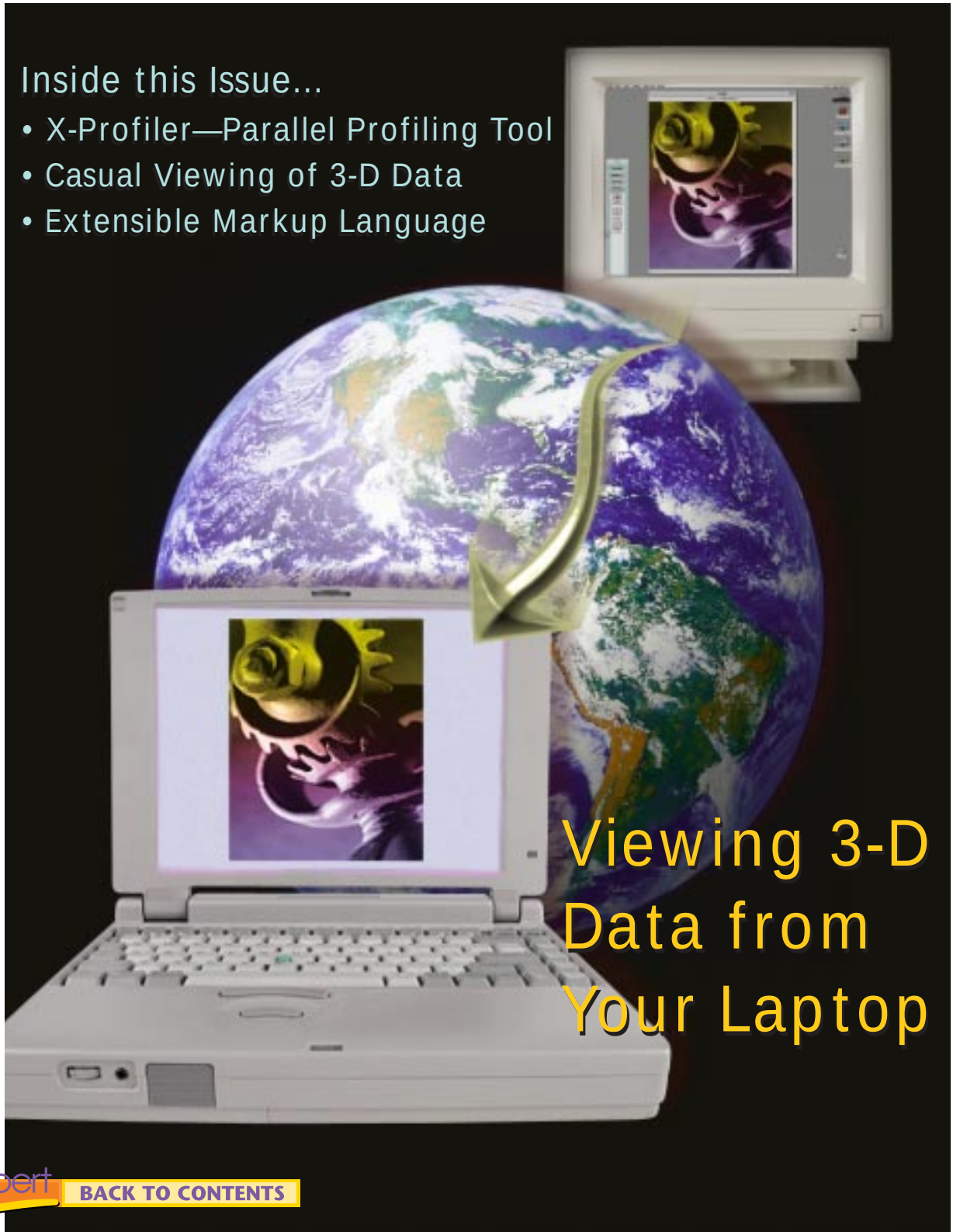
IBM'S MAGAZINE FOR AIX DEVELOPERS SEPTEMBER 1998

AIXpert

FOCUS ON 3-D GRAPHICS

Inside this Issue...

- X-Profiler—Parallel Profiling Tool
- Casual Viewing of 3-D Data
- Extensible Markup Language

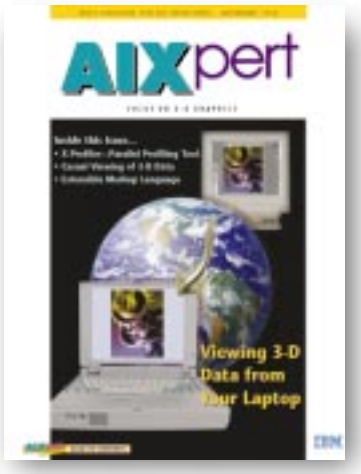


Viewing 3-D Data from Your Laptop

IBM

AIXpert
DIGITAL VERSION
IBM

TABLE OF CONTENTS



Click here to view a full-size version of this issue's cover.

Click here to go to a complete version of this issue for printing.

To submit an article for publication in *AIXpert*, send e-mail to one of the following:

deb@tdagroup.com
gnoren@us.ibm.com

September
1998

AIXpert BACK TO MAIN MENU

If I Can't CAD, I'm DOOMed!

COMMENTARY

Ever wonder why it is that you can play sophisticated 3-D games on your laptop computer (well, maybe your child could if you'd allow it but, of course, it's a business computer), but you need a heavy-duty workstation to just review or approve a 3-D drawing from your CAD software? Well, wonder no more. Instead, read our feature article about using Virtual Frame Buffers. Virtual Frame Buffers allow you to check the details of a design drawing back at the office while you are in your hotel room on a trip to confer with your client. Now you can tap into the power of your home-office graphics software using the same laptop computer that has become a faithful traveling companion.

Perhaps, you've heard of a new language, "XML", and wonder how you, as an AIX developer, can make the best use of this new technique. Check out our article on that subject and you'll know. Also, find out how system administrators can take advantage of the new features in the AIX switchuser (`su`) command. Also, find out how programmers can visually profile an application's performance data at the procedure and source statement level on the RS/6000 SP system. Our article introduces this X-Windows based profiling environment. As you can see, this issue has more than a few things to hold your interest.

So, the next time someone at the water fountain asks you, "How do I make AIX system calls from a Java program using Java Native Interface (JNI)?", you'll be prepared with an answer. **IF** you've read this month's Question and Answer section.

George Noren

George Noren, IBM Corporation, Internal Zip 1034, 11400 Burnet Road, Austin, TX 78758. Internet: geo@austin.ibm.com. Since joining IBM in September 1979, Mr. Noren has written hardware and software manuals, including AIX and RS/6000 manuals, and was a member of the InfoExplorer™ design team. He has worked as a system administrator for several AIX systems and is a Certified AIX System Administrator. He is currently Editor in Chief of the World Wide Web site for IBM's Solution Developer Program (www.developer.ibm.com) in addition to his work with AIXpert *Magazine. Mr. Noren studied engineering at Illinois Institute of Technology, holds a BA in English from the University of Minnesota and an MBA from St. Edwards University in Austin.*

George Noren

Casual Viewing of 3-D Data from a Remote System

By Jeanne Sparlin and Andrew Taylor

The Virtual Frame Buffer and the associated OpenGL enhancements allow programmers to create environments for the low-cost viewing of 3-D data by a casual user.

Graphics users sometimes need to view a 3-D object and even look at the object from different angles and viewpoints. For example, suppose the engineering manager for a company needs to approve a design while traveling on a business trip. The manager simply needs to view the design from different angles and different perspectives, but does not need to edit or animate the object. In this scenario, the PC or laptop does not have either graphics software or hardware required to edit or animate the object, and the communication connection is very slow.

In these types of scenarios that require casual viewing of graphics, the data being viewed is often stored at a remote location from the person who is viewing the data. A super-graphics workstation is not necessary because the requirement is intermittent and occasional compared to a routine or daily use.

This article discusses the Virtual Frame Buffer and Open GL enhancement now available in AIX® to address the issue of casual viewing of remote 3-D graphics—which has previously required a workstation and graphics adapter. AIX now offers several alternatives for casual viewing of 3-D graphics.

Architectures for Remote Graphics

Figure 1 shows a traditional architecture for remote graphics. The application, X Server, and 3-D rendering software reside on the local workstation; the data resides remotely on the server. Each person who views the data uses a system that supports an X Server, 3-D rendering software, and a graphics adapter.

The application retrieves the model to be viewed from a remote system. In addition, all supported UNIX® vendors and Wintel[1] platforms need separate binaries for the application. This environment generally does not support network computers because they are smaller, lightweight systems and cannot support the system requirements needed to display and manipulate the graphics objects.

One advantage of this architecture is that it enables good performance for manipulating the viewed object. Because the model and rendering software reside on the same machine as the graphics hardware, the display is very fast.

Some disadvantages, however, include the higher cost of workstations and software because they require separate binaries for each UNIX and Wintel system that is supported. Applications also tend to be expensive because of the cost associated with developing and testing on various supported platforms.

Jeanne Sparlin

[1] Wintel refers to the combination of the Windows™ operating system running on Intel® processors.

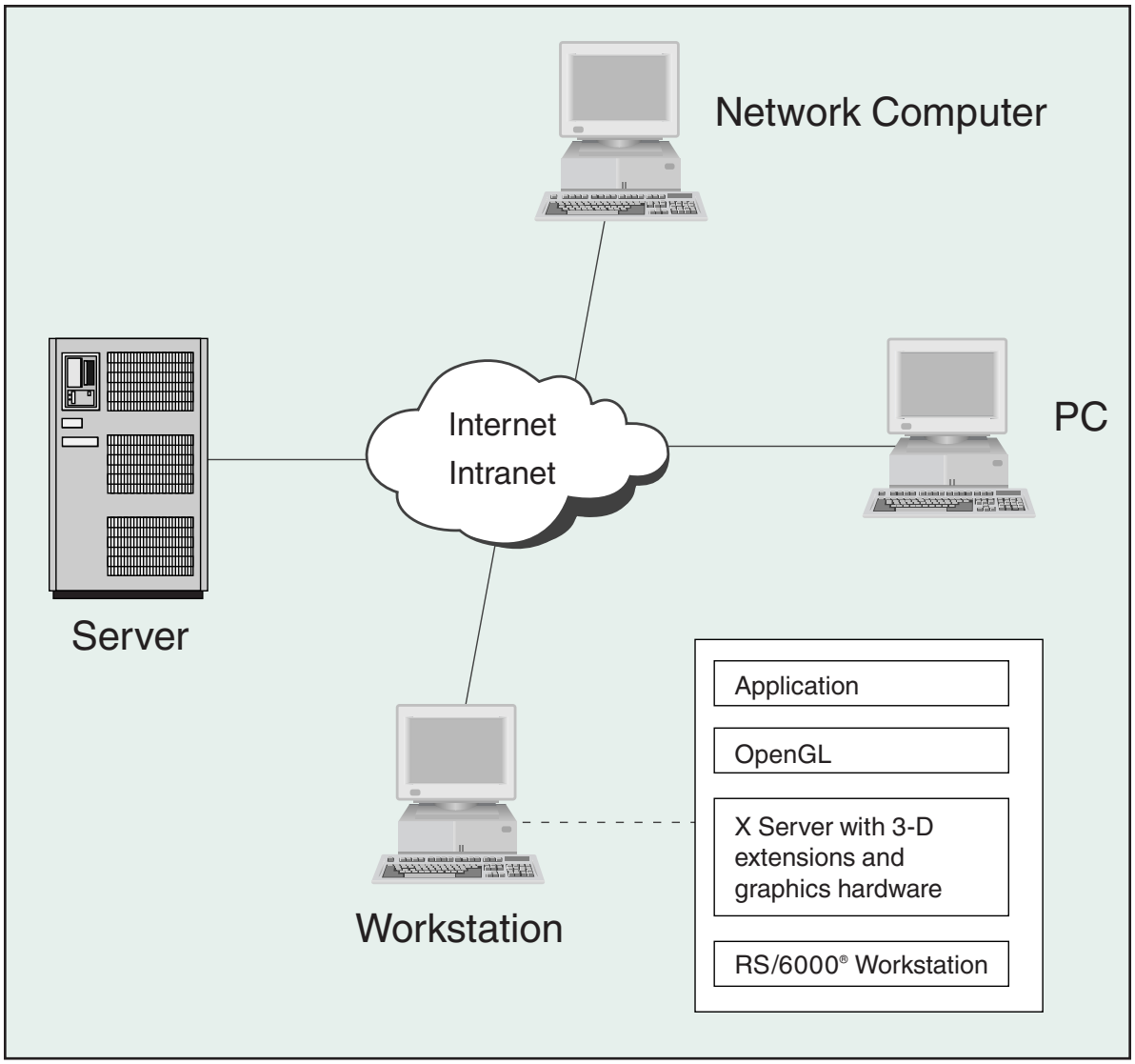


Figure 1. Traditional workstation environment

Figure 2 shows another architecture for remote graphics: remote application/local rendering software. The client application and data reside on the server system and the rendering software and graphics hardware reside on the client system.

The application sends the protocol to the X Server or 3-D extension to the X Server, for example, OpenGL™. The local workstation requires an X Server, the appropriate 3-D rendering software, and possibly graphics hardware. Most major UNIX vendors provide X Servers and appropriate 3-D extensions for their systems. X Server products for the PC are available for Wintel systems.

The X Server and 3-D rendering software are large processes, so they use valuable system resources. Network computers do not have the resources or software to operate in this environment.

One advantage of the remote application/local rendering architecture is that it requires only one application binary that executes on the server system. The development cost is lowered because the application is tested on only one system or hardware architecture.

Disadvantages of this architecture include lower performance because of the amount of data sent across the network each time the model is rerendered. In some cases, the

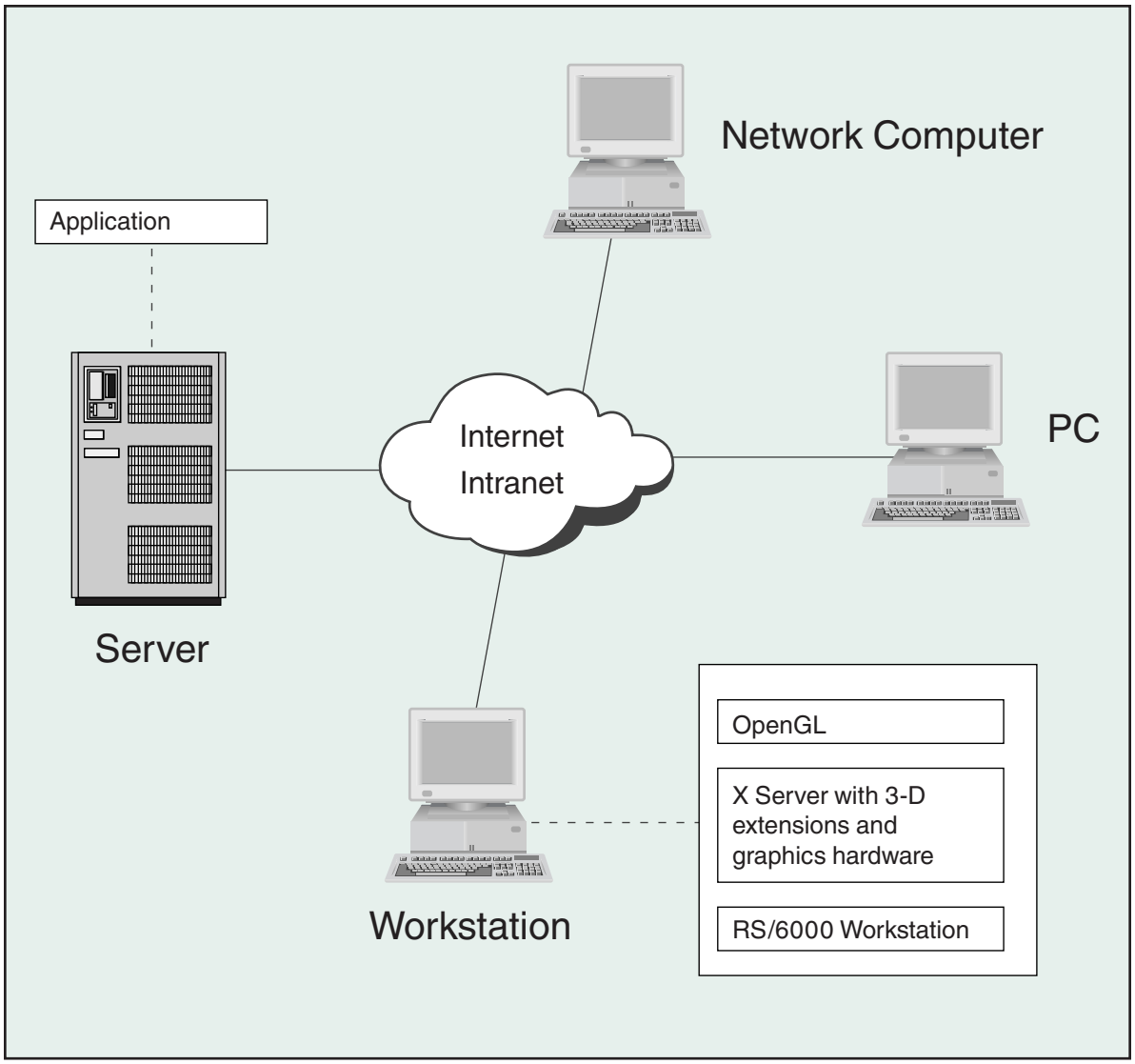


Figure 2. Remote application using local rendering software

inability to direct hardware access can also lower performance. All client systems need an X Server and 3-D graphics software.

Figure 3 demonstrates the Virtual Frame Buffer (VFB) environment. The application, X Server, and 3-D rendering software all reside on the server system. The application renders images on the server, then distributes images to viewing stations on a network, saves images to a database, or uses them in some other way.

Within this client/server or rendering server environment, the end user does not directly drive the application. Instead, the application receives commands from software that is driven by an end user on a low-end system. Although the server system (host machine) may have a graphics adapter, it will more likely contain a Virtual Frame Buffer (VFB) or a non-visible frame buffer.

The VFB and associated OpenGL enhancements are new IBM rendering technologies designed specifically to make 3-D rendering of server applications more efficient and scalable.[2] These two new rendering technologies are discussed in the next sections.

[2] OpenGL is currently supported with Virtual Frame Buffer, but graPHIGS™, PEX, and OpenGL 3.2 are not.

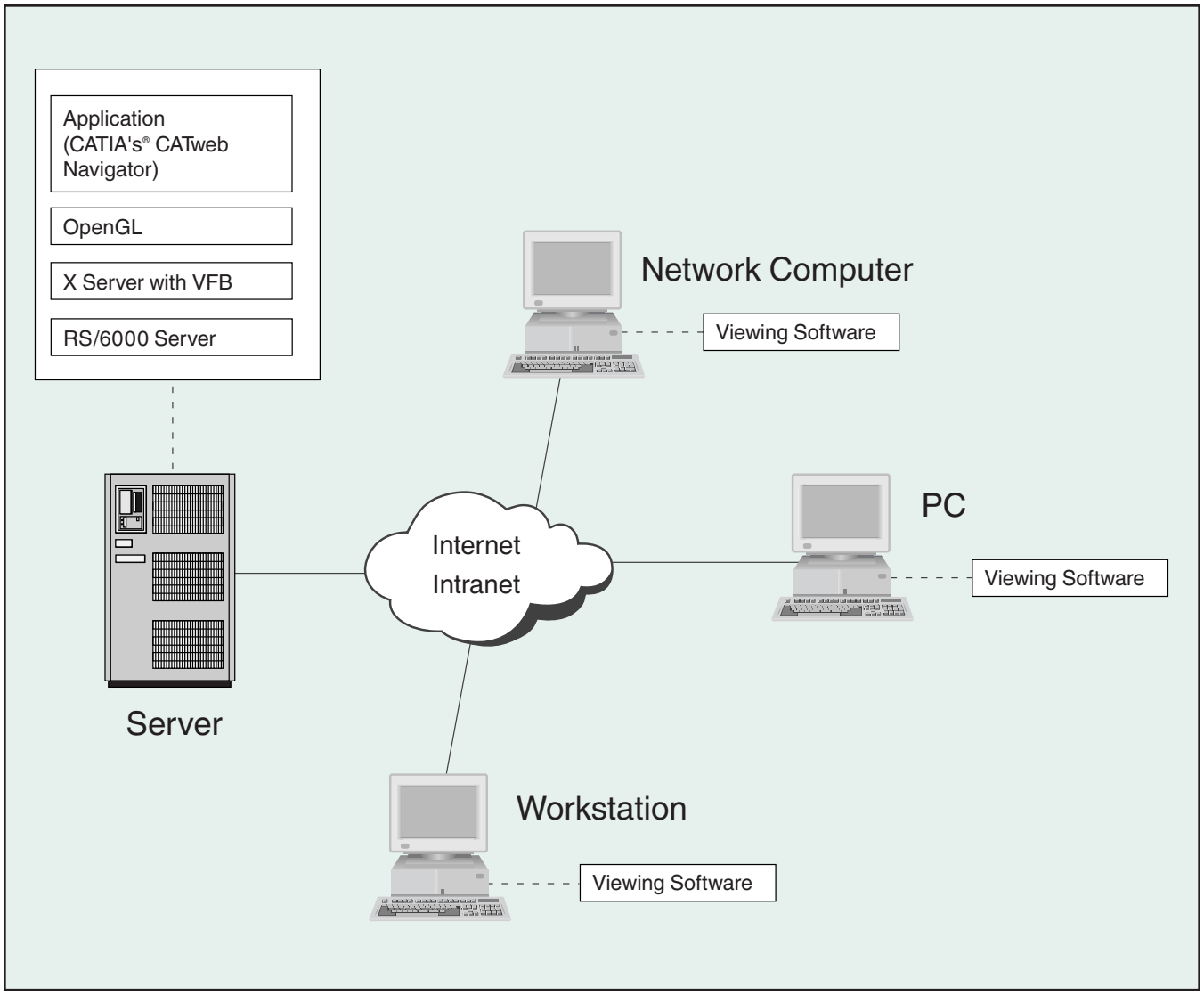


Figure 3. Rendering server environment

Rendering Server Environment

The Virtual Frame Buffer allows the X Server to initialize and run without any physical graphics device present. In the past, the X Server has required one or more graphics devices in order to run, and would exit if no graphics devices were present.

Figure 4 shows the traditional X Server architecture. The shaded section contains rendering software tuned for each graphics adapter. In the VFB solution, this software is replaced by X device-dependent software (ddx) that drives a software graphics adapter. That is, the frame buffer is stored in system memory and all graphics rendering, such as lines or polygons, is handled completely within the software using the system CPU.

VFB is intended for use in a rendering server environment. This means that when the X client application renders an image, the application will then query the image back to the application (for example, `XGetImage`), save it to a file, distribute it to a network, or save it to a database. In this mode, an end user does not directly use the X application interactively. Instead, the application is being driven remotely as a rendering server.

Because there is no physical graphics device, no special hardware, such as RAMDAC,

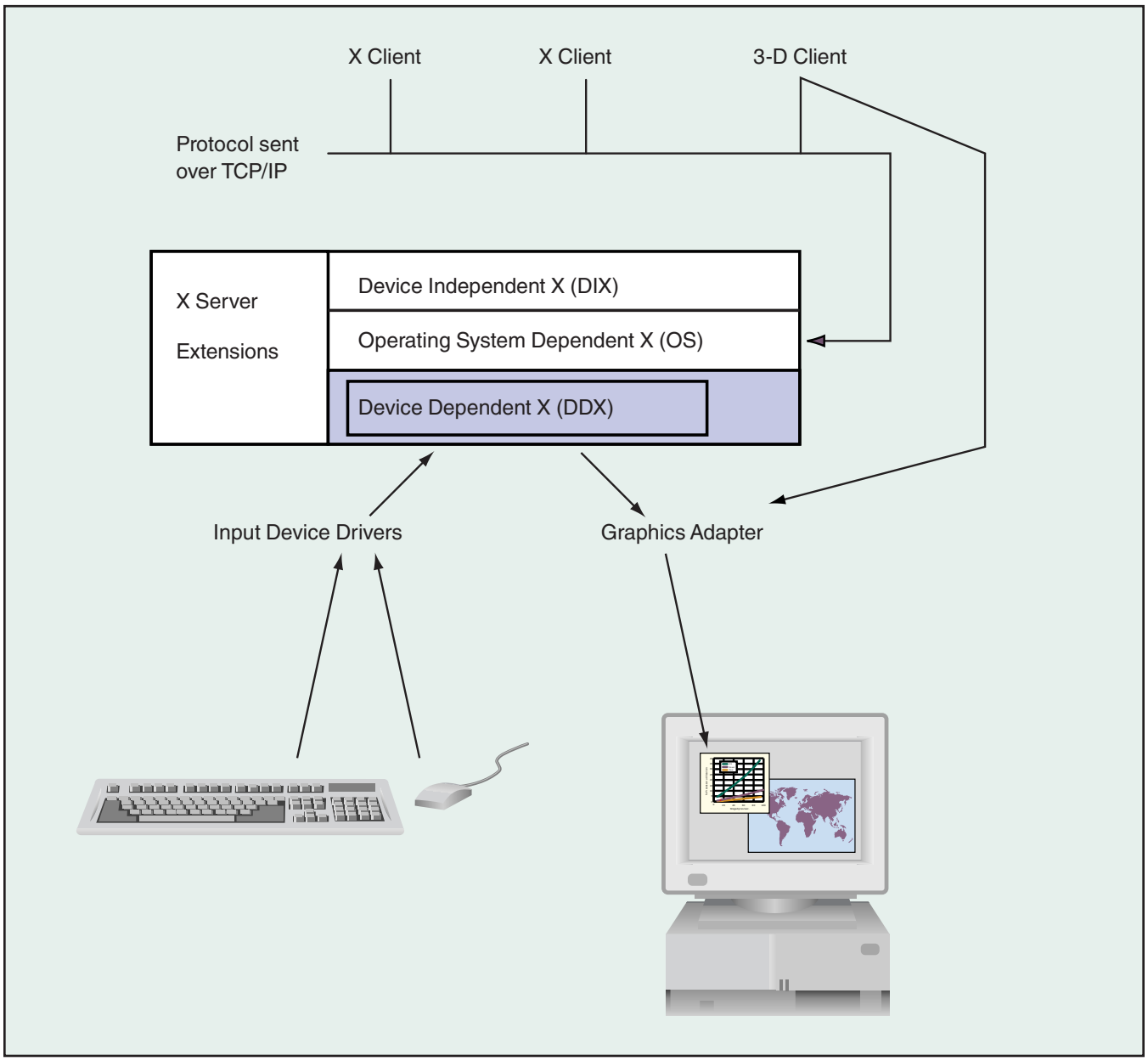


Figure 4. Traditional X environment

is necessary to generate signals used to display the images on the monitor. Therefore, it is impossible to connect a monitor to the system and view the contents of the X Server Virtual Frame Buffer. Although this takes some time to get used to, it is a good solution for rendering server environments, which do not require a high level of interactivity. It also avoids the added expense of a graphics adapter.

One disadvantage, however, is that debugging applications can be more difficult because you cannot see the contents of the frame buffer directly. For this reason, it is helpful to develop applications with a physical graphics adapter, then port the applications to the VFB environment.

X client applications like `xwininfo`, `xwd`, and `xwud` (which are shipped with the AIXwindows® environment) can be used to help verify that your application is running correctly. The `xwininfo` client application provides information about windows, such as the window ID,

geometry, color class (such as PseudoColor, TrueColor), and depth. The xwd application extracts the image and colormap information for a specified window and stores the data in xwd format. Then, xwud displays data that was extracted using the xwd command.

Note: If you are running OpenGL and want to use xwd, set the _OGL_MIXED_MODE environment variable. This environment variable turned on in production mode will slow performance, because drawing occurs in both the vfb frame buffer and the special OpenGL private frame buffer. It is best to use this environment variable in debug mode because its use may slow performance.

The VFB environment does not require input devices. You cannot see the frame buffer, so moving the mouse to do traditional mouse input is impossible. New methods of performing application input are discussed in the next section.

Enhancements to OpenGL

The soft graphics versions of the OpenGL application programming interface (API) contain enhancements designed to work specifically with the Virtual Frame Buffer. These enhancements provide a full software implementation of OpenGL (geometry pipeline and rasterizer), which supports a direct OpenGL context.

OpenGL uses these enhancements to create a private software frame buffer and Z buffer (and possibly alpha, stencil, and accumulation buffers) in system memory for each OpenGL drawable. Consequently, when OpenGL renders graphics, all rendering is drawn in this private software frame buffer. Since each drawable has its own custom-sized software frame buffer to fit the drawable size, rendering into one drawable will not overlap with rendering into another drawable. This is true even if the corresponding X windows for each drawable overlap. Because OpenGl is rendering into its private software frame buffer, there is no data transfer to the OpenGL extension of the X Server. The result is improved performance, since all of the OpenGL graphics commands are rendered directly to the frame buffer.

When images are read back from the drawable (for example, glReadPixels), the read is from the private software frame buffer. As a result, multiple OpenGL rendering server applications can all render concurrently. Since each drawable is rendering to its private software frame buffer, the results from glReadPixels will be correct and complete, because no other application's windows will be occluding parts of the target application's windows.

Enabling multiple applications to render concurrently is a very important feature and causes symmetric multiprocessing (SMP) machines to be fully exploited. For example, running four or more OpenGL applications concurrently on a four-processor SMP machine will cause all four processors to be used.

The VFB and associated OpenGL enhancements are new IBM rendering technologies designed specifically to make 3-D rendering of server applications more efficient and scalable.

To further increase performance, the default OpenGL behavior does not transfer (blit) the contents of the private frame buffer to the VFB frame buffer. As a result, you cannot mix X and OpenGL rendering to a single window. This behavior is fine for most applications, since the image will be queried from the private OpenGL frame buffer using glReadPixels. If required, the transfer behavior can be changed by setting an environment variable (_OGL_MIXED_MODE) that will cause the private frame buffer to be "blitted" to VFB. This environment variable is useful for developing and debugging rendering server applications, but it is not recommended for runtime execution.

```
Start the X Server using XVFB
            /usr/lpp/X11/bin/X -force -vfb -x GLX -x abx -x dbe

      Run the xclient client application
            xclock -display :0.0 &

Use the Xwd and xwud to extract the image of the entire root window on the vfb
X server and display the image on another system
            xwd -display :0.0 -root | xwud -display somedisplay:0.0
```

Figure 5. Rendering of an XVFB server displaying on a second server

Exploiting the VFB Server Environment

Applications should be enhanced in order to operate in a VFB environment. The following represent several enhancements that are needed:

- **Method to extract rendered image from rendering server**. The image to be displayed is extracted or retrieved from the rendering server. Typically, an application would do an `XGetImage` for X applications or a `glReadPixels` for OpenGL applications.

  The programmer must decide for the application the frequency and type of actions that will extract the image from the rendering server. Typically, an application draws an image to the back buffer and "swaps" the back buffer for the front buffer when the image is complete. At this swap time the image could be extracted since the programmer knows that the image is completely drawn. For example, one criteria might be each time the buffer is swapped using the `XdbeSwapBuffers` or `glXSwapBuffer` subroutine.

- **Method to send commands to remote application on server platform**. The application needs input to perform actions, such as opening files or transforming objects. There should be a method to send input to the application from the viewing station. This method could be a command language, a socket connection, interaction with an HTTP server, or another type of communication service. The `XRecord` and `XTest` extensions could be used to send events to the X Server and communicate with the application through traditional X events.

- **Method to display image on a display station**. The VFB environment should contain a method to display the rendered image on the viewing station (for example, the network computer, PC, or workstation). The programmer can determine the display method for the extracted image.

If the viewing station is an AIX workstation executing an AIXwindows X Server, you can use the `xwud` command to display the extracted image. The `xwud` takes an image in the `xwd` format, creates a window, massages the colormap, and does an `XPutImage` to display the extracted image. Figure 5 shows a simple example of rendering on a `XVFB` server and displaying on another server.

More sophisticated methods can include image compression and Java™ display applets. The advantage of a Java display applet is that the image can be displayed on a variety of platform types.

Configuration of the VFB Environment

The `X11.vfb` fileset must be installed before you can use the VFB environment. If it is not available, follow the usual instructions for installing filesets.

Once the `X11.vfb` fileset is installed, start the X Server using the `-vfb` option, shown in Figure 6. In this example, the `-force` option allows you to run the X Server (that is, `X` command) from any shell. If you do not specify the `-force` option, you should invoke the X Server from the low-function terminal (LFT).

```
/usr/bin/X11/X -force -vfb -x GLX -x abx -x dbe
```

Figure 6. The -vfb option

```
/usr/bin/X11/xinit — -force -vfb -x GLX -x abx -x dbe
```

Figure 7. The .xinitrc file

```
xvfb:2:respawn:/usr/bin/X11/X -force -vfb -x GLX -x abx -x dbe >/dev/null
```

Figure 8. Entry to start the X Server automatically


```
/usr/bin/X11/xinit — -force -vfb -x GLX -x abx -x dbe
```

Figure 7. The .xinitrc file


```
mkitab "xvfb:2:respawn:/usr/bin/X11/X -force -vfb -x GLX -x abx -x dbe >/
dev/null"

rmitab "xvfb"
```

Figure 9. Commands to remove entries for automatically starting X Server

The `-vfb` option instructs the X Server to ignore any physical graphics devices that may be present on the system and use the Virtual Frame Buffer instead. The remaining options (`-x GLX`, `-x abx`, and `-x dbe`) instruct the X Server to load extensions that are necessary to render through OpenGL.

By invoking the X Server directly, not by calling `xinit` or invoking it through the desktop, no client applications—including the window manager—are automatically started. Executing without a window manager in a rendering server environment is generally fine because windows will not be manipulated directly with the mouse or keyboard.

To invoke a window manager, you can invoke the X Server directly, as shown in the example above, and then invoke `mwm`, `dtwm`, or some other window manager. In the example shown in Figure 7, the `xinit` command both starts the X Server and several client applications including `mwm`. The client applications that are started automatically are configured in the `.xinitrc` file.

When the X Server is used in a rendering server role, it is often desirable to have the X Server start automatically. This can be done by adding an entry to the `/etc/inittab` file. The entry shown in Figure 8 will cause the X Server to start automatically when the system boots, and also to restart automatically if the X Server should die.

Figure 9 shows two commands that can be used to add or remove the entry shown in Figure 4 from the `/etc/inittab` file.

Some applications may need to execute differently when running in the Virtual

```
/usr/lpp/X11/Xamples/bin/xprop -root | grep XVFB
```

Figure 10. Command to check for XVFB use

Frame Buffer environment. To facilitate applications and tune their behavior for XVFB, the X Server automatically sets the XVFB_SCREEN property to TRUE for any screen that is using XVFB.[3] This property is stored on the root window of any VFB screen.

Users can check the XVFB property by using the xprop command, a utility to display X Server properties associated with windows and fonts. To check for XVFB use the command shown in Figure 10.

To check for the XVFB property, the application would use the XInternAtom

```
      Atom   atom;
Atom  actual_type;
Display      *display;
Screen       screen;
unsigned long       nitems;
unsigned long       bytes_after;
unsigned char       *prop;
int                vfb_screen = False;
.
.
.
/* SET UP SCREEN VARIABLE*/
.
.
.
.
.
/* OBTAIN THE ATOM THAT CORRESPONDS TO THE "XVFB_SCREEN" STRING*/
atom{1} = XInternAtom (display, "XVFB_SCREEN", True);

if (atom != none) {

      /* ATOM EXISTS SO GET THE PROPERTY*/
      if ((status = XGetWindowProperty (display, RoowWindow(display, screen),
             atom, 0, 100, False, atom, &actual_type,&actual_format,
             &nitems, &bytes_after, &prop)) == True ){

             /* BE SURE THAT THE PROPERTY IS SET TO TRUE*/
             if (strcmp((char *)prop,"TRUE") == 0) {
                   vfb_screen = True;
              }
      }
}
```

Figure 11. Checking for the XVFB property

[3] A property is an X Window System® concept. Each property consists of a name, type, and data. Each property is associated with some X Server resource such as a window, font, pixmap, or colormap. X Window System subroutines are available to obtain property information.

```
screen #0:
 dimensions:    1280x1024 pixels (325x260 millimeters)
 resolution:    100x100 dots per inch
 depths (2):    1, 24
 root window id:    0x26
 depth of root window:    24 planes
 number of colormaps:    minimum 1, maximum 1
 default colormap:    0x24
 default number of colormap cells:    256
 predicated pixels:    black 0, white 16777215
 options:    backing-store NO, save-under NO
 current input event mask:    0x0
 number of visuals:    3
 default visual id: 0x21
   visual:
   visual id:    0x21
    class:    TrueColor
    depth:    24 planes
    size of colormap:    256 entries
    red, green, blue masks:    0xff0000, 0xff00, 0xff
    significant bits in color specification:    8 bits
 visual:
    visual id:    0x22
    class:    TrueColor
    depth:    24 planes
    size of colormap:    256 entries
    red, green, blue masks:    0xff0000, 0xff00, 0xff
    significant bits in color specification:    8 bits
 visual:
    visual id:    0x23
    class:    TrueColor
    depth:    24 planes
    size of colormap:    256 entries
    red, green, blue masks:    0xff0000, 0xff00, 0xff
    significant bits in color specification:    8 bits
```

Figure 12. Sample output for VFB screen

and `XGetWindowProperty` subroutines. Figure 11 shows one way that the code could be written.

The current `XVFB` screen only supports 24 bit visuals.[4] The `xdpyinfo` command will return X Server and visual information about screen #0. Figure 12 shows sample output for a VFB screen.

The maximum number of concurrent client applications for an IBM X Server is 128. Client number 129 and those beyond will receive the error message shown in Figure 13.

```
Xlib: Connection to “displayname” refused by server
Xlib: Maximum number of client reached.
```

Figure 13. Error message for clients above 128

[4] A visual is an X Window System concept that gives hints about the configuration of the rendering environment such as depth, pixel type, organization, number, and size of colormaps.

```
/use/bin/X11/X  -vfb -x GLX -x dbe -x abx -auth $HOME/.Xauthority

xinit — -vfb -x GLX -x abx  -auth $HOME/ .Xauthority
```

Figure 14. Invoking an X Server with .Xauthority

```
Display     *display
GLXContext  context
int   OGLEnhancement = False:
.
.
.
.
if ((glXIsDirect(display,context) == True) &&
    (strcmp(glGetString(GL_RENDERER), "SoftRaster") == 0){}
      OGLEnhancement = True;
}
```

Figure 15. Checking for the VFB environment

Security Configuration in X Window System

This section describes several hints about the security configuration in the X Window System environment.

The display access control of the XVFB X Server is the same as the access control for any other X Server: it can be host-based or it can use the `MIT_MAGIC_COOKIE`. When using host-based access, any client on a host in the host access control list can access the X Server. The list of allowed hosts is stored in the X Server and can be changed with the `xhost` command. `xhost+` will allow all remote X client applications to connect to the X Server.

Another method is to create an `/etc/X0.hosts` file. Any host name added to this file can connect to the X Server. The `-ac` flag is specified when the X Server is invoked. The result is the same as using `xhost+`.

The IBM X Server also supports the `MIT_MAGIC_COOKIE_1` mechanism for access control. When using these mechanisms, the client sends a cookie with the connection setup information. Cookies are machine-readable, randomly generated access values that are stored in a `~/.Xauthority` file. When the X Server starts a session, it reads a cookie from the `~/.Xauthority` (or other file specified by the user in the `-auth` argument to the X Server). Subsequently, only clients that know the cookie can connect to the X Server. When it is used this way, `xauth` can limit access to specific users on specific systems.

The `~/.Xauthority` file can be generated each time the X Server is invoked. It then is copied to a remote system when a user wants to connect. Figure 14 shows how to invoke the X Server by specifying an `.Xauthority` file.

The `.Xauthority` file can be generated by many different methods. A random number generator usually creates the key, and the `xauth` command is called to add the key into the `.Xauthority` file.[5] Scripts can be written to copy the `.Xauthority` file to client workstations when specific users log on to the host system.

OpenGL Configuration

To ensure that the OpenGL enhancement for VFB is available, your application should

[5] The `xauth` command, `xdm` command, and other access information are documented as part of the AIX publications.

verify that a direct context and a soft rasterizer are available. In direct context, OpenGL is rendering directly to the frame buffer (for example, hardware or VFB). This value can be determined by using the `glXIsDirect` subroutine. A soft rasterizer means that OpenGL is rendering by using software and not hardware. Both a direct context and soft rasterizer must be true to use the OpenGL enhancement. Figure 15 shows sample code that you can add to your application to check for the VFB environment.

VFB Server-supported Operating Environments

Virtual Frame Buffer Server-supported operating environments include AIX 4.1.5, AIX 4.2.1, AIX 4.3.1, or later versions.

Sample code for modifying applications to exploit the VFB environment can be found at *http://www.rs6000.ibm.com/solution/interactive/rendserv.html*.

Jeanne Sparlin, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Sparlin is currently working in the Business Development Department of IBM Visual Systems. She has previously worked in the Data Management, SNA, Dialog Design Aid, and the X Window System development departments on the RS/6000 and RT/PC. She has a degree from Truman State University.

Andrew Taylor, IBM Corporation, 11400 Burnet Road, Austin, TX 78758.

The Switchuser Command

By George Kraft IV

Any user on the UNIX system can become another user on the system via the switchuser `su` command. This article describes the latest improvements in AIX for the superuser.

The superuser is the privileged administrative account that manages the system for the multipurpose and multiuser UNIX® operating system. Installation and configuration setup of an AIX® system requires superuser authority. For normal operation of the system, the superuser can create and delete accounts, start and stop processes, configure input/output devices, set up networking, and shut down the system. The superuser also can grant privileges to other users on the system.

The privileged superuser account has such a key role in maintaining the UNIX operating system that the system has special allowances for the superuser. Processes run by the superuser bypass runtime and access security restrictions. In addition, when the operating system's process table is full, it reserves some space for the superuser to log in and clean up the system.

The superuser, usually called *root*, has the numeric user ID (UID) of zero (0), shown in the `/etc/passwd` file. The UID 0 indicates the identity of the superuser to the system. Any user with UID 0 has the superuser privileged access, and because of the privileges mentioned above, it is best to run as root as infrequently as possible to prevent accidental system outages due to operator error.

To prevent accidents and to control the use of the root superuser account, by default AIX does not allow programs to execute or library files to be included in the local directory (for example, "."). AIX controls when root can log in and from which TTY ports it can log in. The system can also audit the session of any user including root.

The su Command

With the switchuser `su` command, the superuser can also assume the identity of any user on the system. The `su` command example below assumes the identity of user gk4. The superuser can change its identity from UID 0 to gk4's UID of 1234, which allows the superuser to create and modify files as the user gk4. This is useful for repairing, restoring, and setting up files and directories.

```
/usr/bin/su gk4
```

Although the superuser can assume the identity of user gk4, this does not completely change the environment of the root superuser. To become a completely different user, the superuser must issue the `su` command with the dash (`-`) option, which changes the home directory and initializes its shell environment.

George Kraft IV

The dash option tells the su command to initialize the user shell indicated in /etc/password and to run the user's .profile or .login startup script.

```
/usr/bin/su - gk4
```

In both examples above, the superuser assumed the identity of user gk4 without prompting for the user's password. This takes place because the root user has unrestricted access within the system. However, user gk4 could assume the identity of root—or any other user on the system—by running the su command with a username argument and entering the corresponding user password, when prompted. The su command validates the user account and grants it permission to su.

When using the su command, the non-superuser must authenticate the user's identity by entering the password corresponding to the user. This establishes the user's credentials according to the /etc/passwd and /etc/group databases. However, it is generally recommended that you not divulge account passwords that allow others to assume your identity. If you do share your password with others, then the AIX system administrator could control who can su and who can be "su ed".

The suspend Command

The su command spawns a new shell. When finished, the user can either exit this new shell or suspend the current shell and return to the previous shell. When the shell is suspended in the background, the user can return to the shell in the foreground by using the fg command. This is useful to quickly switch back and forth between the user and root identity, because it is best to run in privilege mode as little as possible.

The su command can spawn a new shell and assume the identity of another user, or it can momentarily change identities while running a specific command. In the example below, a user starts the System Management Interface Tool (SMIT) command as the root user for system administration purposes. Once the smit command exits, the root shell exits and the user's shell is resumed.

```
/usr/bin/su -c /usr/bin/smit
```

A how-to su example appears below in a shell script. The shell's set -x command echos to the screen the command to run. Then the /usr/bin/su -c /usr/bin/smit prints on the screen and prompts the user for the root password to run the smit command.

```
/bin/ksh -c `set -x;
/usr/bin/su -c /usr/bin/smit`
```

Users often find the su command useful in shell scripts to temporarily become the superuser. However, it is prudent to use set -x to show why a prompt appears for the root password.

```
ACTION RebootSU
{
     TYPE          COMMAND
     WINDOW_TYPE   TERMINAL
     EXEC_STRING   /usr/bin/ksh -c 'set -x; \
                   /usr/bin/su - root -c /usr/sbin/shutdown -r now
"Rebooting..."'
DESCRIPTIONThe RebootSU desktop action prompts for the root password,
then shuts down the system for a reboot.
}
```

Figure 1. Desktop action to reboot the system

The Desktop Environment

In the Common Desktop Environment (CDE), it is possible to create a graphical library of administrative commands. However, since the desktop inherits commands from a variety of places, it is dangerous to simply enter the root password in a graphical user interface (GUI) prompt dialog window. Entering the root password prompted by a GUI could actually be coming from a Trojan Horse program trying to gain access to the system.

Figure 1 shows a good example of desktop action which reboots the system. The action opens a terminal emulator window, prints the `shutdown` command to be executed for the user to see, and then prompts the user for the root password.

Conclusion

Any user on the UNIX system can become another user on the system via the `su` command. You can assume the user's home directory and shell initialization by using the dash (`-`) option or simply changing your UID. To return, the user can either exit from or suspend the new shell.

In addition to creating a new shell as another user, you can temporarily become another user to run a command by using the `-c` option. From within a shell script or from a desktop action, you can print to the screen the specific command for which the system is prompting the password.

To change who can issue the `su` command and/or who receives the `su`, run `smit users` command and see the options under Change/Show Characteristics of a User. From the AIX SMIT, the system administrator can control who can and cannot use the switchuser command.

George Kraft IV, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Kraft is an advisory software engineer for IBM's Network Computer Division. He recently moved from IBM's RS/6000 Division where he worked on the AIX integration of the IBM Network Station. He has a BS in Computer Science and Mathematics from Purdue University.

XML—Coming to an Application Near You

By Greg Flurry

What is XML? How does XML relate to HTML? What are DTD, XSL, OSD, CDF, WIDL, and XLL? How does XML fit into the future of network computing? This article answers these questions and more.

The eXtensible Markup Language (XML) is a new kind of markup language from the World Wide Web Consortium (W3C). It represents a much simplified subset of Standard Generalized Markup Language (SGML) ISO 8897, a standard for defining documents used for the last decade. The W3C also defines HyperText Markup Language (HTML). Since HTML is also derived from SGML, how do XML and HTML differ?

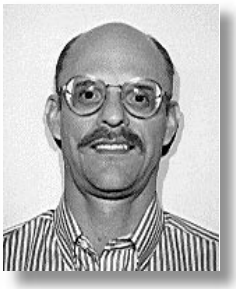
HTML, a key part of today's network computing, is a simple markup language for describing the format of small and simple documents containing text and other media. HTML, as defined by the W3C, uses a fixed vocabulary of tags to describe the document or data. Although a fixed set of tags make it easier to apply HTML, they also limit extensibility, the ability to support rich information structures, and the ability to determine syntactic and structural validity.

To address these issues, the W3C designed XML to fill a much broader role, which is evident in the design goals from the XML specification (see *http://www.w3.org/TR/REC-xml* or *http://www.xml.com/axml/testaxml.htm*):

- XML should be straightforward and usable on the Internet.
- XML should support a wide variety of applications.
- XML should be compatible with SGML.
- Programs that process XML documents should be easy to write.
- The number of optional features in XML should be kept to the absolute minimum.
- The design of XML should be formal and concise.
- XML documents should be easy to create.

To meet these goals, the W3C defined XML as a *meta-language*—a language used to describe other languages. As such, XML allows the definition of custom vocabularies to describe the abstract structure of a document or data.

XML also overcomes the three main limitations of HTML. First, XML supports extensibility. The ability to define custom tags enables its use in a much wider range of applications and with a broad range of clients including RS/6000s and PCs, and servers, such as RS/6000®, AS/400®, S/390®, Netfinity®, and others. Second, XML supports complex data structures that can be nested to any level of complexity. In addition, these structures can be efficiently created, consumed, manipulated, searched, and interchanged. Finally, XML, because of its



Greg Flurry

```
<H3>Moe Howard</H3>
<UL>
  <LI>address</LI>
    <ADDRESS>1212 Nevada
Street</ADDRESS>
    <ADDRESS>Podunk, CA</ADDRESS>
  <LI>phones</LI>
    <UL>
      <LI>555-555-5555</LI>
      <LI>555-555-4444</LI>
    </UL>
</UL>
```

Figure 1. HTML address book entry example

```
<person first="Moe" last="Howard">
  <address>
   <street>1212 Nevada
     Street</street>
   <city>Podunk</city>
   <state>CA</state>
  </address>
  <phones>
   <work>555-555-5555</work>
   <home>555-555-4444</home>
  </phones>
</person>
```

Figure 2. XML address book entry example

crisp rules for defining documents, supports the validation of a document's syntax and structure. These three enhancements have created much excitement about XML. In short, XML can do almost anything HTML can do, plus much more.

The next section presents a simple example that clarifies some XML advantages. Later sections describe how XML can define additional document types for new or existing applications, some of the current or proposed uses for XML, and activities and tools required for complete solutions.

An HTML versus XML Address Book

Figure 1 shows an example HTML used to describe an entry for an address book that might be found in Lotus Notes® or on an IBM WorkPad™. Figure 2 shows the XML equivalent.

Both HTML and XML use tags for markup. Tags in the HTML version can only identify the expected format. More abstract information, such as the person's state, typically would not be part of the tag, but would be part of the content of the tag. This abstract information could also perhaps be assumed from positioning within the `<h3>` tag. The tags in the XML markup, however, clearly identify the pieces of the address.

Consider which example—HTML or XML—is easier to process. Suppose an application needed to identify all the people from California within an address book. The application in the HTML implementation must search the entire entry, because only de facto rules exist about the structure of an entry. Additionally, there is no way to guarantee that Moe Howard is from Nevada or California, since both are valid states. The XML implementation clearly identifies the address and the state, making the search much easier and more accurate.

XML Advantages

XML has significant advantages for today's network computing applications, given the possibility of limited bandwidth and clients of variable capabilities. Assume that an XML-based address book database is stored on a server, such as an RS/6000 (see Figure 2). Then suppose the owner wants to access the address book from a wired network-attached PC, from a wireless network-attached device such as an IBM WorkPad, from an intelligent telephone and a regular (dumb) telephone. See Figure 3.

The server can send the same data to the first three devices. A browser on the PC can render the XML directly to Windows™ display orders, or convert first to HTML to leverage the existing browser HTML renderer. The WorkPad can convert the XML directly to its own unique display orders. An intelligent telephone can convert the XML to speech. For the regular telephone, the server itself, or more likely some intermediary, can convert the XML to speech.

The key point in this example is that the content is the same in all cases; only the rendering is different. XML's ability to

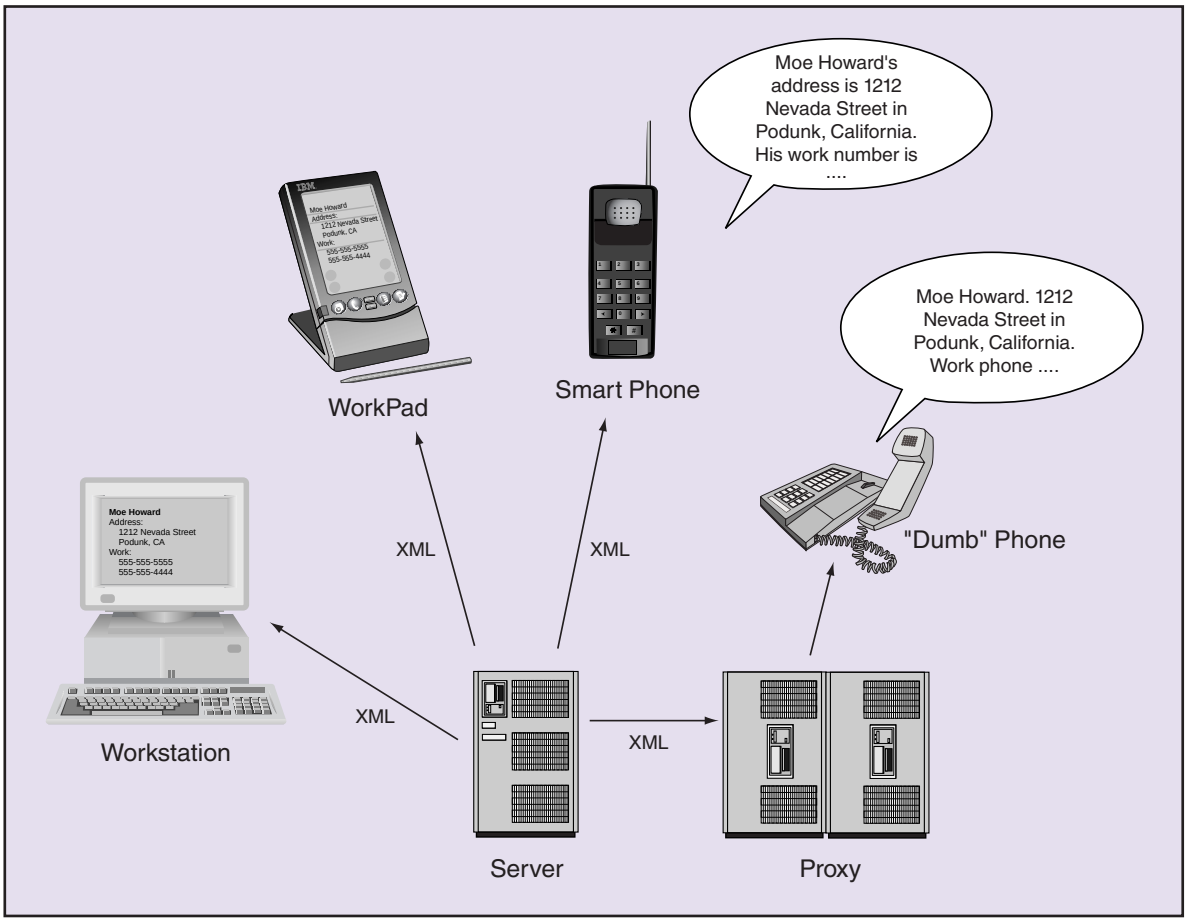


Figure 3. Accessing the address book

explicitly define the structure of the address book entry makes this possible.

Using XML for New Applications

The question arises about how to use XML in a new application. The first step is to define a new XML dialect or document type.

A new XML document type is defined using yet another concept from SGML called the Document Type Definition (DTD). The DTD defines the vocabulary and grammar for an XML-compliant document, that is, the set of tags that can appear in a document and how the tags relate to each other. This aspect of XML provides the extensibility and the ability to describe complex data structures. The DTD and additional rules defining XML also provide XML's ability to validate or prove the "correctness" of XML document syntax and structure.

Formally, an XML document contains a prolog and a body. The prolog contains XML version and character encoding information, the DTD, and miscellaneous markup; all parts of the prolog are optional. The body contains elements, entity references, and miscellaneous markup.

An XML document has both a logical and a physical structure. Elements, which describe the valid tags in a document, define the logical structure. Elements also have attributes that provide additional information about the elements. Entities define the physical structure of an XML document.

The DTD contains declarations that define the elements, element attributes, and entities. The DTD also contains notation declarations, conditional section declarations, and miscellaneous markup. Finally, miscellaneous markup, which is optional no matter where it appears, consists of com-

ments and processing instructions.

Parts of an XML Document

This section offers additional definitions and examples of the various XML document parts.

Element type declarations: Contain the name of the element type and a content specification for the data that can appear within that element. When used for markup, the elements are identified by start and end *tags*. For instance, to define the address element in the above example that contains street, city, and state elements, use the following:

```
<!ELEMENT address (street, city, state)>
```

Attribute declarations: Declared by associating the element name with an attribute list. Each attribute consists of an attribute name, attribute type, and default value. For example, the "person" element from Figure 4 has attributes "first" and "last" defining the name.

Elements and attributes: Used as described above:

```
<!ATTLIST person
  first CDATA #REQUIRED
  last  CDATA #REQUIRED>
```

Figure 4. Element attributes

```
<person first="Moe" last="Howard" ...
</person>
```

Entities: Provide "macro" type capability for XML. A full description of entities is beyond the scope of this article, but a few examples give a hint of their usefulness. For example, suppose you want to use the phrase "Howard, Howard, Fine & Sons" in many places within a document. The document could contain the following internal general entity declaration:

```
<!ENTITY theCompany "Howard, Howard,
Fine & Sons">
```

A document could use "`&theCompany;`" as a representation for the phrase. Next, suppose that a DTD contains several element type declarations that have the same content specification. It is possible to define an internal parameter entity as follows:

```
<!ENTITY  % genContentSpec "#PCDATA |
link | image">
```

and use it as follows:

```
<!ENTITY xyz (%genContentSpec)>
<!ENTITY mno (%genContentSpec)>
```

A final example shows additional flexibility of XML. Entities also can define the physical structure of a document. For example, an external general entity might point to some set of boilerplate for a business that is used in many documents; therefore it is kept in an external file. The entity declaration is shown in Figure 5 and the entity is referenced using "`&boilerplate;`".

```
<!ENTITY boilerPlate
      SYSTEM
"http://www.hhfs.com/internal/boilerplate.xml">
```

Figure 5. Declaration referencing frequently used entity

Notation declarations: Identify non-XML content, which allows XML parsers or applications that cannot handle such content to pass it to the appropriate processor.

Conditional sections: Allow XML parsers to include or ignore such sections depending on a fixed keyword or a parameter.

Processing instructions: Identify directions and information for XML parsers and applications. Processing instructions are not considered part of a document, but are passed on to an application identified in the declaration. The following is an example:

```
<?XML> version= "1.0"?>
```

Comments: Intended primarily as notes for a person to read and not considered part of a document. Comments are declared as follows:

```
<!– this is a comment –>
```

XML documents come in two major flavors: well-formed and valid. A well-formed XML document has the following characteristics that make the document easy to process:

- All beginning tags have matching end tags; for example, <person> ... </person> are properly nested; that is, <person> ... <address>...</address>... </person>, not <person> ... <address>... </person>...</address>.
- Tags with no content use the special XML syntax; for example, a page break could be indicated with a <pagebreak/> tag.
- All attributes are correctly quoted as in the <person first="Moe"> tag.
- All entities are declared.

```
<?xml version="1.0"?>
<!– address book –>
<!DOCTYPE addressbook SYSTEM
"addressbook.dtd">
<addressbook>
<person first="Moe" last="Howard">
  <address>
    <street>1212 Nevada
Street</street>
    <city>Podunk</city>
    <state>CA</state>
  </address>
  <phones>
    <work>555-555-5555</work>
    <home>555-555-4444</home>
  </phones>
</person>

<!– here could be other persons –>
</addressbook>
```

Figure 6. A valid address book example

Being well-formed implies nothing about having a DTD. It simply means that the document is syntactically correct. For example, the <person> element might require both a first and last parameter, but that could not be detected in a well-formed document. A valid XML document does contain or reference a DTD and complies with all the constraints expressed by the declarations in the DTD. XML allows a document to contain a DTD in a prolog, or reference an external DTD or DTDs. Using a set of DTDs, an XML parser or application can determine if a document is not only well-formed, but also logically correct, that is, valid.

Figure 6 represents a more complete example of the address book from Figure 2. The example above was well-formed, but not valid; the version in Figure 6 is valid.

The XML document in Figure 6 indicates that the document uses XML Version 1.0 and references a DTD in the file called addressbook.dtd. The rest of the markup is the same as in Figure 2. Figure 7 shows the DTD for the address book document type.

The address book DTD says that the <addressbook> element consists of one or more (+) <person> elements. The attributes of the <person> are the first and last name. Each <person> element contains one

```
<?xml encoding="US-ASCII"?>
<!– dtd for the address book –>
<!ELEMENT addressbook (person)+>
<!ELEMENT person (address, phones)>
<!ATTLIST person
  first CDATA #REQUIRED
  last  CDATA #REQUIRED>
<!ELEMENT address (street, city,
state)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT state (#PCDATA)>
<!ELEMENT phones (work*,home*)>
  <!ELEMENT work (#PCDATA)>
  <!ELEMENT home (#PCDATA)>
```

Figure 7. DTD for the address book

<address> and one <phones> element. An <address> element contains one each of <street>, <city>, and <state> elements. The <phones> element contains zero or more (*) <work> phone numbers and zero or more <home> phone numbers.

For additional details about XML documents, see the XML 1.0 specification by visiting the following Web sites: *http://www.w3.org/TR/REC-xml* or *http://www.xml.com/axml/testaxml.htm*.

Uses of XML in Applications

XML is useful and relevant in many applications. More than 20 XML dialects have been proposed or are currently in use, and the number is growing daily. The following list represents some selected uses of XML:

1. The Channel Definition Format (CDF), created by Microsoft® in partnership with DataChannel™, describes Web-based "channels" used for application and data distribution. CDF is built into Internet Explorer 4.0. See *http://www.w3.org/Submission/1997/2/*.

2. The Open Software Description (OSD), related to CDF, was defined by Marimba™ and Microsoft. The objective of OSD is to describe software packages in a way that makes it possible to distribute applications on a timely basis with a minimum of user involvement. See *http://www.w3.org/TR/NOTE-OSD*.

3. The Chemical Markup Language (CML) and the Mathematical Markup Language (MML) have proved effective at representing the arcane symbols, complex characters, and demanding markup that chemists and mathematicians require to disseminate important ideas. See *http://www.venus.co.uk/omf/cml/doc/* and *http://www.w3.org/TR/WD-math/*.

4. The Resource Description Framework (RDF) is a foundation for processing metadata that describes different types of Web resources. RDF represents the W3C's attempt to unify three separate, but similar, XML efforts:
   - W3C's work on describing data using the Platform for Internet Content Selection (PICS) dialect
   - Netscape's™ Meta Content Framework (MCF) for describing Web pages
   - Microsoft's XML-Data, intended for representing schema

   See *http://www.w3.org/RDF/Overview.html*.

5. The Extensible Linking Language (XLL) defines a rich syntax and semantics for hyperlinks. XLL promises to expand the capabilities inherent in hypertext well above and beyond current HTML implementations. It defines mechanisms such as bi-directional links, multi-way links, aggregate links (for multiple sources), and link types (links with attributes). See *http://www.w3.org/TR/WD-xml-link*.

6. The Synchronized Multimedia Interface Definition Language (SMIL) is a recent effort intended to help content developers better synchronize media for Web delivery, without having to depend upon specialized and expensive tools. See *http://www.whatis.com/smil.htm*.

7. Open Financial Exchange (OFX), a collaborative effort among Microsoft, Intuit®, and CheckFree®, is intended "for the electronic exchange of financial data between financial institutions, business, and consumers." Applications such as Microsoft Money and Intuit Quicken® use OFX. Although currently SGML based, the collaborators are converting OFX to XML so XML-capable Web browsers can use it. See *http://www.ofx.net/ofx/default.asp*.

8. OpenTag™ from the International Language Engineering® Corporation (ILE®) is designed to permit a single XML document to deliver content in different languages. International companies wanting to conduct e-business outside the English-speaking world obviously have a great interest in OpenTag. See *http://www.opentag.org/otdownld.htm*.

9. The Web Interface Definition Language (WIDL) from webMethods, but submitted to the W3C, is targeted at Web automation. For example, WIDL helps integrate information retrieved from the Web and disseminate that information throughout all business processes of an enterprise. See *http://www.webmethods.com*.

10. The Wireless Markup Language (WML) is part of the Wireless Application Protocol. WML is intended for use with narrow band devices, such as cellular phones and pagers, to specify content and user interface. See *http://www.wapforum.org/docs/technical/wml-30-apr-98.pdf*.

11. The Real Estate Listing Markup Language (RELML) is a relatively new dialect from OpenMLS and 4thWORLD Telecom. The purpose of RELML is to describe real estate listings that can be distributed across the Web, but still allow searching of the distributed listings that match a customer's criteria. See *http://www.xml.com/xml/pub/98/08/real/tech.html* and *http://www.4thworldtele.com/public/Client.html*.

You can find additional information about XML dialects and XML in general by visiting *http://www.xmlinfo.com* and *http://www.xml.com* on the Web.

What More is Needed?

The definition of an XML document does not describe how a document might be formatted when it is either displayed, printed, or otherwise rendered. Today, most active developments use the Cascading Style Sheets (CSS1 and CSS2) defined by the W3C and used with HTML documents.

Efforts are under way to define the Extensible Style Language (XSL) as a more dynamic and powerful notation for defining document style. Just as XML is derived from SGML, the work on XSL is derived, in large part, from the Document Style, Semantics, and Specification Language (DSSSL) used in the SGML community. See *http://www.w3.org/Submission/1997/13/ArborText*.

An application must be able to process the content of an XML document, otherwise it has no value. Each application can process a document in its own unique way. However, XML eliminates this chaos. The rules for the construction of well-formed and valid XML documents bring a regularity to such documents. In recognition of this, the W3C is developing the Document Object Model (DOM), "a platform- and language-neutral program interface that will allow programs and scripts to access every element in a document and update the content and structure of documents in the standard way." See *http://www.w3.org/DOM*.

XML parsers are one of the key targets for DOM. Parsers are tools that can determine whether an XML document is valid or well-formed, and produce normalized versions of the document for easier processing by applications. A good example of such a parser is XML4J, a Java™-based parser from IBM. Since XML4J is written in Java, it can run without modification on AIX® as well as other operating systems. XML4J is freely available on the IBM alphaWorks™ Web site. See *http://www.alphaWorks.ibm.com/formula.nsf/system/technologies/7BC35F3E4E69996A882565A700035C56*.

XML Tools

One strong indicator of an important technology is tools for using the technology. XML tools exist and more are coming. The major Web browsers (Netscape Communicator 5.0 and Microsoft Internet Explorer 4.0) support or will support various uses of XML. Because of its SGML heritage, XML can be handled by hundreds of existing SGML tools, such as GRIF Symposia and SP. Some existing authoring tools like FrontPage and HomeSite also support XML, and others are expected to incorporate support rapidly. For information about XML tools, see *http://www.sil.org/sgml/*, *http://www.infotek.no/sgmltool/guide.html*, *http://www.infotek.no/sgmltolguide.htm* and *http://www.xmlinfo.com*.

What Does It All Mean?

From the beginning, XML was designed to serve a wide range of network computing

applications. XML certainly overcomes the disadvantages of HTML. The sample of XML dialects above show that XML can serve in almost any application, extending far beyond HTML, in roles such as defining complex content and facilitating processing of that content. XML can provide an answer for just about any question concerning document or data processing in network computing applications.

The companies promoting some aspect of XML (IBM, Netscape®, Sun®, Microsoft, and many others) guarantee that XML will have an excellent chance of playing a role in virtually all new network computing applications. While XML will not necessarily replace HTML, XML's power, flexibility, extensibility, and openness should make it a major factor in the network computing environment of the next decade. XML will come to an application near you—and soon.

Additional References

Leventhal, Michael; Lewis, David; and Fuchs, Matthew. *Designing XML Internet Applications*. Upper Saddle River, NJ: Prentice-Hall, Inc. 1998. ISBN 0-13-616822-1.

Megginson, David. *Structuring XML Documents*. Upper Saddle River, NJ: Prentice-Hall, Inc. 1998. ISBN 0-13-642299-3.

Tittle, Ed; Mikula, Norbert; and Chandak, Ramesh. *XML for Dummies*; Foster City, CA: IDG Books Worldwide, Inc. ISBN 0-7645-0360-X.

Greg Flurry, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Flurry is a senior technical staff member in the Server Group Division. His responsibilities, as part of the Systems Architecture & Technical Strategy team, include network computing and Java. He has a BS in Electrical Engineering from Vanderbilt University and an MS in the same field from the University of Kentucky.

Xprofiler—Parallel Profiling Tool

By Jhy-Chun Wang, Xianneng Shen, and Ted Hoover

This is the first article in a series about the application development environment (Parallel Environment) of the IBM RS/6000 SP system. This article introduces Xprofiler, a profiling tool based on X-Windows that provides profiling at both a procedure and a source statement level. It combines graphical and textual displays of function call trees to reveal an application's performance profiling data.

Profiling the performance of an application normally involves counting the CPU time used by each procedure in the application and the number of times each procedure was called by other procedures within the application. Profiling is the first step in understanding an application's performance behavior: which procedures used the bulk of the CPU time or which procedures were called the greatest number of times. Once you identify these heavyweight procedures, you can focus your efforts on those procedures to improve the application's overall performance.

Xprofiler Characteristics

Xprofiler, a profiling tool based on X-Windows, has the capability of profiling at both procedure level and source statement level. Procedure-level profiling reveals the CPU time consumed by each procedure and the number of times each procedure was called by other procedures in the application. Source statement level profiling reveals the CPU time consumed by each source statement in a procedure. The procedure has to be compiled with the `-g` option in order to map profiling data to source statement level.

Xprofiler's graphical function call tree shows the CPU usage of each function within the application. The nodes in the tree represent program functions. The width of the node is proportional to the CPU time spent in the function and its descendant functions. The height of the node is proportional to the CPU time spent in the function itself.

Using this presentation scheme, a function spending little CPU time in its descendant functions will have a square shape; a node spending a significant amount of CPU time in its descendant functions (for example, the `main()` in C programs) will have a rectangular shape. In the graphical display, the larger a node, the more CPU time was consumed by the function (and/or its descendant functions). Xprofiler provides a link between the call tree structure and the source statements. The source code associated with a node in the call tree can be located using point-and-click.

The graphical user interface of the Xprofiler enables you to navigate between functions of the application and the system calls used in your application. You can zoom in and out of the main graphical display,

placing functions into different groups according to their location in files, and so on. This can simplify the main display or reveal the execution relationship of functions in different object files.

Xprofiler handles textual profiling reports well and provides statistical reports for parallel applications, such as the average time spent performing a function, tasks with minimum and maximum time in a function, and so on. Xprofiler will work on both the RS/6000 SP™ and the stand-alone RS/6000® server. It is packaged in the SP Parallel Environment product and currently available only within the parallel environment on the RS/6000 SP.

Xprofiler is similar to the AIX® `gprof` command in analyzing the profiling data and merging multiple output profiling files of a parallel application. Xprofiler can analyze an application's performance profiling data not only at the procedure level like `gprof`, but also at the source statement level. The `gprof` command presents the profiling information via plain text reports, which are not intuitive and often difficult to analyze if you use many functions/processors in an application. Xprofiler has a comprehensive graphical user interface that provides flexibility in navigating the profiling data.

Xprofiler works for both serial and parallel applications. When compiled with the proper compiler options, a sequential application generates only one profile data file (that is, the `gmon.out` file), while a parallel application will generate one profile data file for each task in the application. Once the profile data files are collected, Xprofiler can analyze the result. Xprofiler requires AIX 4.2.1 or higher to run on the system.

How to Use Xprofiler

To use Xprofiler, compile and link an application with the `-pg` compiler option, which is required for either sequential or parallel applications.[1]

The application generates the CPU usage and call count information when it executes and writes the data into one or more `gmon.out` files.

Xprofiler Features

Xprofiler not only supports all the functions and command-line options provided by `gprof`, but it also includes many functions that make analyzing an application's performance effective and seamless.

In addition to `gprof` functions, Xprofiler provides the following functions to help users navigate an application's performance profile data:

- Function call tree graphical display: Using functions as nodes and the caller-callee relationship as arcs, Xprofiler constructs a function call tree from an application's `gmon.out` files to represent the application's execution structure.

> ***Xprofiler can analyze an application's performance profiling data not only at the procedure level, but also at the source statement level.***

- Configuration file: With the configuration file, users can save one (customized) function call tree structure to be used in future Xprofiler sessions.
- Alter file search path: Users can specify the paths and search order in which a source code file or a library object file will be located.
- Overview window: This highlight area makes it easy to manipulate the graphical display.
- Summary mode/average mode: This is available when more than one `gmon.out` file is entered. This feature uses a different node presentation scheme in average mode to reveal

[1] *IBM Parallel Environment for AIX: Operation and Use,* Volume 2, Tools Reference. SC28-1980. Page 94.

```
        // The application must be compiled with the -pg option.
        // -g is required for source statement profiling. It is optional.
        // -O is required for compile optimization. It is optional.
xlC -O -g -pg -o xprof_1 xprof_1.C

        // Run the application. A gmon.out file will be generated.
xprof_1

        // bring up Xprofiler to analyze the profiling data.
xprofiler xprof_1 gmon.out
```

Figure 1. Profiling an application with Xprofiler

the load-balancing problem in a parallel application.

- Reconstruct function call tree: Users can remove nodes and/or arcs from the function call tree or add previously removed nodes/arcs back into the tree to customize or simplify the graphical display to focus on a particular area, such as the functions on a CPU usage critical path.
- Group functions into load units: Functions can be grouped into load units (that is, object files) to reveal the execution relationship between units and to simplify the graphical display.

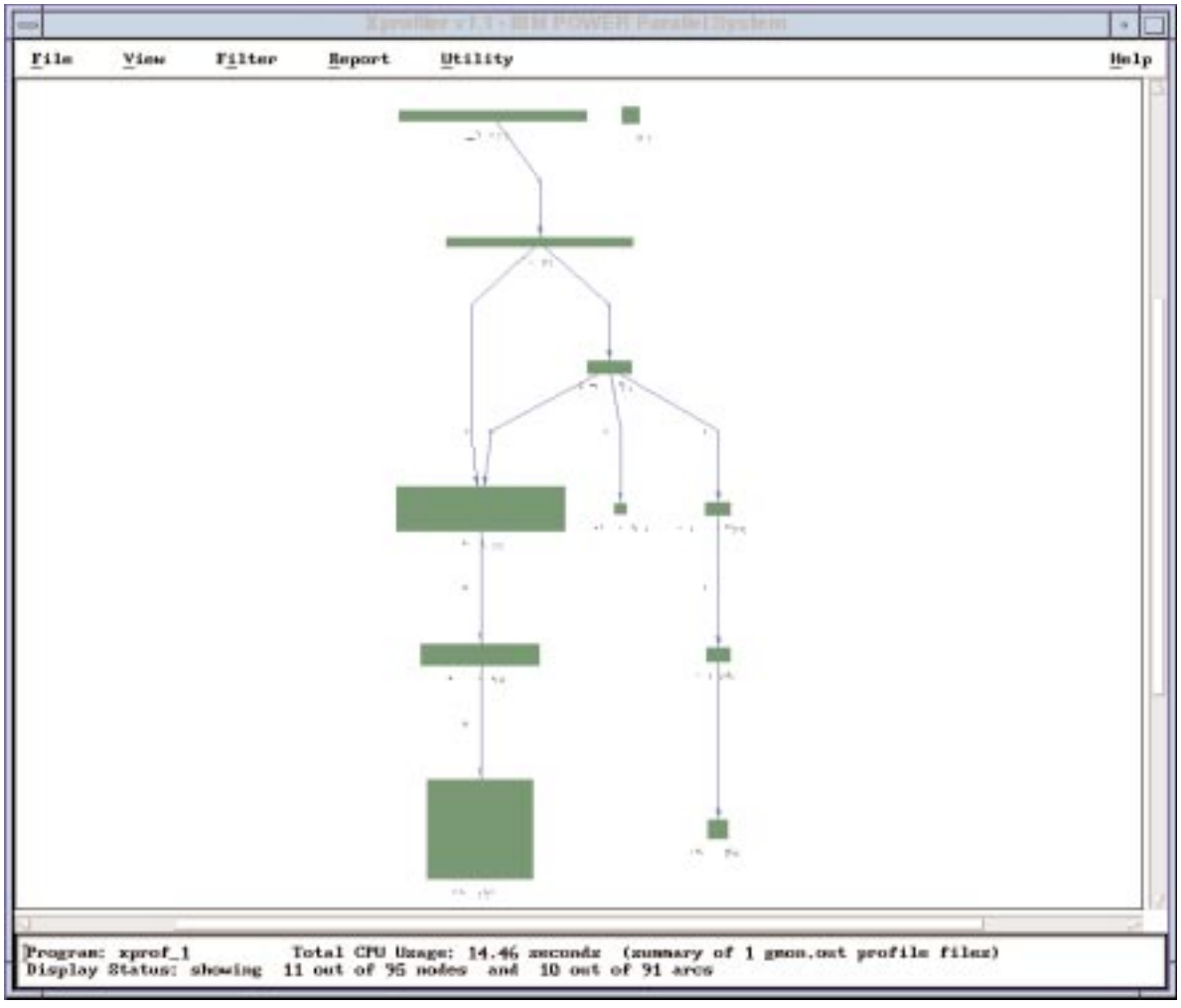


Figure 2. Xprofiler main graphical display

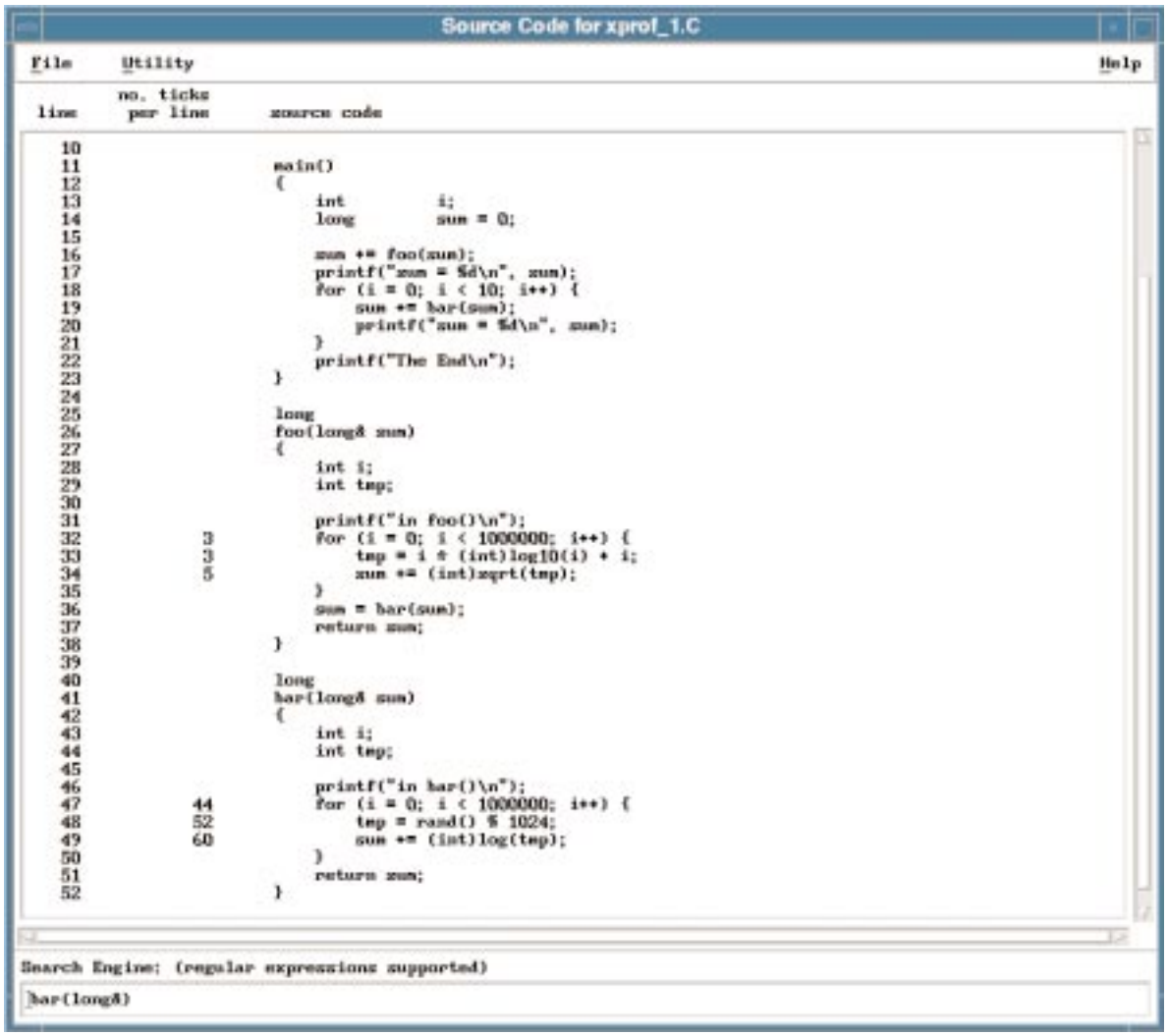


Figure 3. Xprofiler source code display

- Filter functions: The function call tree can be filtered by function names, CPU usage, or function call count information. These filter functions make it easier to reconstruct a function call tree.
- Textual reports: In addition to the `gprof` profiling reports, Xprofiler provides reports on the call count summary and load units' statistics data.
- Source code display: Profiling data (that is, CPU usage) can be mapped to source code by a point-and-click on the graphical display.
- Disassembler code display: Profiling data (CPU usage) can be mapped to disassembler code by a point-and-click on the graphical display.

Xprofiler supports computer languages such as Fortran 77, Fortran 90, High Performance Fortran, C, and C++.

Xprofiler Examples

Figure 1 shows an example program with three functions: `main()`, `foo()`, and `bar()`. If the source code is in the `xprof_1.C` file, execute the steps in Figure 1 to analyze the application using Xprofiler.

Figure 2 shows a simplified function call tree of the application `xprof_1`, where many system calls were removed from the display. The display employs a presentation principle whereby the larger a node is,

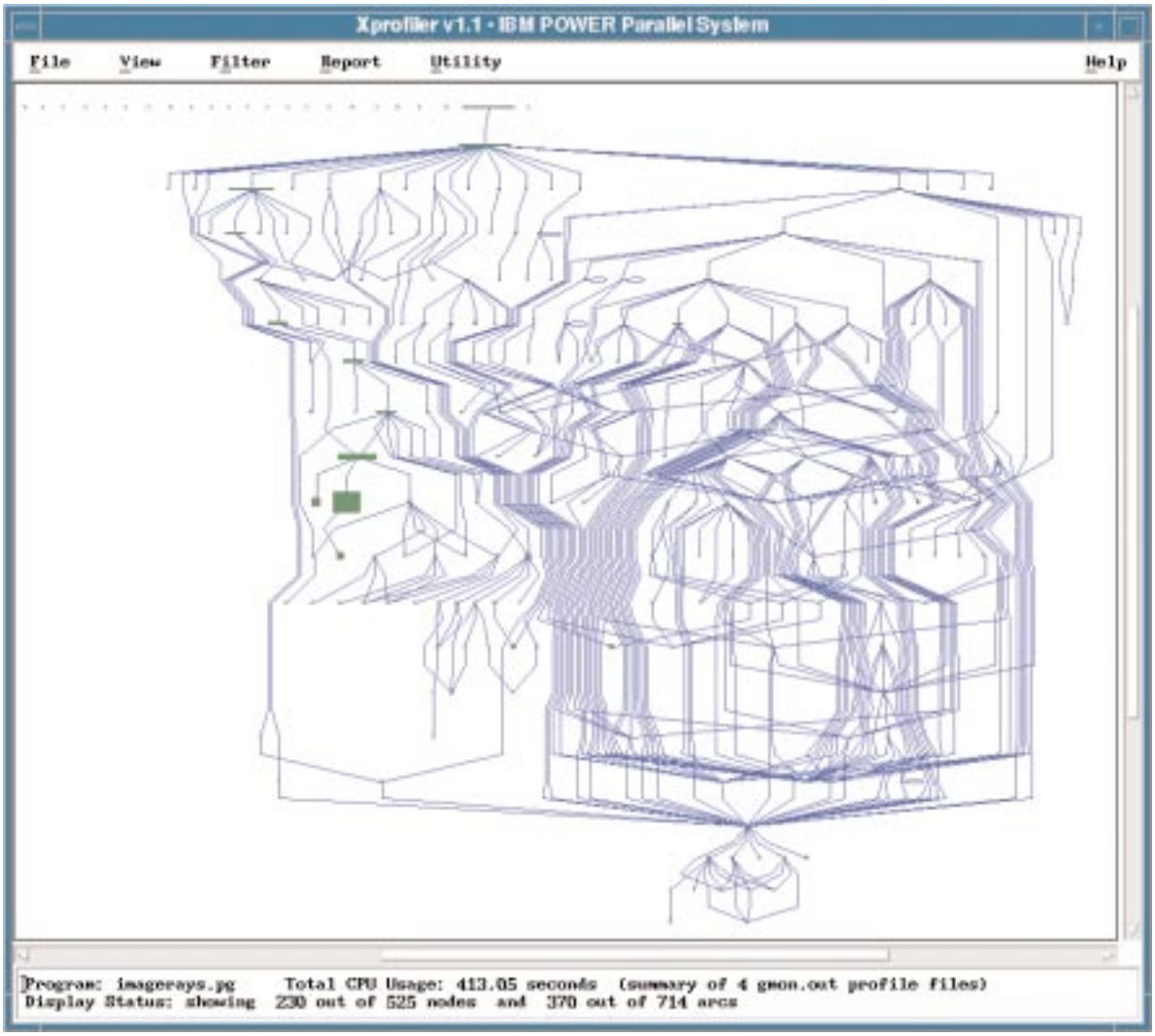


Figure 4. Xprofiler main graphical display of a large application

the more CPU time is spent by the node (remembering that each node represents a function in the application, including the system calls).

In this application, `function bar()` (at the bottom left area) used most of the CPU time. By clicking on the `node bar()` and selecting the source code display, the source code file that contains `function bar()` will be brought into the source code display.

Figure 3 shows the source code display of the file `xprof_1.C`, the second column (the number-of-ticks-per-line) in the display represents the number of ticks (on AIX, one tick equals 10 milliseconds) that occurred at each source statement. The statements that consumed more CPU time will have larger numbers of ticks mapped to them.

In Figure 3, source code lines 47 to 49 consumed 93% (156 ticks out of the total of 167 ticks) of the application's CPU usage; that is, they are the hot spot of the application. By combining Figures 2 and 3, you can easily locate the function that spent the most CPU time, then click on the node to bring up the function's source code file. The number of ticks mapped to each source statement will reveal where the bulk of CPU time was spent in a function. Once a CPU hot spot is located, you can devote some effort in that area to improve the application's performance.

Figure 4 shows the partial function call tree of a large application, which uses 525 functions, including system calls. Although the display seems complicated, a large node

on the center left area clearly stands out. It represents a function that consumed a lot of CPU time, and is therefore a good candidate for performance tuning.

Conclusion

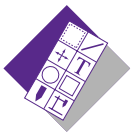
Performance profiling is an important step in performance tuning. Profiling data can reveal functions in an application that consume large amounts of CPU usage or call counts.

Xprofiler is superior to traditional command-line profile tools like `gprof`. It not only provides all the functions that `gprof` supports, but it also combines a graphical display with a source code browser and textual reports to provide a user-friendly environment for performance profiling on both sequential and parallel applications.

References

IBM Parallel Environment for AIX: Hitchhiker's Guide. GC23-3859.

IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference. SC28-1980.



***Jhy-Chun Wang**, IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: wangjc@us.ibm.com. Dr. Wang is an advisory software engineer in the RS/6000 SP Group. His current interests include performance analysis and portable high-performance computing tools. Dr. Wang holds a PhD in Computer Science from Syracuse University.*

***Xianneng Shen**, IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: xshen@us.ibm.com. Dr. Shen is a programming consultant in the RS/6000 SP Teraplex Integration Center. His current interests include scalable parallel algorithms, scalable I/O, parallel database, data warehouse, and data mining implementations. Dr. Shen holds a PhD in Electrical Engineering from Syracuse University and an MS in Computer Engineering from Syracuse University.*

***Ted Hoover**, IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: hoov@us.ibm.com. Mr. Hoover is an advisory software engineer and the team lead for the Application Development Tools team in the RS/6000 SP Group. He has an MS in Computer Science from Rensselaer Polytechnic Institute and a BS in Computer Science from the University of Pittsburgh at Johnstown.*

AIX Questions

Compiled by Jeff Simon

The AIX Solution Provider Technical Support Group in Austin, Texas, supports software vendors who are developing or porting applications to AIX. This article is a compilation of questions that are frequently asked by vendors. The name of the responding Technical Support Group staff member appears after each response.

How can I determine which shell I am using?

One option, `chsh`, which is shipped with `bos.rte.security`, shows the shells installed on your system and provides an option to switch to another shell. Alternately, if your shell has not changed since the initial login, you can use the following:

```
grep <userid> /etc/passwd
```

which provides output similar to

```
jsimon:!:200:1::/u/jsimon:/usr/bin/ksh
```

—Jeff Simon

Which AIX® signals are thread-specific and use signal handlers?

The following synchronous signals are thread-specific and include signal handlers: `SIGILL`, `SIGTRAP`, `SIGARBT`, `SIGEMT`, `SIGFPE`, `SIGBUS`, `SIGSYS`, and `SIGSEGV`.

—Jeff Simon

Can pthread_exit() be called from within a signal handler?

A signal handler cannot call any `pthread` primitive, including `pthread_exit`.

—Jeff Simon

Are sigsetjmp() and siglongjmp() thread safe?

Yes, `sigsetjmp` and `siglongjmp` are signal- and thread-safe.

—Jeff Simon

Jeff Simon

How can I identify a 64-bit enabled RS/6000®?

If the following line appears in `/etc/inittab`, you are running on a true 64-bit enabled machine:

```
load64bit:2:once:/etc/methods/cfg64
```

—Wade Carlin

Are there any known performance problems using Java™ 1.1.4 with BOS 4.3.1?

It is necessary to apply APAR IX77074 to achieve top performance from the Java Development Kit (JDK) 1.1.4 on AIX 4.3.1.

—Jeff Simon

Which system calls are available to all kernel extensions?

Figure 1 shows a list of the system calls and describes the function of each. For additional information, see *AIX Kernel Extensions and Device Support Programming Concepts* (SC23-2611) or InfoExplorer™.

—Wade Carlin

How do I make AIX system calls from a Java program using Java Native Interface (JNI)?

The code example in Figures 2 through 5 demonstrates this procedure and includes the following filesets:

- Makefile
- JavaCrypt.c
- JavaCrypt.java
- JavaCrypt.exp

—Hung Dinh and David Carew

System Call	Function
gethostid	Gets the unique identifier of the current host
getpgrp	Gets the process ID, process group ID, and parent process ID
getppid	Gets the process ID, process group ID, and parent process ID
getpri	Returns the scheduling priority of a process
getpriority	Gets or sets the nice value
semget	Gets a set of semaphores
seteuid	Sets the process user IDs
setgid	Sets the process group IDs
sethostid	Sets the unique identifier of the current host
setpgid	Sets the process group IDs
setpgrp	Sets the process group IDs
setpri	Sets a process scheduling priority to a constant value
setpriority	Gets or sets the nice value
setreuid	Sets the process user IDs
setsid	Creates a session and sets the process group ID
setuid	Sets the process user IDs
ulimit	Sets and gets user limits
umask	Sets and gets the value of the file creation mask

Figure 1. System calls for kernel extensions

```
/* Makefile */

# The JAVA_HOME variable should be set to the installation directory for
# your AIX JDK.
#
# note: The JAVA_HOME variable will need to point to the current
# JDK directory.
```

Figure 2. Makefile (continued)

Wade Carlin

```
JAVA_HOME=/usr/lpp/J1.1.1

libcrypt.so: JavaCrypt.o
        rm -f libcrypt.so
        ld -o libcrypt.so JavaCrypt.o -bnoentry -bM:SRE -bE:JavaCrypt.exp \
           -blibpath:/lib:/usr/lib -lc_r \
           -L${JAVA_HOME}/lib/aix/native_threads -ljava

JavaCrypt.class: JavaCrypt.java
        ${JAVA_HOME}/bin/javac JavaCrypt.java
        ${JAVA_HOME}/bin/javah -jni JavaCrypt

JavaCrypt.o: JavaCrypt.c JavaCrypt.class
        xlc_r -c -I. -I${JAVA_HOME}/include -I${JAVA_HOME}/include/aix \
              -o JavaCrypt.o JavaCrypt.c

clean:
        rm -f *.class *.o JavaCrypt.h libcrypt.so
```

Figure 2. Makefile

David Carew

Hung Dinh

```
/* JavaCrypt.c */

/*
This program demonstrates the ability to make an AIX system call (i.e., crypt)
from within a Java application.
*/

#include “JavaCrypt.h” /* Include file generated by javah command */

JNIEXPORT jstring JNICALL Java_JavaCrypt_getEncryptedPassword
  (JNIEnv *env, jobject obj, jstring jpassword, jstring jsalt)
{
  const char *password;   /* String to be encrypted */
  const char *salt;       /* The 2 character salt (see crypt) documentation */
                          /* for details                                    */
  jstring encrypted;

  /* convert Java strings to C strings */
  password = (*env)->GetStringUTFChars(env, jpassword, 0);
  salt = (*env)->GetStringUTFChars(env, jsalt, 0);

  /* Encrypt the string */
  encrypted =  (*env)->NewStringUTF(env, (const char *) crypt(password, salt));

  /* tell Java we've finished with the strings */
  (*env)->ReleaseStringUTFChars(env, jsalt, salt);
  (*env)->ReleaseStringUTFChars(env, jpassword, password);

  /* Return results */
  return encrypted;

}
```

Figure 3. JavaCrypt.c

```
/* JavaCrypt.java */

/*
This class uses the JNI interface to call the AIX routine crypt. This technique
can be utilized to call any AIX system call from within a Java application.

Use the provided Makefile to compile this program.

It should be run as follows:

          java JavaCrypt “anystring”

Where “anystring” is the string to encrypt (enclosed in quotes if it contains
any whitespace characters).


*/

/**
  * This class uses the native crypt subroutine to encrypt a string.
  *
  * @version 1.0
  *
  */

public class JavaCrypt {

    /**
     * This constructor loads the library containing the native code
     *
     */

    public JavaCrypt() {
       System.loadLibrary(“crypt”);
    }

    /**
     * Calls the native routine to encrypt a string.
     *
     * @param String password - The string to be encrypted
     * @param String salt     - The salt used in the encryption.
     *                          Consult the crypt documentation
     *                          for more information about this parameter.
     *
     * @returns String - The encrypted string
     */

    public native String getEncryptedPassword(String password, String salt);

    /*
     * Used to test the JavaCrypt class
     *
     */
    public static void main (String [] args) {
        if (args.length != 1) {
           System.out.println(“Usage: java JavaCrypt \”String to encrypt\””);
           System.exit(1);
        }
```

Figure 4. JavaCrypt.java (continued)

```
        System.out.println("Encrypted string = " +
                           new JavaCrypt().getEncryptedPassword(args[0],
"dc"));
    }

}
```

Figure 4. JavaCrypt.java

```
/* JavaCrypt.exp */

Java_JavaCrypt_getEncryptedPassword


/* This program demonstrates the uses of the wait() system call and the
   WIF macros to determine the status from the child process  */

#include <sys/wait.h>
#include <stdlib.h>
main()
{
 int i,pid,status;
 pid = fork();
 if (pid == -1)
  {
   perror (" problems with fork child process ");
   exit(1);
  }
 if (pid ==0)   /* successfully forked  - child starts */
  {
  printf("child  does li -al    \n");
  execlp("li", "li", "-al",NULL);

 /*  The execlp subroutine searches each of the directories
     listed in the PATH environment variable for the 'li' command,
     and then it overlays the current process image with this command.
     The execlp subroutine is not returned unless 'li' command cannot
     be executed. */

  perror("  li command not found    \n");
  exit(2);
  }            /*  child finishes   */


  printf("parent does some tasks   \n");
  for (i=0;i<4;i++)
    {
     printf("parent sleeps for 1 sec \n");
     sleep(1);
     printf("parent wakes up \n");
    }

  printf("parent finishes the task   \n");
  printf("and child status is \n");
  wait(&status);
```

Figure 5. JavaCrypt.exp (continued)

```
/*
 The wait subroutine suspends the calling thread until the
 process receives a signal that is not blocked or ignored,
 or until any one of the calling process' child processes
 stops or terminates. The wait subroutine returns without
 waiting if the child process that has not been waited for
 has already stopped or terminated prior to the call.

 If the wait subroutine is unsuccessful, a value of -1 is
 returned and the errno global variable is set to indicate
 the error. In addition, the waitpid and wait3 subroutines
 return a value of 0 if there are no stopped or exited child
 processes, and the WNOHANG option was specified. The wait
 subroutine returns a 0 if there are no stopped or exited
 child processes, also.

*/

/*

 The value pointed to by StatusLocation when wait subroutines
 are returned, can be used as the ReturnedValue parameter for
 the following macros defined in the sys/wait.h file to get
 more information about the process and its child process.

 WIFSTOPPED
 Returns a nonzero value if status returned for a stopped child.

 WIFEXITED
 Returns a nonzero value if status returned for normal termination.

 WIFSIGNALED
 Returns a nonzero value if status returned for abnormal termination.

*/

if WIFSTOPPED(status)
  printf(" child stopped and stop sig was %d \n", WSTOPSIG(status));
else if WIFSIGNALED(status)
  printf("child was terminated and signal was %d\n", WTERMSIG(status));
else if WIFEXITED(status)
  printf("child ended ok and status was %d\n", WEXITSTATUS(status));

exit(0);

}
Describe mail filters on AIX.
```

Figure 5. JavaCrypt.exp

Describe mail filters on AIX.

Figure 6 demonstrates the use of mail filters on AIX. Figure 7 shows the message generated by the mail filter.

—David Carew and Hung Dinh


```
/*

 The program demonstrates the use of mail filters on AIX.


 To compile use the following:

    cc -o mailf mailf.c

 The program can be triggered by adding the following changes to the .forward
 file in your home directory.

    username, | mailf username

Where username is your login ID. The executable file should be in a directory
like /bin that is in the default path. Alternatively, you can add the fully
qualified path name to the executable file name in the .forward file.

note: A message file (mailf.msg) must be installed in the user's home
      directory (see below).

*/

#include <string.h>
#include <sys/errno.h>
#include <pwd.h>
#include <stdio.h>

extern struct passwd *_getpwnam_shadow(char *, int);

/*
**  MAILF — return a message to the sender of a mail message.
**
*/

main(int argc, char **argv)

{
   char msgf[128];
   char *username;
   char *sender;
   struct passwd *pw;
   char *getsender();

   /* Make sure the username is specified */
   if (argc != 2) {
               fprintf(stderr, "Unknown user %s", username);
               exit(EINVAL);
   }

   /* Get the username of the user who has installed the filter */
```

Figure 6. Mail filters on AIX (continued)

```
     username = argv[argc - 1];

         /* find user's home directory */
         pw = _getpwnam_shadow(username, 0);
         if (pw == NULL)
         {
                 fprintf(stderr, "Unknown user %s", username);

                exit(EINVAL);
         }

    /* Construct message file name */
         strcpy(msgf, pw->pw_dir);

         (void) strcat(msgf, "/mailf.msg");

    /* Get the sender of the mail */
    sender = getsender();

    /* Send reply to sender */
    sendreply(msgf, sender, username);
}


/*
**  GETSENDER -- read message from standard input and return sender
**
**      Parameters:
**              none.
**
**      Returns:
**              pointer to the sender address.
**
**      Side Effects:
**              Reads first line from standard input.
*/

char *
getsender()
{
        static char line[512];
        char *p;

        /* read for the line */
        if (fgets(line, sizeof line, stdin) == NULL ||
            strncmp(line, "From ", 5) != (int)NULL)
        {
                fprintf(stderr,"Invalid message format");
                exit(EINVAL); /* Invalid message format */
        }

        /* Parse out sender info and return it */

        p = strstr(&line[5], " ");
        if (p == NULL)
        {
           fprintf(stderr,"Invalid message format");
           exit(EINVAL); /* Invalid message format */
}
```

Figure 6. Mail filters on AIX (continued)

```
        *p = '\0';

        /* return the sender address */
        return (&line[5]);
}

/*
**  SENDREPLY - send a message to a particular user.
**
**      Parameters:
**              msgf - file name containing the message.
**              user - user who should receive it.
**    myname - sender of message
**
**      Returns:
**              none.
**
**      Side Effects:
**              sends mail to 'user' using /usr/sbin/sendmail.
*/
int sendreply(msgf, sender, user)
        char *msgf;
        char *user;
        char *sender;
{
        FILE *f;
        char newaddr[128];
   char *to;

   /* Verify that the destination address is in correct format */
   /* i.e., in the form <joe@mycompany.com>           */
   /* Fix it if it isn't.                             */
        if (sender[0] == '<' && sender[strlen(sender) - 1] == '>')
                to = sender;
        else {
                newaddr[0] = '<';
      newaddr[1] = '\0';
                strcat(newaddr,sender);
                strcat(newaddr, ">");
                to = newaddr;
        }

   /* find the message to send and replace stdin with this file */
   /* sendmail will read input of message from stdin            */
        f = freopen(msgf, "r", stdin);
        if (f == NULL)
        {
           fprintf(stderr, "No message to send");
      exit(EINVAL);
        }

   /* Execute sendmail */
        execl("/usr/sbin/sendmail", "sendmail", "-f", user, sender, NULL);

   /* If we get here then execl failed */
        fprintf(stderr, "Cannot exec /usr/sbin/sendmail") ;
   exit(ENOEXEC);
}
```

Figure 6. Mail filters on AIX (continued)

```
/* mailf.msg
**
** This file contains the replied message generated
** by the mail filter.
**
** The accompanying message file, mailf.msg, must be installed in the user's
** home directory. This file should be edited to include the correct e-mail
** address of the sender and the appropriate message.
**
*/

From: joe@mycompany.com (Joe User)
Subject:  Mail filter test

SYSTEM GENERATED MESSAGE - DO NOT REPLY -

Enter your message
```

Figure 7. Message generated by the mail filter

Compiled by Jeff Simon, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Simon has worked in technical support since 1990 and currently works as a technical lead and systems administrator for a series of technical Web sites. He has a BS in Computer Science from Southwest Texas State University.

Explore the Power of the RS/6000 SP Server

Solution Partnership Centers are a benefit of the IBM Solution Developer Program. The IBM Solution Developer Program provides marketing support to solution developers, helping them reach new customers and expand into new markets worldwide.

IBM

IBM Solution Partnership Center
2929 Campus Drive Suite 275
San Mateo, CA 94403
Phone: 650-524-5246
Fax: 650-524-5310
E-mail address: spcw@spc.ibm.com
Web: http://www.spc.ibm.com.

IBM Solution Partnership Center
404 Wyman Street – North Entrance
Waltham, MA 02154
Phone: 1-800-678-4249 or
781-895-2610

If you are a commercial software developer and you want to know how your application runs on the new RS/6000 SP large scale server, visit the IBM Solution Partnership Center (SPC). At the SPC, you can explore the power and performance of the SP server for transaction processing and discover how the new version of AIX provides a solid foundation for a variety of application—including applications for the world of e-business and business intelligence.

The SPCs offer private labs to commercial software developers for porting, enabling, and testing their applications. These labs are equipped with a wide range of tools and technologies from industry-leading hardware and software providers. While visiting a Solution Partnership Center, developers can work with their applications on various platforms without investing in any additional equipment or staff. The SPCs also offer ongoing technical seminars and hands-on workshops related to critical technology issues.

For additional information about the Solution Partnership Centers, visit the Web site: http://www.spc.ibm.com/

Information about the IBM Solution Developer Program is available at the Web site: http://www.developer.ibm.com/