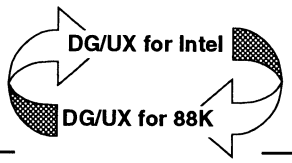
Data General
Bringing Common Sense to Computing

# DG/UX® Quick Reference

September 1995




## Porting Applications to DG/UX for Intel

Contents



Since 1989, Data General has provided solutions for commercial customers who want to use cost-effective, open computing solutions instead of expensive, proprietary solutions.

With its AViiON® line of Symmetric Multiprocessor (SMP) servers and its CLARiiON® line of disk and tape arrays, Data General offers its customers a range of high performance computing solutions. The AViiON SMP servers are currently available in configurations with one, two, four, eight, twelve, sixteen, or thirty-two processors. The CLARiiON disk array systems, which use RAID technology, provide high storage capacities as well as high availability.

The other key part of this common sense computing solution is the DG/UX® operating system. This mature and highly regarded implementation of the UNIX® SVR4 operating system provides full support for the SMP hardware used in AViiON servers. One of the strengths of the DG/UX operating system is its conformance to important programming standards, such as those from POSIX and X/Open.

Prior to the availability of DG/UX System release 4.10, AViiON servers used only the Motorola 88K family of CPUs (the 88100 and 88110 CPUs). To provide its customers with more computing options, the AViiON product line now includes servers that use Intel Pentium CPUs. For several years, enterprises have used Intel-based workstations in conjunction with larger servers. Now, the performance capabilities of the new Pentium CPUs bring with it the opportunity to use these commodity CPUs in enterprise servers.

To coincide with the availability of the new AViiON for Intel servers, the DG/UX operating system (starting with release 4.10) supports AViiON 88K servers *and* AViiON for Intel servers. These versions of the DG/UX operating system offer software development environments that are tailored for the two AViiON platforms.



012-004709-00

## What's in This Quick Reference

This Quick Reference provides high-level information about the process of porting applications to DG/UX for Intel from other platforms. We assume that you're an experienced software developer or porting engineer and that you're familiar with the software development environment of the DG/UX operating system. The Quick Reference:

- Introduces the porting-related features and benefits of DG/UX System release 4.10.
- Highlights the factors that influence the level of portability that an application can have.
- Identifies the architectural differences between the Intel and 88K architectures that may be visible to applications.
- Reviews different factors that can affect the porting processes of compiling, linking, and debugging.
- Provides tips on how to create applications that are independent of the architecture on which they will run.

Although the Quick Reference focuses on moving applications from the DG/UX for 88K operating system to the DG/UX for Intel operating system, it also outlines issues relating to moving applications to DG/UX for Intel from the UnixWare and other versions of the UNIX operating system.

## What's New in DG/UX System Release 4.10?

Because of Data General's commitment to providing an operating system that conforms to industry programming standards, the DG/UX operating system has gained a reputation of being an easy platform to which to port applications. With the availability of DG/UX System release 4.10 and its support for Intel platforms, the DG/UX operating system (along with the optional DG/UX Software Development Kits for the Intel and 88K servers) becomes an even more attractive environment in which to create and run portable applications.

Although the most significant enhancement in DG/UX System release 4.10 is its ability to run on both AViiON 88K *and* AViiON for Intel servers, DG/UX System release 4.10 brings with it many enhancements.

**FYI—Related Documentation**

Because this is a Quick Reference, it doesn't provide the level of detail that you might need to address a specific porting question. For more detailed information, you should refer to *Porting and Developing Applications on the DG/UX System* (069-701059). This manual provides information and techniques to help you develop DG/UX applications and port them among platforms. You can find a more complete list of standards-related documentation at the end of this Quick Reference (page 17).

In addition, the Intel and 88K versions of the DG/UX operating system share the same base of source code, which helps ensure compatibility between the two versions of the operating system. Here are some other porting-related highlights of the general and Intel-specific enhancements in DG/UX System release 4.10.

### General Enhancements

Some general enhancements in DG/UX System release 4.10 are:

- Application compatibility from the earlier 3.10 release of the DG/UX operating system—at the binary, object, and source level for 88K; at the source level for Intel.
- The optional Software Development Kit (SDK) that provides the necessary components for creating and debugging software.
- A reorganized library structure that matches more closely the structure of the libraries as specified in the Intel Application Binary Interface (ABI) and used by other UNIX implementations. This reorganization is transparent to 88K applications.
- For TCP/IP, support for Internet Protocol (IP) multicast configurations, for the Dynamic Host Configuration Protocol (DHCP), for Point-to-Point Protocol (PPP); and IP broadcast forwarding.
- For SNMP, a user-extensible agent and enhanced **sysadm** package installation.
- Several new commands and command options, including GNU's **gzip**, **gunzip**, and **gzcat** commands, and a -o argument to the mount command for displaying ISO-9660 filenames.
- **imake**, a utility that enables you to create platform-independent makefiles.

### Intel-Specific Enhancements

From the point of view of someone porting applications to AViiON for Intel servers, DG/UX System release 4.10 provides important features and enhancements, including:

- Device drivers for AViiON for Intel devices, such as SCSI host adapters and Ethernet adapters.
- An optional DG/UX Application Capture Option (**aco**) package that allows certain UnixWare and SCO applications to run on DG/UX for Intel without being recompiled.

In addition to these enhancements, the DG/UX System release 4.10 complies with the X/Open XPG4 Portability Guide (XPG4). The R3.10 version of the DG/UX operating system complied with XPG3.

The operating system's release notice has more information about these and other features and enhancements.

## When Are Source-Code Changes Required?

If a DG/UX for 88K application has no architectural dependencies, uses no assembly language routines, and conforms to industry programming standards, the process of porting the application to DG/UX for Intel can be as straightforward as compiling and linking the application in the DG/UX for Intel Software Development Environment (SDE). However, the architectures of the Motorola 88K and Intel processors are different, so you cannot create a single binary program that will run on both platforms.

Whether you'll need to change an application's source code (port the application) to run on DG/UX for Intel is determined by the relationship among three factors:

- Type of operating system environment for which the application was written originally
- Type of processor architecture for which the application was written originally
- Degree to which the original application conforms to industry programming and portability standards (see the "FYI" inset at the bottom of this page)

### FYI—Compatibility and Standards

There are three kinds of compatibility: binary, object, and source.

*Binary* compatibility enables you to copy an application's binary program onto a system and execute the program without any compiling or linking—no porting effort is needed.

*Object* compatibility enables you to take objects from one system and relink them on another system.

*Source* compatibility enables you to compile and link the same source code in the software development environments of different operating system/hardware platforms.

Industry organizations have published standards that help ensure binary and source compatibility. The DG/UX for Intel operating system implements a subset of the Intel ABI supplement to the System V Application Binary Interface (gABI); the DG/UX for 88K version implements a subset of the Motorola 88K supplement to the gABI. The DG/UX operating system conforms to the following source-level standards: POSIX, FIPS 151, System V, BSD, and XPG4.

Table 1 provides some general guidelines that can help you determine whether you'll need to make changes to an application's source code before you move the application to DG/UX for Intel. The sections that follow discuss the guidelines in detail.

*Table 1 What to Look For When Porting an Application*

| Application Characteristics | Porting Considerations |
|---|---|
| **88K Applications** | |
| Uses 88K assembly language routines | If possible, rewrite the assembly language routines in C, then compile and link in the DG/UX for Intel environment. If assembly routines are required (for reasons of performance or to access some special hardware features), then rewrite the routines in Intel assembly language before compiling and linking. |
| Has dependencies on the 88K processor's byte ordering ("endianess") | Rewrite the routines that have the dependencies and convert existing data. Or if the application is to run on both platforms, rewrite the data-handling routines so that they are endian independent. |
| **UnixWare Applications** | |
| Adheres to the Intel ABI | May run without recompilation. |
| Depends on features of the DG/UX Application Capture Option | May run with no or with minor changes. |
| Uses features specific to UnixWare (features not supported by the DG/UX Application Capture Option) | Will require source code changes, compilation, and linking in either the DG/UX for Intel or UnixWare environments. |
| **Applications Based on Other UNIX Implementations (Such as SCO)** | |
| Built for other UNIX implementations | Many of these applications will compile, link, and run in the DG/UX for Intel environment. However, applications that use an operating system's proprietary features may require source code changes. |

**An 88K application can have one or more of these characteristics.**

# Moving DG/UX 88K Applications to DG/UX for Intel

Because this is a Quick Reference, it doesn't provide step-by-step instructions for the porting process—every application is different. However, we can review the general porting process and provide tips about what to look for when you're moving applications from DG/UX for 88K to DG/UX for Intel.

## *Before You Start—Qualifying an Application*

In many cases, the easiest way to determine what (if any) parts of your application's source code have to be changed to run on DG/UX for Intel is to "let the computer do the work"—by compiling the application's source code in the DG/UX for Intel Software Development Environment (SDE). If the application compiles and links without errors, you can continue and test the application.

If the compiler or linker generates errors, you can fix the errors, and then recompile and relink. During this iterative process, you might consider characterizing the fixes so that you can incorporate what you learn into your development group's programming standards or porting guide.

Another approach, before compiling, is to make changes to your application's source code based on your own porting experiences or the tips that are provided in this section. For example, you'll probably know beforehand whether your application has dependencies on the big-endianess of the 88K processor. In that case, you can take steps to make the application independent of the way that multi-byte data is stored—perhaps by using the DG/UX **xdr** library routines.

Figure 1 on page 7 highlights the porting process. The figure points to pages that provide more information about features or changes in DG/UX System release 4.10 that can affect the porting process. We've placed the items into the general categories of:

- Build environment differences (page 8), which include a reorganization of the libraries.
- Architectural (hardware) differences (page 8), which include byte-ordering (endianess), use of assembly language, and new device names.
- Updates to the development environment (page 13), which include updates to the compiler, to porting standards, and to the window manager.
- Updates to the runtime (windows) environment (page 13).

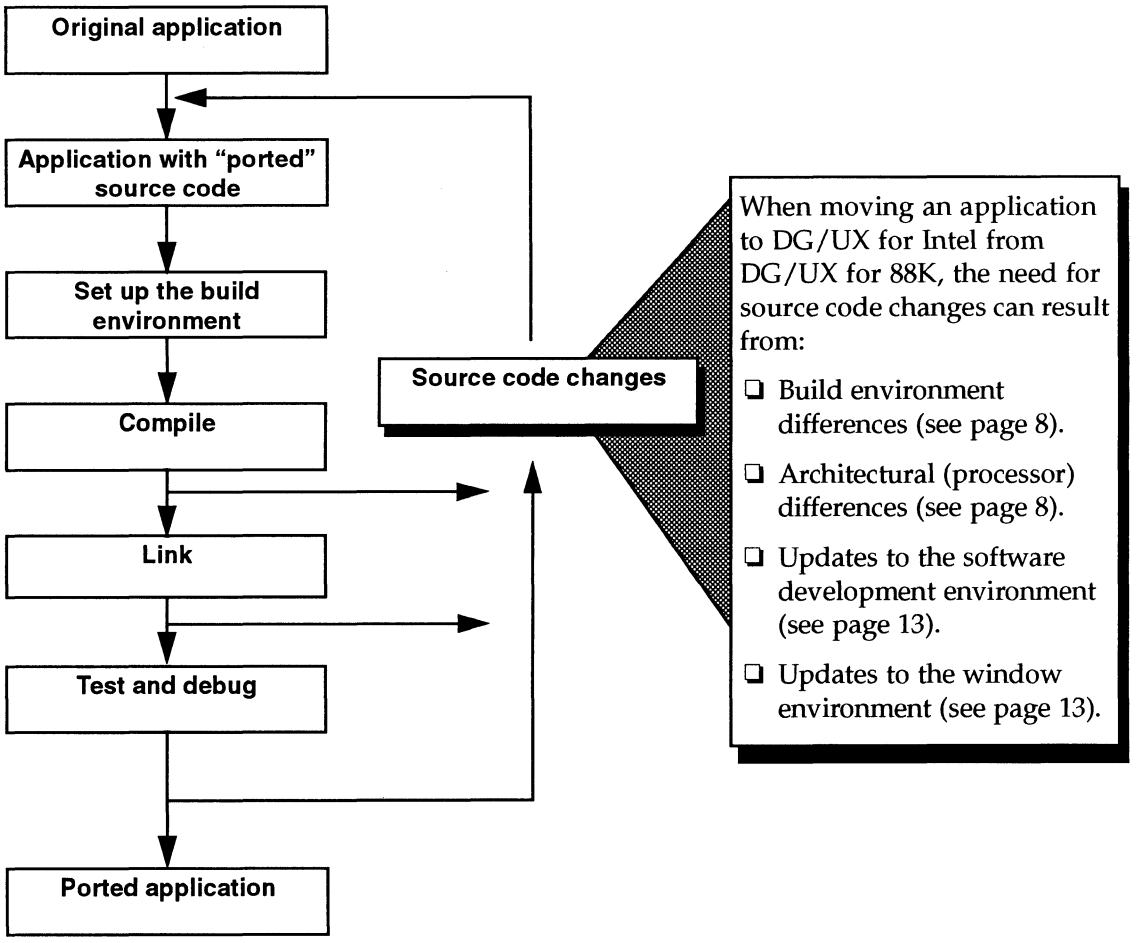



*Figure 1 The Porting Process*

## Build Environment Differences

In DG/UX System release 4.10, the library structure was reorganized to more closely match the structure that is specified in the Intel ABI and other UNIX implementations. The library **libdgc** has been removed and its functions were moved to **libc, libnsl, libsocket,** and **libresolv.**

Because of this reorganization, you may find that some 88K link-lines (in makefiles or shell scripts) generate linking errors for "undefined references" when used in the DG/UX for Intel environment. (The reorganization has no effect on 88K applications that are compiled and linked only in the 88K SDE.)

You can take the following steps to determine the minimum common libraries that you can use with DG/UX for Intel:

1. If you have the **-ldgc** argument in your link line, remove it.
2. Run the linker. If the linker returns no errors, the functions that you need are in **libc.** You can stop at this point and run the application.
3. If the linker produces unresolved symbol errors, add the following arguments, one at a time (in this order), and run the linker. Stop when there are no more errors and use the one, two, or three arguments in your link line.

   **-lnsl**
   **-lsocket**
   **-lresolv**

It's important to note that the same link lines that work correctly in the Intel SDE will work correctly in the 88K SDE. This provides you with the convenience of using the same link lines in both SDEs.

Chapter 7 of *Porting and Developing Applications on the DG/UX System* provides more details about how to create link lines that work with both SDEs.

## 88K and Intel—Architectural Differences

In most cases, a platform's DG/UX SDE hides the architectural differences of processors, so the porting process can be as simple as compiling an application's source code in the target system's development environment. However, if routines in your existing application's source code "see" some architectural differences, you'll have to change the routine to make it work properly with the other architecture.

From an architectural point of view, the things that can affect the porting process include:

- ❑ Ordering of multi-byte data—"endianess"
- ❑ Use of assembly language
- ❑ New device names
- ❑ The way that 88K and Intel systems format and store data on disks

## Ordering of Multi-Byte Data (Endianess)

The 88K and Intel processors handle and store multi-byte data, such as shorts, longs, bit fields, and floating point numbers, in fundamentally different ways.

- ❑ The Motorola 88K is a *big-endian* CPU; it stores multi-byte data by placing the data's *most* significant byte at the data's starting address in memory.
- ❑ The Intel Pentium is a *little-endian* CPU; it stores multi-byte data by placing the data's *least* significant byte at the data's starting address in memory.

If the application you are porting to DG/UX for Intel was written for a big-endian processor, you will need to evaluate the application to ensure that byte and bit accesses are not endian-dependent. This includes an examination of the structure of the data that the application uses (or generates) to ensure that the data does not depend on the byte ordering for proper interpretation.

If your application or its data is byte-order dependent, you'll need to take steps to either:

- ❑ Perform a one-time conversion of your data to change its format to that of the target platform. For example, if your application works with database data, you'll find that many database management systems, such as ORACLE, Sybase, Informix, and Progress, provide utilities that enable you to import and export data in different endian formats.
- ❑ Make your application completely "endian-neutral" by storing data in non-binary format or by using conversion routines. You could write your own conversion routines. However, a better choice might be to use the **xdr**(3) library routines, which are designed to work across different languages, operating systems, and hardware platforms, and translate different types of data into a machine-independent format.

The approach you take depends on the degree of portability you want your application to have. If you are doing a one-time port, a one-time conversion might be appropriate. However, if your application needs to run on different platforms, a change to the use of endian-independent data may be the best long-term approach.

Chapter 3 of *Porting and Developing Applications on the DG/UX System* describes big- and little-endian byte ordering and its effect on applications. The chapter also discusses the **xdr**(3) library routines.

### Endianess and Bit Fields

Bit fields, like integers, are multi-byte data types and have similar big-endian and little-endian differences. However, unlike integers and doubles, bit fields are not restricted to sizes that are multiples of a byte.

The compiler can pack several bit fields into a single word (in different order, depending on the endianess of the platform) and add padding between fields when they don't fit on word or byte boundaries. The exact format of the packing and padding is implementation-dependent and defined by the ABI for the target machine. Therefore, you should look for places in your application where bit-field data moves across hardware boundaries that use different endian protocols.

For example, instead of manipulating bits directly, you can use shift operations and masks, which are independent of a hardware platform's endianess. In addition, you remove byte-order dependencies by using byte streams (instead of multi-byte structure members).

### Endianess and Device Drivers

If your 88K application interacts with devices for which you have written a device driver, you may have to provide an Intel version of that driver.

As with the rest of your application, you should examine your device driver code to ensure that the code has no byte-ordering or bit-field dependencies. In particular, you should examine device drivers that you have written for use with SCSI adapters or SCSI devices. Like the 88K processor family, SCSI was designed to use the big-endian protocol.

If your device drivers have byte-order dependencies (if a driver uses multi-byte data) or if a driver manipulates bit fields directly, you will have to remove endian dependencies from the code in order to port the drivers to the little-endian Intel platform.

As mentioned above, you should also avoid writing bit fields to peripheral devices (across hardware boundaries). The resulting data will require a careful analysis to determine if it can be read on a different (endian) platform. You can eliminate bit-field dependencies by using bit-shift and masking operations.

With DG/UX, all user-written device drivers must be linked into the kernel and the machine rebooted before the device driver becomes active.

Chapter 7 of *Porting and Developing Applications on the DG/UX System* describes how to set the proper feature test macros for linking with DG/UX kernel libraries. *Programming in the DG/UX Kernel Environment* (093-701083) contains information about preparing device drivers for use with DG/UX platforms.

## Assembly Language

Assembly language routines are platform-dependent. If your 88K application uses Motorola assembly language routines, you'll have to rewrite the routines in C (to make them platform-independent) or in Intel assembler.

If you need to use assembly language routines, for reasons of performance or to work with a special device, you can simplify the job of porting by isolating the assembly language routines when you move the application among platforms.

The syntax of the DG/UX assembler for Intel (**as**) is compatible with UNIX SVR4 assemblers, such as the UnixWare assembler. Chapter 10 of *Porting and Developing Applications on the DG/UX System* provides details about the differences between the **as** assembler's syntax and the syntax used by Intel/Microsoft assemblers.

## New Device Names

The characteristics of the AViiON for Intel servers requires new boot commands for disks and tapes (see Table 2).

*Table 2 AViiON 88K and AViiON for Intel Boot Command Examples*

| AViiON 88K | AViiON for Intel |
|---|---|
| `b sd(ncsc(),0)root:dgux -3` | `sd(npsc(pci(),C),0)root:dgux -3` |
| `b sd(ncsc(),4)root:dgux -3` | `st(npsc(pci(),C),4)` |

A related change involves incorporating the names of the AViiON for Intel hardware into the DG/UX for Intel operating system's `/dev` directory. Table 3 summarizes these names.

*Table 3 New /dev Names for AViiON for Intel Servers*

| DG/UX for Intel Name | Comment |
|---|---|
| `/dev/console` | The SVGA console that you use when you boot the system. This is essentially an ASCII system console with no virtual console support. |
| `/dev/floppy` | The first floppy disk device (instead of the 88K platform's `/dev/pdsk/X` (where X is a unit number) |
| `/dev/keyboard` | The AT-101 keyboard device. On AViiON for Intel platforms, the keyboard runs in mode 1 rather than in mode 3 as on the 88K platforms. This is done to adhere to the ad-hoc convention that vendors have for using keyboards in mode 1. |
| `/dev/mouse` | The integrated mouse port on the back of the AViiON for Intel server or on the keyboard (for an integrated mouse/keyboard combination). You can also use a serial mouse on the port COM1 or COM2, if desired. |
| `/dev/tty00` | The server's COM1 port |
| `/dev/tty01` | The server's COM2 port |
| `/dev/parallel/lpt(0, X)` where `X` is 1-3 (typically 2) | The server's parallel printer port |

## A Note About Transferring Disk Drives Among Systems

The AViiON 88K and Intel servers use different disk formats. Therefore, you cannot perform a "plug and play" transfer of hard drives between the two platforms. Instead, you can copy a drive's data to tape (with **tar**, for example), move the drive to the other platform, format the drive, create the appropriate file systems, and reload the data. Or if your source and target platforms are on a LAN, and the source platform has multiple drives (and adequate space), you can copy the data to a different drive, move the drive to the target platform, NFS mount the appropriate file systems from the AViiON 88K server, and copy the data to the AViiON for Intel server.

## Updates to the Software Development Environment

DG/UX System release 4.10 includes updates to the development environment that could require some source code changes. These updates include some header file modifications and a new default for the `cc` command's `-X` option.

---

*Note* – You need to be concerned with these changes only if you are rebuilding your DG/UX R3.10 application—DG/UX System release 4.10 maintains binary compatibility with DG/UX R3.10. Note also that these updates are independent of Intel and 88K platform differences.

---

To comply with X/Open's XPG4 specification, the DG/UX System release 4.10 system headers introduced a number of new function prototypes and were modified so that some headers define additional functions. Some of the new prototypes are for new functions (such as `fputwc`—write a wide character), while others are for those traditional functions (such as `write`), that didn't have function prototypes in previous releases. This change may cause sources that formerly compiled correctly to get compilation errors if they are recompiled with release 4.10.

The typical kinds of problems this change can cause are:

- Redefinition errors caused by local use of names defined in an included header. The fix is to change your program to not use the offending name.
- Interface mismatch error messages. These are caused by incorrect invocations of header-defined functions. These invocations may have "worked" in a previous version, but are a potential source of failure in your software. The best fix for this problem is to code the invocation correctly.

Another change is a new default for the `-X` option of the `cc` command. The default has changed from `-Xt` (traditional) to `-Xa` (ANSI). The simple fix (if you get compilation errors) is to supply an explicit `-Xt` on your compile line.

## Porting Applications That Have a Graphical User Interface

The DG/UX for Intel X Window System is essentially the same as it is on DG/UX for 88K platforms. DG/UX for Intel supports version X11R5 of the X Window System and version 1.2.4 of the OSF/Motif window manager.

DG/UX for Intel provides local X display support, as well as all of the components needed to build X applications that are displayed on X terminals, workstation X servers, or other remote X displays.

### Motif 1.2.4

The DG/UX System release 4.10 software package includes an update from version 1.2.2 of Motif to Motif 1.2.4 and a new Motif window manager. Motif 1.2.4 is a bug fix enhancement of the Motif 1.2 product. Applications that were originally written to work with Motif 1.2 should work with this new revision with no or minor modifications.

If your application was built using DG/UX shared libraries you should need to do nothing but recompile. At run time, your application will be dynamically linked with the new `libXm.so` and should work properly. However, your application may not run properly if it used a "feature" of Motif 1.2 or Motif 1.2.2 that was reported as a bug and later fixed by OSF as part of Motif 1.2.4. In this case your application will reflect the new behavior of the library.

Applications that were linked statically, using `libXm.a`, will experience no behavioral changes. However, if you want your application to pick up the behavior of the new library, you will need to relink your application.

### Support for Fonts

Starting with DG/UX System release 4.10, the DG/UX operating system will provide only Portable Compiled Fonts (.pcf fonts). Prior to DG/UX System release 4.10, the operating system provided both .pcf fonts and Server Neutral Fonts (.snf fonts). If your application uses its own fonts, you should consider providing .pcf fonts to maximize portability.

## Moving UnixWare Applications to DG/UX for Intel

The DG/UX and UnixWare operating systems have a common history and follow many of the same source standards. Therefore, a UnixWare application is likely to compile and run on DG/UX for Intel without any modifications.

However, if your UnixWare application uses features of UnixWare that are not supported in DG/UX for Intel, you must modify your application to use the equivalent DG/UX for Intel features. Table 4 lists some UnixWare features that DG/UX for Intel implements differently or does not support.

You may find that features of the DG/UX Application Capture Option Package, described in the **aco** release notice, will provide an environment for your application that is more like UnixWare. If you're moving a UnixWare application to DG/UX for Intel, you may not have to do any porting. You may be able to move a UnixWare application's binary onto a DG/UX for Intel server, load and set up the optional **aco** package, and run the application.

*Note* – The **aco** package is a dynamic package—over time, it will be updated to provide support for additional UnixWare functions. However, because the **aco** package is not available on all DG/UX platforms and contains functions specific to UnixWare, using it limits the portability of your application.

*Table 4 UnixWare Features that Affect Portability*

| UnixWare Implementation | DG/UX for Intel Implementation |
|---|---|
| `/dev/kmem` | DG/UX for Intel does not support UnixWare-style `/dev/kmem` devices. DG/UX for Intel applications should not read `/dev/kmem` and should be changed to use the DG/UX system call interface to obtain the needed information. |
| `/etc/cmos` | Not supported |
| `/proc` | DG/UX for Intel provides the `ctrace` and `dg_xtrace` calls, which are similar in function to `/proc`. |
| Asynchronous I/O operations | DG/UX for Intel contains limited binary support for some UnixWare async I/O operations, provided for support of existing UnixWare applications. Applications ported to DG/UX for Intel should use DG/UX async I/O or `listio` interfaces. |
| BSD compatibility in `libucb.a` | Not supported. However, the DG/UX system supports some BSD compatibility through its feature test macros. |
| COFF and ELF | DG/UX for Intel does not support linking COFF objects with ELF objects. An application cannot link a COFF object built on another OS with ELF objects built on DG/UX. |
| Cumulative model of starting `init` scripts | Not supported in plain DG/UX for Intel. If the system is taken from `init S` to `init 3` on DG/UX for Intel, only those RC scripts in `init 3` are run. However, the optional **aco** package does provide support for the cumulative `init` script model. (Taking the system from `initS` to `init3` causes all `init` scripts in the successive levels to be run.) |
| DDI/DKI interface for drivers | Not supported. Refer to *Programming in the DG/UX Kernel Environment* for information about writing device drivers for DG/UX systems. |

*Table 4 UnixWare Features that Affect Portability (Continued)*

| UnixWare Implementation | DG/UX for Intel Implementation |
|---|---|
| Device naming conventions | Conventions for naming devices differ between DG/UX for Intel and UnixWare. An example of a tape device naming difference is:<br>`DG/UX: /dev/rmt/0`<br>`UnixWare: /dev/rmt/ctape1`<br>Floppy device names also differ on the two operating systems. Some compatibility for UnixWare device names is provided in the optional **aco** package. |
| Dynamically loadable modules | Not supported |
| File systems (proprietary to UnixWare): `ufs, s5, vxfs, bfs, or sfs` | Not supported. Use standard file systems. |
| Internationalization: `/etc/default/lang` | Support for `/etc/default/lang` is provided only in the optional **aco** package. To maintain portability, applications should use DG/UX `/etc/TIMEZONE`. |
| Persistent `/dev` | In DG/UX for Intel, `/dev` is a memory resident directory and is re-created every time the system is rebooted. Applications should not put entries in `/dev`. |
| Physical disk format | DG/UX for Intel does not support the physical disk format of UnixWare. Applications will fail if they attempt to read the UnixWare VTOC or other UnixWare internal structures directly from raw disk. Use standard interfaces to request information. |
| Remote File System (RFS) | Not supported. DG/UX System release 4.10 supports NFS. |
| Security features | Not supported. However, the DG/UX DSO security option provides a range of features, including B2 level of assurance, virus protection, access control lists, mandatory access control, auditing, capability access control, and Trusted IP. |
| Streams-based sockets | DG/UX for Intel supports Berkeley-style sockets. Applications should not send streams **ioctls** to sockets or attempt to perform any other streams operations on sockets. |
| System administration utility (**sysadm**) | DG/UX for Intel uses a proprietary interface description language, **idl**, to define **sysadm** menus and options. System administrators can use **idl** to customize **sysadm**. |
| **uname** command | The output of this command is different in DG/UX for Intel and UnixWare; applications should not depend on output of this command. |

*Table 4 UnixWare Features that Affect Portability (Continued)*

| UnixWare Implementation | DG/UX for Intel Implementation |
|---|---|
| Tape `ioctls` | DG/UX for Intel does not support many of the UnixWare tape `ioctls`. Applications should use Data General's multithreaded `ioctls` to make `ioctl` calls on tape devices. |
| Threads | DG/UX for Intel implements POSIX threads (Pthreads). |
| Xenix support in `libx.a` | Not supported |
| XTI library network interface | Not supported. The DG/UX system supports the TLI library interface. |

## Other Documentation

In addition to *Porting and Developing Applications on the DG/UX System* (069-701059), several other documents (and providers of documents) may be of interest to porting engineers and programmers.

- *Release Notice: DG/UX System 4.10 Software*
- *Programming in the DG/UX Kernel Environment* (093-701083)
- Novell— information about UnixWare
- X/Open Company Limited—information about the X/Open Portability Guide (XPG) and the Common Application Environment (CAE)
- IEEE—information about POSIX standards
- Dr. Dobb's Journal—*Endian Neutral Software*—October 1994 and November 1994 (a two-part article)
- DG/UX Technical Brief *Support for Threads in the DG/UX 5.4 Operating System* (012-004405)