other file. Actually, subdirectories can be built on any mass storage volume, but they're most practical and important on ProFile. For example, you could have one subdirectory called **VC.FILES** for all your VisiCalc spreadsheets; one called **BG.FILES** for your Business Graphics data, and so on.

These files are addressed simply by adding the name of the subdirectory to the pathname. For example, if we wanted to store our VisiCalc report in a subdirectory called VC.FILES on the ProFile, we could use a pathname of **.PROFILE/VC.FILES/REPORT**, with the slashes separating each level of the pathname.

You can see that entering an entire pathname for a file in a subdirectory can be tedious — the above pathname, for example, is 24 characters long. Once again, SOS provides us with a tool to make it easier: the Prefix. The prefix is part of a pathname that specifies a default path for disk files. For example, we could set the prefix to read .PROFILE/VC.FILES. If we then specified file names such as REPORT, SOS would automatically add the prefix to the front of REPORT, giving the full pathname. This is especially handy when using a number of files in the same subdirectory or on the same volume.

How do you set the Prefix? Most applications programs and language systems provide a way to do it. In BASIC, the prefix is contained in a variable called PREFIX$. In Pascal and in the System Utilities program, it's set in the Filer. VisiCalc sets the prefix implicitly based on the last pathname you used. AppleWriter /// has a menu option to set the prefix.

One other point crops up when talking about addressing files in the Apple ///. What happens if the prefix is set to address the ProFile and you want to save a file onto a floppy disk? How can you cause the prefix to be temporarily ignored so that you can address a file somewhere else? The rule is this: if the pathname you enter starts with a dot or a slash, the prefix will not be used. That means that if the prefix is set for the ProFile and you want to save a file on Drive 1, all you have to do is specify **.D1/FILENAME** and the prefix will be ignored because of the leading dot. Similarly, if you wanted to save a file on the floppy disk called ROGER and the prefix was set for the ProFile, you would specify a path of /ROGER/FILENAME and again the prefix would be ignored, this time because of the leading slash.

Please note that the slash has two separate and distinct functions: one is to separate the levels in a pathname, and the other is to suppress the prefix when it is the first character in a pathname. Not realizing that this character has two meanings has caused an awful lot of confusion in the Apple /// world.

Congratulations! You now know an awful lot about how to use your Apple /// system to its fullest power. If everything didn't catch on, try re-reading this article and see if that helps. With some practice, I hope you'll see that the Apple /// and SOS provide power not just for programmers, but for regular humans, too.