# The Need For Vendor Source Code at NAS

**Steve Acheson, Bruce Blaylock, David Brock, Nick Cardo, Bob Ciotti, Alan Poston, and Parkson Wong**

NAS Systems Division
NASA Ames Research Center
Mail Stop 258-5
Moffett Field, CA 94035-1000

NAS-96-022
August 1996

## Abstract

The Numerical Aerospace Simulation (NAS) Systems Division has a long standing practice of maintaining buildable source code for installed hardware. There are two reasons for this: NAS' designated pathfinding role, and the need to maintain a smoothly running operational capacity given the widely diversified nature of the vendor installations. NAS has a need to maintain support capabilities when vendors are not able; diagnose and remedy hardware or software problems where applicable; and to support ongoing system software development activities whether or not the relevant vendors feel support is justified. This note provides an informal history of these activities at NAS, and brings together the general principles that drive the requirement that systems integrated into the NAS environment run binaries built from source code, *onsite*.

This is a revision of Russell Carter's June 1994 report: RND-94-007.



---

## 1.0 Introduction

The Numerical Aerospace Simulation (NAS) Systems Division Program at Ames Research Center is a large-scale integrated computer center featuring vector supercomputers, highly parallel systems, high-performance graphics workstations, and appropriately large storage servers. These systems run various flavors of Unix, and are interconnected via one or more high-performance LANs. NAS supports over fifteen major systems and 450 systems overall. Each subsystem is characterized by production use of multiple-vendor solutions, which must coexist peacefully in an open environment. The NAS is responsible for the continued smooth functioning of this very complex environment.

A major goal of the NAS Division is pathfinding in the area of high-performance system integration. Achieving this goal requires the flexibility to install the highest- performing systems, regardless of vendor. Thus, the NAS pathfinding goal is the source of much of the diversity, and accompanying challenges, inherent in the NAS operational environment.

It is critical for the tools and techniques that are used to maintain the effectiveness of the NAS environment that the operating system and layered software environments be run from source code built onsite. Each vendor (which includes Wellfleet, Cray Research, Silicon Graphics, Sun, Convex, BSDI, StorageTek, Ultra, IBM, and Intel Supercomputing Systems Division) supplies system source code, and in most cases the running operating system is built onsite. NAS personnel frequently respond to questions about the origin and necessity of the buildable source code requirement, particularly from vendors unfamiliar with or new to the NAS environment. This document provides a sampling of our experiences, pros and cons, with operating-system issues that pertain to the NAS requirement that "we build what we run," and thereby should provide insight into the motivation behind our requirement that buildable operating system source code must accompany any system installed at our site.

The structure of this document is as follows: First, there is a description of the real-world vendor environment in which NAS operates. Next follow the three categories of use that we have for system source code:

- Documentation
- Software Development
- Bug identification, patching and resolution of vendor design flaws

Then follows a discussion of vendor data rights, and some of our procedures that we use to protect them. Finally, there is a summary. It is important to note that no proprietary code is ever transferred to unlicensed platforms at NAS--even for projects that require porting.

## 2.0 Practical Considerations

Many vendors sell a product and offer little or no support for integration into complex multi-vendor sites like NAS. The product may work well under some circumstances, but due to the complexity of software

and configuration issues, it may not work as intended for NAS.

Reasons for the lack of support are varied but are usually one or more of the following:

- Hard problems!
  (Complex systems have complex problems!)
- Management issues:
  Magnitude of support task underestimated
  Poor internal communications
  Support staff not competent enough
  Insufficient engineering resources
- Firm having financial difficulties
- Conflicts with marketing/sales goals

In the last item, a firm may be overly sales-oriented, particularly when a new product is introduced. The vendor may not give sufficient priority to support, or a vendor may choose not to recognize an issue as a bug due to sales/marketing ramifications.

### 3.0 Experiences at NAS

The experiences fall roughly into the following three different categories listed below. In each category, We have included several typical examples. This is by no means comprehensive: the list maintained for these examples has many more entries.

### 3.1 Documentation

The provided documentation on the types of systems installed at NAS is often out of date with the installed software, or even just wrong. Some documentation may be out of date due to running early releases or beta software. Others are out of date even though packaged in an official release. Fundamental tasks, such as adding a device driver, say, for a HiPPI-attached RAID array, are much more difficult without access to source, because the definitive documentation on what kernel calls (for instance) are available is the source code itself.

- On Cray Research Systems, source code was used to fix incorrect documentation. The actual system-scheduling parameters turned out to be entirely different from claims in the manual. Similarly, incomplete documentation was filled in by using source code. Exactly how the scheduler determines an "interactive" process was discovered this way.

- NAS' Portable Batch System's (PBS) resource monitor needs to be able to extract information from the system kernel concerning global resource availability and utilization, and about per-session resource utilization. PBS's machine-oriented miniserver module has similar needs to control session limits. Generally, documentation of the necessary kernel

interfaces is sketchy enough that reference to code is required to find out how the interfaces work. In the case of the Intel Paragon, the documentation is completely lacking. Fortunately, persons working on the Portable Batch System can refer to the source code to learn what is needed.

- Wellfleet source code provided the only accurate documentation on the workings of the supplied routers. This knowledge was necessary to develop effective network monitors.

- Rebuilding everything from vendor-supplied source guarantees that you have the correct source. Problems with support distributions were uncovered by building from source onsite.

- Wellfleet routers experienced system crashes when confronted with multi-cast traffic. In the absence of source code this bug has been, and remains, unresolved as of 8/96.

- The SGI IRIX 6.2 operating system has a deficient version of "gated." The SGI version does not support router discovery, and loses the default route when multiple interfaces are present on the machine. Porting a more robust version of gated to IRIX 6.2 requires access to internal routing structures present in the source.

- The network appliance toasters were tuned by the vendor to ignore duplicate packet requests. The vendor assumed the network would never be broken or otherwise fail. The implementation error was inferred through discussion with the vendor. Lack of source prevented on-site diagnosis and correction.

- While testing the CRI version of NQS, CRI inserted a "return immediate" function as a placeholder. This placeholder function was not removed prior to system release. As a result, job data structures were not properly updated during the normal execution of NQS.

- A CRI exception-handling failure resulted in a "Kill -SIGKILL" signal being sent to process 1. This resulted in a system failure and a forced re-boot.

- The UNICOS operating system computes 3 load averages. These averages could not be properly interpreted without the source code.

- Fortran writes under AIX which exceeded 256 Mbytes failed. The fault was traced to an unadvertised limit on single write sizes.

- The IBM SP-2 POE program failed to correctly propagate signals from the front nodes to the compute nodes. Examination of the source code disclosed the error and allowed its on-site resolution.

- The delivered IBM SP-2 NFS implementation buffers NFS writes (a protocol violation). When a parallel job is killed because it exceeds its time allocation, the process is blocked inside the kernel until the buffers are flushed. Given the amount of data that may be buffered, and the small I/O bandwidth to the NFS server, this could take up to an hour. During this time, the unkillable process owns the switch, so any subsequent jobs started on this node will fail. Because this "feature" is tightly integrated with the IBM virtual memory design, development efforts have been directed to making PIOFS functional instead of attempting to "fix" NFS.

- The Intel Paragon virtual memory system had numerous design and implementation flaws. Examination of the source provided avenues for testing, verification of problems, and workarounds. System fixes were provided to Intel for incorporation into the release source.

- Systems can not be verified as secure in the absence of source. Delivered binaries are always suspect. Known or discovered security weaknesses require source for resolution.

- The addition of batch scheduling software is vastly improved with system source. Scheduling software requires an understanding of how to interact with the native system scheduler. The source code is the best documentation for this activity.

- The ULTRAnet provided FTP client crashed if the user's shell name was a multiple of 4 characters. Without source code, this bug could not have been corrected.

### 3.2 Bug identification, patching, and resolution of vendor design flaws

There are numerous examples of site-critical bugs fixed using source code before the vendor could (or would) respond to a bug report.

- Incorrect padding of tape blocks in a tape driver ruined backups. Botched system scheduling parameters required fixing.

- The Proteon Ring evaluation required driver debugging using UNICOS source code.

- Early SGI releases included a bug that prevented telnet connections to the Cray-2. Source code was required to fix the bug.

- Lack of Informix source prevented an on-site solution to resolution of commit function bugs. In the absence of vendor cooperation, a workaround replacement was instituted.

- The Morris Virus sendmail bug was fixed using source code for all systems except SUN, within two days. Without source code for SUN sendmail, the fix for SUN systems took six

months.

- Resolution of vendor-vendor incompatibilities is facilitated through the use of diagnostic traps implemented in the operating systems, as was the case with Cray and NSC.

- iPSC/860 remote host software was ported to SGI systems. Without this added functionality, the iPSC/860 could not have supported the NAS workload. NAS personnel successfully carried out the port.

- iPSC/860 cube limit was increased to more than ten. The artificial cube limit severely impacted the ability of the iPSC/860 to support multidisciplinary codes.

- Routing protocol bugs in the Wellfleet routers were fixed in order to support the Routing Information Protocol package. Workarounds for other problems were implemented in this manner as well.

- After a catastrophic crash of the Amdahl hosted mass storage system, NAS personnel diagnosed an Amdahl-generated fatal design error. Access to source code allowed the implementation of file system repair tools that Amdahl had never bothered to produce, with a partial recovery (85%) of the lost data. Tools to rebuild the file system and rescue a majority of the files and file names were created.

- Over the last three years, NAS Systems developers have documented and fixed at least 12 serious CONVEX kernel bugs thanks to access to the source code. These bug fixes were later incorporated into the Convex OS. They were in the areas of file system code; networking code; and specific device drivers like the SCSI driver, the HiPPI driver, and the TLI (Block MUX channel) driver. Lack of source would have meant perhaps weeks of waiting for a patch from Convex.

- Access to the STK source code has allowed NAS to fix at least two problems with the STK sun server software (ACSLS). These problems and fixes were submitted to STK and later incorporated into the ACSLS product.

- Numerous modifications have been made to the CRI-supplied UNICOS system. They include:

    1. Online backup of DMF databases. This functionality is now incorporated in release distributions from CRI.
    2. Make all files dual-resident. Functionality now incorporated in release distributions from CRI.
    3. Enhance "date" to allow for specification of +/- days.
    4. Security modifications.

5. Queue identifiers for accounting.
6. Disk Highwater mark.
7. Improved mv/cp command which adjusts block size depending on device characteristics.

- In order to improve batch job scheduling and workload execution, the NAS Systems Division developed the Session Reservable File System (SRFS). By using SRFS system file space is guaranteed to a job during its entire execution. SRFS facilitates the allocation of file space on system devices including the CRI solid- state disk device. This major enhancement to UNIX was only possible because of access to system source.

- Intel Paragon OS was instrumented to observe the system message passing profile. This data was returned to Intel and was used in the re-design of the message passing subsystem.

- Compliance with NASA and NAS security policies require modification to "*login*," "*password*" and "*rpc.mountd*."

### 3.3 Software Development

The nature of the systems installed at NAS requires the ability to add new features and functionality that, frequently, the vendor will not agree to do for us in a timely fashion, if at all. NAS acquires many technologies before they are "proven" or "mature," and often making such a technology work in production implies that we perform development support on the technology. Without source code, the vast majority of this work would not be possible, and NAS would be subjected to the sometimes-unsuitable agendas of vendors for support and further development. The following examples illustrate the varieties of NAS development needs that are met using source code.

- The Network Queuing System (NQS) was developed using source code for targeted systems.

- The Portable Batch System's (PBS) resource monitor needs to be able to extract information from the system kernel concerning global-resource availability and utilization, and about per-session resource utilization. PBS's machine-oriented miniserver module has similar needs to control session limits. Generally, documentation of the necessary kernel interfaces is sketchy enough that reference to code is required to find out how they work.

- Amdahl UTS source code enabled the implementation of NAStore and port TCP/IP when it wasn't available from Amdahl.

- ConvexOS source code allowed us to port NAStore and enhance the OS. Joint

development with Convex resulted in an improvement in file system performance from 20 MB/s to 150 MB/s.

- Wellfleet router source code allowed the implementation of router discovery. This is now a distributed product by Wellfleet.

- Source for UNICOS and Cray NQS allowed the implementation of the Session Reservable File System (SRFS), an enhanced resource management function. Disk quotas and an Ultra driver (mandatory in the NAS environment) were also implemented.

- UNICOS source code was need to develop tools to modify the kernel mount table and develop the top and mu (memory usage) commands. Minor hooks were added for new user-level services such as real-time and cpu-time gid limits.

- Non-intrusive, low-level data collection requires modification to the operating system. Two successful projects that required access to system source code are the iPSC/860 Concurrent File System monitoring and a message sizes monitoring project.

- The Map library on the iPSC/860 (which provides support for multidisciplinary applications) is a modified version of an Intel message-passing library, successfully modified with no performance degradation. It allows application programmers access enhanced message-passing functionality.

- For the p2d2 project and MPK project, source is required on targeted platforms for the libraries that support parallel processing (For example: message passing, collective operations, process creation, termination, locks, events, and critical sections).

- Access to the Amdahl UTS source code allowed NAS to embark on the native file system HSM (Hierarchical Storage Manager), it also allowed us to create one of the very first RAID.

- Access to the Convex source code allowed NAS to port the native file system HSM to the Convex platform.

- NAS (in conjunction with Convex staff) completely redesigned the Convex RAID file system to be much more like the original developed for the Amdahl. The performance improvement for disk transfers was 18MB/sec to 200+ MB/sec on the same hardware for reads, and from 13MB/sec to 110 MB/sec for writes.

## 4.0 Vendor Data Rights

NAS recognizes the sensitivity of proprietary source code and works with software vendors to establish

a protection scheme that meets their requirements. Vendors have expressed widely different views on protection policies[1]. Rather than enforce a blanket source code security methodology, NAS has established a simple baseline protection policy which is then augmented with vendor-specific requirements.

The baseline protection scheme recognizes that the primary method for controlling disclosure of sensitive material is to restrict the set of people with access to it. NAS maintains lists of active source-code users, periodically culled as requirements change, and allows access only to listed users. Before being added to the lists, users are typically required to meet some vendor pre-requisite, such as signing a non-disclosure agreement. All NASA employees, both civil servants and contractors, are legally bound to support proprietary data rights of vendors.

NAS maintains a physically secure, logged-access source code laboratory to facilitate implementation of vendor-specific requirements. The laboratory is equipped with build computers, removable disk drives, and key-secured cabinets. The build machines can be readily physically disconnected from the NAS network, if required. In any case, login access is restricted to listed users of the source code currently installed on each system. If network access is permitted, filtering programs prohibit connections from any but an approved list of IP addresses (typically workstations belonging to users approved for access). If necessary, network connections are encrypted to preclude "sniffing." All access to the source code laboratory, whether physical or via the network, is logged. Non-listed personnel are permitted access only under direct supervision. In short, the source code laboratory provides the capability to enforce almost any access methodology negotiated with a vendor.

In any case where source code is transmitted over the Internet (which occurs only if the vendor chooses the Internet as a delivery medium), it is encrypted using at least a 56-bit key. This is equivalent to using the U.S. Government DES algorithm or better.

## 5.0 Summary

In an ideal world, hardware works flawlessly, software works flawlessly, kernel interfaces are documented completely and flawlessly, documentation is in sync with binary distributions, and vendors are strongly motivated to meet NAS mission critical needs in a timely manner. None of these hold in practice. Hence the requirement that NAS run system software built onsite from source code. NAS access to buildable source code has demonstrable benefits for the vendor, and historically has been a basis for continued progress and improvement of the supplied systems, which has lead to increased markets and revenue for vendors.

## 6.0 Acknowledgments

This note represents the distillation of the collective wisdom of the NAS Systems Division personnel, including supporting contractors. Without the help of the many persons who have worked on the complex systems problems historically encountered at the NAS, this information would not have been

documented. Indeed, much of the information in this note has not previously been formally documented from the perspective of the source code requirements. Contributors of comments, examples and/or text include (in no particular order) include: John Lekashman, Dave Tweten, Bill Kramer, Toby Harness, Jonathan Hahn, Bill Nitzberg, Sam Fineberg, Bernard Traversat, David McNab, Eric Townsend, Keith Thompson, Alfred Nothaft, Doreen Cheng, Jeff Becker, David Barkai and David Henkel-Wallace.

---

1 For example, one vendor is content to keep the source code in a UNIX permission-protected directory hierarchy on a busy multi-user system. Another requires that the source be kept only on machines physically isolated from NAS' network, and even then that the disks storing the code be physically removed from the build machine when not in use. A third vendor asks us to avoid mentioning that we even have their source code.