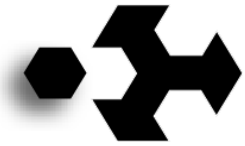
White Paper: System Neutral Access Protocol

SciTech SNAP - The future of Device Driver Technology

This document contains SciTech Software confidential and proprietary information



SciTech Software Inc.

180 E 4th St, Suite 300
Chico, CA 95928
(530) 894-8400
(530) 894-9069 FAX
www.scitechsoft.com

Revised (10/1/02)

Contents

Introduction

This white paper describes the SciTech System Neutral Access Protocol device driver architecture, or SciTech SNAP for short. SciTech SNAP represents a new paradigm for device driver development, quite different from traditional methods commonly employed in the computer industry today. This new driver architecture is the result of years of research and development by SciTech into solving the problems that plague traditional device driver development.

Although many of the examples in this document refer to graphics hardware devices, the same problems afflict other hardware such as audio, communications, network and mass storage devices. The solutions presented in this document are directly applicable to those devices as well.

The "Driver Crisis"

Recognizing the Crisis

A device driver is the software component that allows operating systems and application software to use a hardware device. Under ideal circumstances, every user would have a properly functioning device driver regardless of which company manufactured their hardware, its age, which operating system they choose, or what processor they are using.

The "Driver Crisis," as illustrated in the quotations above, is the dilemma the computer industry currently faces due to the lack of sufficient device drivers. Most computer users are affected by the Driver Crisis, since the functionality, stability and performance of their computer system is entirely dependent upon the availability and proper operation of the device drivers for their specific hardware and operating system combinations.

The Popularity of Open Architectures Precipitated the Driver Crisis

In the past, some computer system vendors have attempted to avoid the Driver Crisis by building specialized, closed architecture computer systems running proprietary operating systems. Closed architecture computer systems have the potential to avoid the Driver Crisis since the manufacturer has control over all devices and the entire operating system. The personal computer industry is moving away from closed architecture systems since there is a much wider choice of hardware and software when open architectures are employed.

The PC architecture, which evolved from the original IBM Personal Computer, is fundamentally different from closed architectures because the it uses various standards which allow multiple system vendors and operating system vendors to compete using the same basic hardware design. Today's PC architecture scales all the way from low cost, sub $200 systems up to workstations and servers costing well over $20,000. Such a vast price range for PCs means that there will be many different installed hardware devices, but they can run can run most of the same operating systems and applications.

The Operating System Vendor Driver Crisis

This wide choice of hardware generates a Driver Crisis for operating system vendors, since they need to ensure their operating system is compatible with hardware devices already in existence and those new devices entering the market. It is not cost effective for operating system vendors to build drivers for every hardware device, so operating system vendors try, often unsuccessfully, to persuade hardware manufacturers to develop and support drivers for their operating system.

The Driver Crisis for operating system vendors generally affects smaller, emerging operating systems more than larger more established operating systems. However even the larger established operating system vendors are affected by the driver crisis through driver and hardware aging as we will see later.

The Hardware Vendor Driver Crisis

The need to support the multiple operating systems and processors that their customers are using generates a Driver Crisis for the hardware manufacturer. The intense competition in the PC hardware industry means that vendors cannot justify using their slim profit margins to build drivers to support all possible operating systems for all the hardware devices they have manufactured. Some of the reasons for this include:

- Hardware vendors tend to focus on driver performance over compatibility to get favorable product reviews. Product reviews help sell hardware, and performance is often over emphasized in those reviews.

- Lack of ongoing revenue to justify updating drivers for existing products. Due to slim profit margins and limited software development resources, many hardware vendors focus on supporting only their latest hardware with the latest operating systems at the expense of users with older hardware or “non-strategic” operating systems and processors.

- Hardware vendors are primarily in business to sell hardware. They often have a short-term mentality, focused primarily on selling new hardware products, often at the expense of supporting their installed base. For many hardware vendors, software components are secondary to their primary product. Device drivers are simply a necessary evil required to capture the initial sale of the hardware product.

The End User Driver Crisis

The final result is that the drivers provided by the hardware vendors are often of less-than-adequate quality, which in turn results in the Driver Crisis for the end user. The end user has no choice but to complain to the hardware vendor or the operating system vendor. In many cases where ongoing revenues cannot justify the expense of supporting users, hardware vendors inform users the only solution is to upgrade their hardware or switch operating systems.

The Driver Crisis Worsens over Time

A device or operating system has different levels of support depending on its popularity and its life cycle stage. Over time, an operating system that may have once had good device support ages to the point where its device support is less than adequate. Conversely, a device driver will age so that new features and operating systems are not supported and the driver is no longer actively maintained. Table 1 defines some examples of current operating system tiers (note: these will change over time):

Tier 1 PC Operating System Examples:	Tier 2 PC Operating System Examples:
Windows Me Windows 2000 Windows XP	**Emerging** Linux Windows CE MacOS X **Existing** MS-DOS OS/2 Windows 3.x Windows 95 Windows 98 Windows NT 3.5x Windows NT 4.0 **Lower Volume/Specialty** QNX FreeBSD Solaris Other Unix Variants Other real time and embedded OS's

Table 1: Operating System Examples and Categories

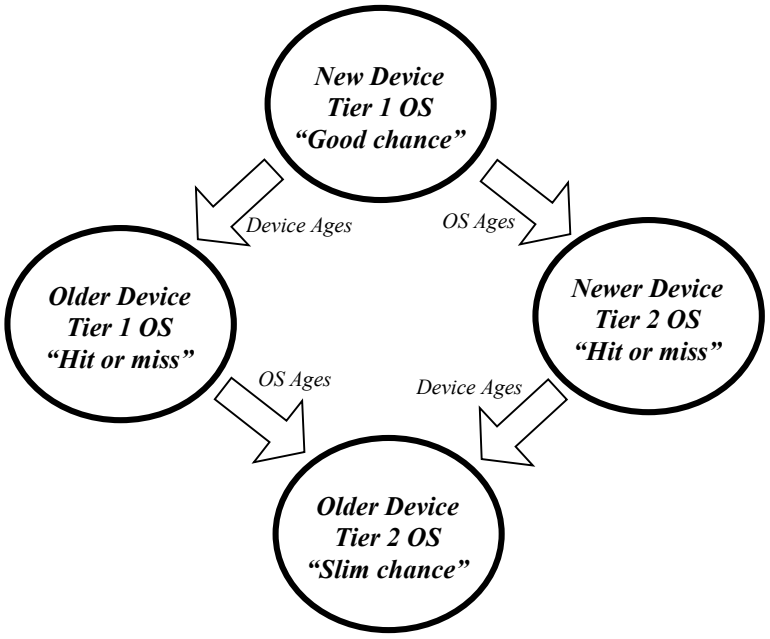


Fig. 2: Driver Aging over Time

As depicted in Figure 2, users purchasing new hardware running Tier 1 operating systems may start out with a good chance of having adequate driver support. However, over time both the hardware devices and the operating systems age. As the hardware devices age, users upgrading to the latest operating systems find themselves moving into middle-left left group and the chance of getting good drivers becomes 'hit or miss'. Likewise, as the operating system itself ages, it becomes a Tier 2 operating system and the chance of getting good drivers decreases.

With the current frantic pace of hardware development, eventually most users end up in the bottom group within a short time after they purchase their new computers. This leaves them with a 'slim chance' of finding drivers that will work properly with their hardware.

Root Causes of the Driver Crisis

Using traditional driver development methods, it is nearly an impossible task to support all the operating system and hardware devices that users may need. Since hardware vendors are forced to choose amongst these options, the result is that they tend to focus on supporting the latest chips (i.e. revenue generating devices) on the most popular operating systems and the most popular processor platforms.

Overview of the Traditional Device Driver Model

The traditional device driver development model for today's operating systems consists of three layers, as depicted in Figure 1:

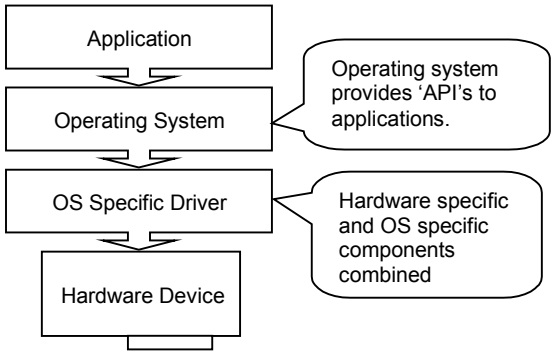


Fig. 1: Traditional Operating System Driver Model

Application programs communicate with the operating system via Application Programming Interfaces, or APIs, unique to each operating system. The operating system, in turn translates these service requests into device specific requests to the hardware device through a device driver.

The device driver layer implements all device specific code as well as the higher-level features that the operating system requires for that class of hardware device. The device driver layer is defined differently for each and every operating system, which then requires the hardware vendor to develop individual device drivers specific to each supported operating system. These OS specific device drivers use up precious hardware vendor resources, precipitating the driver crisis for the hardware vendors.

Non-unified Source Code

One of the biggest flaws in traditional device driver development is the lack of unified source code between drivers for different operating systems, and between drivers for different hardware devices. Differing device driver layers between operating systems usually means that, during driver development, the source code for the equivalent drivers on each operating system is developed from scratch, or source code from a similar operating system is copied and the driver is converted to the new operating system. The end result is that the driver source code for each operating system is different, even though the drivers for different operating systems provide essentially the same functionality.

This means that any generic device driver bug fixes or hardware workarounds discovered on one operating system must be re-discovered and fixed independently for each operating system in turn. Likewise, any performance enhancements implemented in a driver for one operating system must be manually added to the code for all other operating systems. As the number of devices and operating systems supported increases, it becomes unmanageable for a hardware vendor to maintain device drivers for all but the most popular operating systems.

Recently a number of hardware manufacturers have started implementing ‘Unified Drivers’ that share common source code across different operating systems and chipsets, but their solutions do not completely eliminate the driver crisis.

Code Rot

The process of building a device driver involves compiling source code into a binary module that runs on a specific operating system and processor platform. Every time the source code is compiled into a binary module, the resulting binary module must be tested to ensure that it still functions correctly. If the code is not fully tested, there is no guarantee that it will function correctly. The lack of testing allows Code Rot to cause source code that used to function correctly to cease to function correctly. Code Rot can occur in source code that has never been changed directly, but is affected by changes to code that interfaces with the original unchanged source code. Such changes may be caused by platform header file changes when upgrading to a new version of a DDK, or by changes in the internal API’s used to build the device driver as support for new hardware devices and functionality is implemented.

Code Rot can also occur due to changes in shared source code modules, especially a ‘Unified Driver’ environment where a lot of source code is shared amongst different chipset modules. When support for many devices is compiled into a single binary module, there is a much greater chance that changes to source code unrelated to the specific device in question could cause Code Rot. This is especially prevalent where there is a lot of common source code shared amongst each specific supported device. Programmers working on support for new chipsets can easily cause Code Rot in code for other devices that they are not actively testing.

Another less prevalent but important form of Code Rot can occur due to changes in the compilers and tools used to produce the binary modules. Bugs introduced in new versions of development tools can cause Code Rot, and even bug fixes in development tools may effect code that worked previously due to the bug in the original development tools!

The final result is that the only way to eliminate Code Rot is to completely test the resulting binary modules for correct operation ***every*** time the compiled module is released to unsuspecting end users.

Large Test Matrices

To avoid Code Rot, completely testing and verifying the operation of each support device is very important. However verifying correct operation of a device on a specific operating system and hardware device requires complex and time consuming testing procedures. Since this testing process is so complex and time consuming, hardware vendors often do not update older drivers with new features and bug fixes as a simple way to avoid Code Rot. A classic sign of this approach is when you see different versions of the same device driver listed in the hardware vendors download page depending on which specific operating system device you need support for. The downside of this approach is that older devices do not get the latest bug fixes, features and operating system support that the newest device gets.

In cases where the hardware vendor uses a ‘Unified Driver’ approach, the same device driver source code can be built to support multiple hardware devices and operating systems. However due to the large test matrices involved, the driver is often not completely tested with older hardware during each new release cycle (or the release cycles are few and far between due to the testing requirements). The end result is that the ‘Unified Driver’ is either not released with support for earlier hardware devices or older operating systems, or if it is, the quality of the drivers suffer due to Code Rot. Not releasing the binaries with support for the older chipsets defeats the purpose of the Unified Driver in the first place, while releasing the drivers either means the release schedules get delayed or the drivers ship prematurely with less testing on older hardware devices. The end resulting is that the quality of the device drivers for older hardware gets worse over time as the testing and verification process becomes more and more time consuming.

SciTech SNAP, the Total Driver Solution

SciTech Software has used its years of experience in creating drivers that operate with hundreds of graphics chips to develop a new device driver technology called the SciTech System Neutral Access Protocol, or SciTech SNAP. The SciTech SNAP device driver technology represents several significant advancements in device driver development. These advancements have far reaching implications for the future of driver and operating system development.

Architectural Overview

Overview of a Device Driver

In examining the SciTech SNAP architecture, it is helpful to first see what it has in common with the traditional device architecture. Both the traditional and SciTech SNAP device drivers have two main parts:

1. The Operating System Device Class: Every operating system defines a unique device driver interface for each class of hardware device it supports (for example, graphics cards, mice, keyboards, etc.). Since each operating system defines the class in a slightly different way, this driver component is operating system dependent.

2. The Device Dependant Element: This is the device driver component that interfaces directly with a specific piece of hardware. Since the actual hardware device does not change when using different operating systems, this core code is essentially the same between operating systems.

Using traditional device driver development techniques, the two driver components are combined to form a single, logical device driver. Therein lies one of the central causes of the Driver Crisis: commingling operating system and device specific code. In contrast, SciTech SNAP separates these standard driver functions into two, independent components.

Driver Partitioning

SciTech SNAP takes the traditional driver model several steps further as shown in Figure 3:

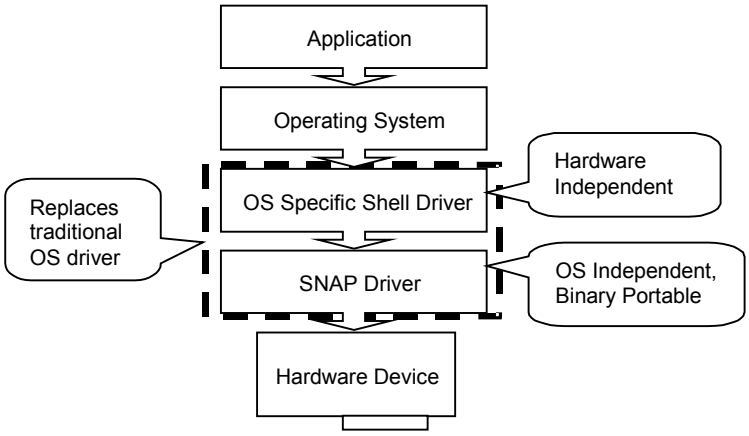


Fig. 3: SciTech SNAP Driver Architecture

Internally SciTech SNAP partitions a traditional device driver into two parts:

- The operating system specific portion: The top-level component, or the operating system 'Shell Driver', accepts native operating system requests. The operating system shell driver contains no hardware specific code and contains only the code necessary to link the operating system with the SciTech SNAP driver. This shell driver operates with any device with a SciTech SNAP driver.

- The device specific portion: The bottom-level component, the SciTech SNAP driver accepts generic requests from the shell driver and directly programs the specific hardware device. Since a SciTech SNAP driver contains code only the code necessary to directly program the hardware device and no operating system specific code, it will work with any operating system shell driver.

This modular design means that any optimizations or bug fixes implemented in the operating shell driver is effective for *all* hardware devices, not just a single device. Likewise, any optimizations or bug fixes implemented in the SciTech SNAP driver are effective on all supported operating systems. Hence, this design allows the maximum amount of code sharing among different hardware devices and operating systems.

Binary Portable Drivers

In traditional development, device drivers for different operating systems rarely share the same code. With the above changes to the traditional device driver model, it is possible to build device drivers that share the same source code across multiple operating systems. SciTech SNAP takes this concept one step further, making the code binary compatible between operating system using a Binary Portable Driver.

A Binary Portable Driver is an operating system independent, Dynamically Linkable Library ('DLL'). The DLL concept is a standard feature on all modern operating systems, however the mechanisms used to create a DLL and the format of a DLL are usually specific to each operating system. Hence, a DLL built for the Windows platform cannot be used under OS/2, Linux or the BeOS and DLLs created for those environments cannot be used under Windows. In contrast, the exact same SciTech SNAP Binary Portable Driver can be loaded and utilized on any operating system within a single processor family (and they can be recompiled for use on different processors).

The use of Binary Portable drivers allows sharing of the common device driver code across multiple operating systems. For example, Figure 4 illustrates how SciTech SNAP can be used to support seven different operating systems:

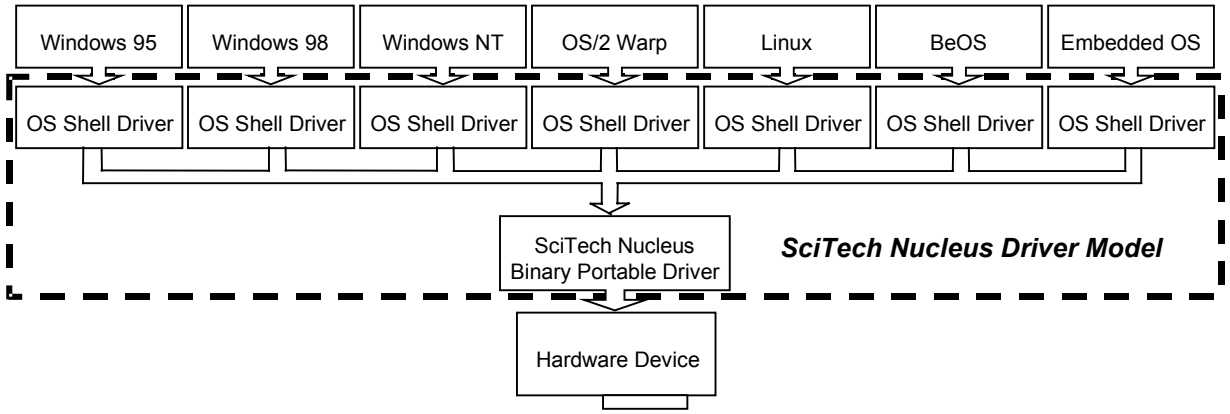


Fig. 4: SciTech SNAP Driver Architecture

One of the key benefits of this partitioned driver model is that the device specific components only need to be developed once for a particular hardware device, which reduces the overall development time for a particular hardware device.

Processor Independent Source Code

SciTech SNAP device drivers are developed using the highly portable C programming language. This allows the source code for the SciTech SNAP drivers to be portable to different processor architectures. Moving SciTech SNAP to a new processor platform is a matter of porting the tools and recompiling all the device drivers. Thus, the effort involved in supporting new emerging processor platforms such as Intel Merced is drastically reduced, both because the driver development time is eliminated and the testing and certification time is reduced.

In addition to source code portability, one important feature of SciTech SNAP is that drivers for all operating systems can be built and tested using a single development environment. Since a single compiler can be used, device drivers can utilize inline assembler, specialized processor features and other compiler-specific optimizations without having to translate those special functions to other compilers.

BIOS Independent Drivers

SciTech SNAP device drivers are designed to be completely independent of the underlying legacy BIOS installed on PC hardware devices. This dramatically simplifies porting SciTech SNAP to other operating systems and architectures. It also means that the SciTech SNAP drivers will run on any hardware device that uses the same hardware components, regardless of the board vendor who built the hardware device.

Key Benefits

Maximum Performance

Real world benchmarks have proven that SciTech SNAP drivers perform as good or better than drivers hard coded to a particular operating system and hardware device. This is due to the fact that any potential performance overheads have been eliminated in the design of the SciTech SNAP driver interface. The SciTech SNAP driver interface is designed as a pure hardware abstraction layer, where there is a one-to-one correspondence between SciTech SNAP driver functions and hardware features.

Also, since SciTech SNAP drivers share code, generic optimizations that are implemented in the operating system shell driver can be used by all devices and optimizations to a SciTech SNAP driver are accessible by all supported operating systems. Since optimizations only need to be developed in one place, the optimizations can be used by the widest variety of device/operating system combinations and there are more opportunities for spending additional time on further optimizations.

Rapid Device Driver Development and Reduced Costs

SciTech SNAP minimizes the costs of maintaining updated device drivers for multiple hardware devices. The following illustration compares the costs of developing device drivers to support seven different operating systems for a single hardware device. For the purposes of this comparison, the following assumptions have been made:

- Annual cost per engineer: $100,000
- Time to develop a traditional driver for a single operating system: Two man-months
- Time to develop a SciTech SNAP shell driver for a single operating system: One man-month
- Time to develop a SciTech SNAP device driver for a single device: One man-month

Using traditional development methods, seven complete device drivers would need to be developed: one for each operating system. Hence, the total development time for supporting this single device in all seven operating systems can be computed as follows:

1 device x 7 operating systems x 2 months = 14 man-months

With SciTech SNAP however, it is only necessary to develop a single, hardware specific SciTech SNAP driver and one shell driver for each operating system. Hence, the total development time for the SciTech SNAP based device drivers can be computed as follows:

1 device x 1 month + 7 operating systems x 1 month = 8 man-months

Because the hardware specific code only needs to be developed once, SciTech SNAP cuts the development time for a single hardware device nearly in half. When multiple devices are considered, the cost savings are even more significant. Consider that there are over 250 different models of graphics hardware devices currently available and in use. Using traditional methods, developing device drivers to support all 250 hardware devices on all seven operating systems would take:

250 devices x 7 operating systems x 2 months = 3,500 man-months

= 291 man-years ($29.1 million)

Now compare this to development using SciTech SNAP. The total time to develop device drivers supporting all 250 hardware devices on all seven operating systems would be:

250 devices x 1 month + 7 operating systems x 1 month = 257 man-months

= 21.4 man-years ($2.14 million)

This amounts to 1/14th the development time and cost required by traditional methods. These scenarios do not take into account support for future operating systems, development of device drivers for other processor platforms, or the time and costs involved in testing and verifying that the drivers are correct. Once those additional costs are considered, the SciTech SNAP development method is even more attractive.

Rapid Device Support for New Operating Systems

As operating systems technologies advance, new device drivers must be developed to support the new operating systems. Using traditional methods, the total development time to develop device drivers for a new operating system would be:

250 devices x 1 operating system x 2 months = 500 man-months

= 41.7 man-years ($4.2 million)

With SciTech SNAP binary portable device drivers, adding device support for a new operating system is simply a matter of developing a shell driver for the new operating system, instantly leveraging existing SciTech SNAP device support libraries. The total development time in this case would be:

1 operating system x 1 month = 1 man-month

= 0.08 man-years ($8,300)

This amounts to a reduction in time and cost in by a factor of 500. Hence, SciTech SNAP can be used to slash the development time for device drivers for new operating systems by a significant margin.

Automated Driver Certification

Because of the costs involved, many hardware vendors only focus on fully testing their latest hardware running on the most popular operating systems. To reduce the complexity of this problem, SciTech has developed software quality assurance tools that certify that a SciTech SNAP driver is generating the correct output. Rather than certifying the operating system shell driver and SciTech SNAP driver as a single unit, each component is certified separately.

To certify that the SciTech SNAP drivers are generating the correct output for a given set of input conditions, a sequence of complex stress tests are passed through both the hardware device driver and a software reference rasterizer. The reference rasterizer can simulate every SciTech SNAP function in software. The hardware and software generated output are then compared to ensure that the hardware device driver is generating the correct output. This process is illustrated in Figure 5:

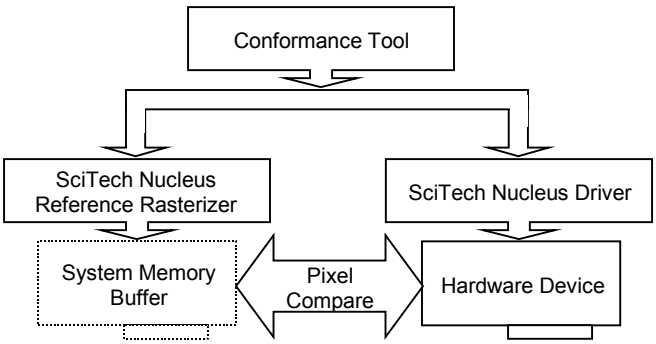


Fig. 5: Certifying the SciTech SNAP Driver

Once the low-level SciTech SNAP hardware device driver is certified, the next step is to certify that the operating system specific shell drivers work correctly when running on top of a certified SciTech SNAP hardware device driver. This is accomplished by running the operating system specific conformance tests (such as the Windows Hardware Quality Labs tests) on the SciTech SNAP software reference rasterizer, configured to simulate a full-featured hardware device, as depicted in Figure 6:

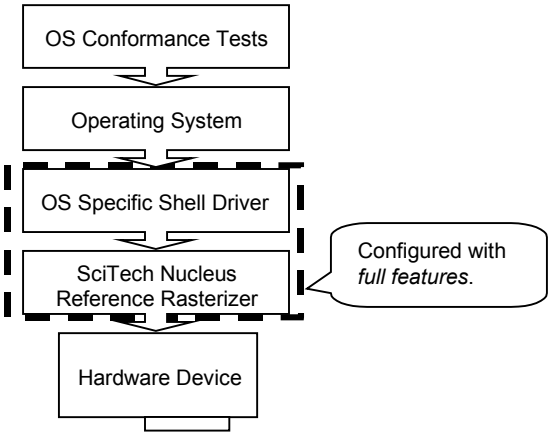


Fig. 6: Certifying the SciTech SNAP Operating System Shell Driver

This enables the testing of all code paths in the operating system shell driver without the need to test the operating system shell driver with every possible hardware device. To ensure that the operating system shell driver works properly with actual hardware devices, full compliance tests are performed on the shell driver using a random sample of representative hardware devices.

Rapid Quality Assurance and Testing

Consider how much time and effort can be saved by certifying the operating system shell drivers and hardware drivers separately. It generally takes at least a full day to run a graphics hardware device through the Microsoft Windows Hardware Quality Labs conformance tests. Running the suite of SciTech SNAP conformance tests on a single hardware device takes approximately 30-60 minutes. Considering there are 250 different SciTech SNAP drivers (there are actually many more variations depending on memory type and board vendor variations), to certify all devices for a single operating system it would take:

$$250 \text{ device} \times 1 \text{ hour} + 1 \text{ day} = 258 \text{ hours}$$

$$= 32.3 \text{ man-days}$$

To run the same 250 devices through the Microsoft Windows Hardware Quality Labs conformance tests, it would take:

$$250 \text{ device} \times 1 \text{ day} = 250 \text{ man-days}$$

Using the split certification process, SciTech can test and certify additional drivers in approximately $1/8^{th}$ the time it takes using traditional operating system conformance test suites. Since SciTech SNAP drivers are binary portable between operating systems, once a SciTech SNAP driver is fully tested and certified, it does not need to be re-tested and certified unless the driver is subsequently modified. Therefore, when a new revision of the operating system arrives, re-certifying the SciTech SNAP drivers only takes a single day, whereas re-certifying the drivers for the new operating system the traditional way would take the full 250 man days.

Universal Driver Libraries

SciTech has developed a system that allows the binary device drivers to be collected into a single binary file which can be linked into programs to provide SciTech SNAP support directly in applications or utilities. This enables the creation of universal driver products, such as SciTech Display Doctor, that support every device within a given device class (i.e. all graphics controllers). Intelligent partitioning between device specific code and generic SciTech SNAP framework code means only the executable code necessary to drive a particular device is ever loaded in memory.

The SciTech SNAP universal driver library mechanism is also designed to be fully plug-and-play, even on legacy non-plug-and-play devices. When a SciTech SNAP driver library is first loaded, it runs a set of detection routines to automatically determine the hardware installed on the users system. The driver library then selects the appropriate hardware device driver from the library, and initializes it for use.

Flexible Licensing

SciTech has developed powerful new techniques to manage licensing of device drivers to multiple parties. Traditionally, licensing device drivers to third parties required custom development, or a special re-compile of the driver specifically for a particular third party. SciTech SNAP eliminates that with the ability to provide encrypted, binary license certificates to software or hardware vendors which unlock specific licensed feature sets. All licensed parties ship the exact same binary device driver library, and the only thing that differs is the binary license certificate. Licensing SciTech SNAP device driver technology to a new third party requires no additional quality assurance or testing cycles.

Beneficiaries

End Users

Personal Computer end users benefit from the SciTech SNAP device drivers because they will have stable, usable drivers regardless of the hardware or operating system they choose.

Information Systems Managers

Information system managers will be able to have a single driver interface across their entire heterogeneous base of installed hardware, thus reducing support costs and improving the reliability of their entire network.

Software Developers

Software developers benefit because they are no longer tied to developing for a particular operating system because of existing device support options, or lack thereof.

Hardware Vendors

Hardware vendors can continue to focus on supporting their latest products on the most popular operating systems and SciTech SNAP can provide driver coverage for legacy and Tier 2 operating systems.

Processor Vendors

Processor vendors benefit because they can create new processor designs that may break compatibly with existing legacy CPUs without having to worry about losing device driver support.

Operating System Vendors

Operating system vendors benefit because they can change their driver architecture without fear of losing existing device support. Also, this solution will give new or less popular operating systems a better chance to succeed on their own merits rather than fail because of a lack of available device support options.