November 1996 • $11.50
VOL. 2 NO. 11

INSIDE SOLARIS
Tips & techniques for users of SunSoft Solaris™

in this issue

An introduction to Java applets and applications

By Probal Shome

One reason for all the hoopla regarding Java is that it holds great promise—the promise of machine independence. You're supposed to be able to write an applet on one of many platforms (from a PC running Windows 95 to an IBM mainframe running MVS) and instantly achieve cross-platform portability of the applet to all platforms on which the Java Virtual Machine (JVM) is implemented.

In this article, we'll show you some of the basics of creating Java applets and applications. We assume that you have a working knowledge of C and object-oriented concepts, such as class, objects, methods, attributes, and inheritance.

We'll first build the traditional "Hello, World!" applet and present some variations. Next, we'll build a standalone Java application version of the "Hello, World!" program. Finally, we'll add the ability to display text that the user enters.

Installing Java

If you don't have Java installed, you'll have to download the Java Development Kit (the latest version at the time of this writing is v1.0.2) from *http://java.sun.com/java.sun.com/products/JDK/index.html*. Note to x86 users: Starting with JDK v1.0.2, Sun provides a version that runs on the Intel platform.

Quick introduction

Writing a Java applet is similar to using a third-party application toolbox with C++. Many of these provide a base class that manages the nuts-and-bolts details of an application, and you provide the code that makes the application do something interesting. Similarly, Java provides the base class Applet, which provides the basic details in telling the application how to open a window, redraw itself, and other general features. When you create a Java applet, you typically just extend the Applet class.

When you create a new class in Java, the new class must be in a file named *ClassName.java*, where *ClassName* is the name of the class you're creating. You can compile the *ClassName.java* file using `javac` to get the *ClassName.class* file, which holds the compiled JVM code. You can then run the code in *ClassName.class* with the Java interpreter, or if you have an HTML document that specifies it, you can view it with an HTML browser that supports Java, such as HotJava or Navigator. The JDK also comes with an applet viewer, which will execute Java applets specified in an HTML document.

INSIDE SOLARIS
Tips & techniques for users of SunSoft Solaris

Inside Solaris (ISSN 1081-3314) is published monthly by The Cobb Group.

Prices
U.S. ..... $115/yr ($11.50 each)
Outside U.S. ..... $135/yr ($16.95 each)

Phone and Fax
US toll free ..... (800) 223-8720
UK toll free ..... (0800) 961897
Local ..... (502) 493-3300
Customer Relations fax ..... (502) 491-8050
Editorial Department fax ..... (502) 491-4200
Editor-in-Chief ..... (502) 493-3204

Address
Send your tips, special requests, and other correspondence to:

The Editor, *Inside Solaris*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
Internet: inside_solaris@merlin.cobb.zd.com.

For subscriptions, fulfillment questions, and requests for group subscriptions, address your letters to:

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
Internet: cr@merlin.cobb.zd.com.

Staff
Editor-in-Chief ..... Marco C. Mason
Contributing Editors ..... Al Alexander
Probal Shome
Jerry Phillips
Production Artist ..... Liz Palmer
Editors ..... Linda Recktenwald
Karen S. Shields
Circulation Manager ..... Mike Schroeder
Editorial Director ..... Linda Baughman
VP/Publisher ..... Lou Armstrong
President ..... John A. Jenkins

Back Issues
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $11.50 each, $16.95 outside the US. We accept MasterCard, Visa, or American Express, or we can bill you.

Advertising
For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Postmaster
Periodicals postage paid in Louisville, KY.
Postmaster: Send address changes to:

Inside Solaris
P.O. Box 35160
Louisville, KY 40232

The main difference between a Java applet and a Java application is that an applet needs to be run from a supporting program, such as a browser or the Java applet viewer program. A Java application may be run from the Java interpreter. An applet relies on the browser for supporting code, where an application is self-sufficient.

A short note about comments—Java recognizes three styles of comments: text enclosed by /* and */, text after // till the end of the line, and finally, text enclosed by /** and */, which has a special significance for the `javadoc` tool that generates documentation for Java applets. Since Java contains no preprocessor, we'll use /* and */ to block out chunks of code, which we would do in C using `#if 0 ... #endif`. Since /* and */ pairs do not nest, we'll leave them for block quotes and use // for line-at-a-time comments.

QUICK TIP: Note that Java, like C/C++, is case-sensitive. The words Class and class are different words. The Java tools (like the Java compiler `javac`) are also case-sensitive.

The HelloWorld applet

Let's start by writing a simple applet that displays the string "Hello, World!" Figure A shows the code for our first Java applet, *HelloWorld.java*.

The first two statements import the Applet and Windowing classes that we're building upon for our example. You must include them, or the applet won't work. The Abstract Windowing Toolkit (AWT) provides the windowing user interface to our Java applet.

Next, we tell Java we're creating a new applet class named *HelloWorld* by inheriting all useful methods (ways of doing things) from the generic Applet class. We redefine the way this applet displays itself on the screen by providing a new version of the `paint()` method. We use the `drawString()` method to display the text inside the applet's `paint()` method.

Using a text editor, we type the Java code above into a text file called *HelloWorld.java*. We then compile the code by typing

```
$ javac HelloWorld.java
```

If the */java/bin* directory isn't included in your `PATH` variable, you'll have to specify the complete path to `javac`, like this:

```
$ /java/bin/javac HelloWorld.java
```

If `javac` compiles the code and finds no errors, it creates a file named *HelloWorld.class* in the current directory. In order to run the applet, we need to create an HTML document telling the browser to run the applet. To do so, create the file *HelloWorld.html* in the current directory. Figure B shows the contents of the new *HelloWorld.html* file.

If you open this HTML file in your Java-enabled Web browser, you should see your first Java applet! You can execute this applet

Figure A

```
import java.applet.* ; // Remember the import statements
import java.awt.* ;  // .. Abstract Windowing Toolkit

public class HelloWorld extends Applet {
   // Display the applet using this method
   public void paint(Graphics g) {
    g.drawString ("Hello, World!", 50, 75 );
   }
}
```

Our first Java applet simply displays a text string in the `paint()` method.

with the `appletviewer` program like this:

```
$ appletviewer HelloWorld.html
```

Once you do so, you'll see the window shown in Figure C appear on your screen.

For each new applet in this article, you'll have to create a new HTML file that looks like *HelloWorld.html*, changing only the name of the class in the APPLET tag. You can, of course, name the HTML files anything you choose.

HelloWorld: Where did the applet go?

Next, we'll change the foreground and background colors of the *HelloWorld* applet and examine an interesting technique for visually integrating Java applets into a Web page. While we could simply add some code to our *HelloWorld.java* program, we'll use this opportunity to show how easy it is to extend an existing class. So rather than modifying *HelloWorld.java*, we'll create a new file, called *Color_HelloWorld.java*, that extends *HelloWorld.java*. Figure D contains the code for our *Color_HelloWorld* applet.

As you can see, we merely redefine the `init()` method that the Applet class executes when it starts. You may be wondering (if you've not seen it in C++) what the keyword `this` denotes in the listing in Figure D. It's a self-reference that is implicitly passed to each method (operation) of the current object, i.e., `this` object—the object for which we're defining the method or operation. In this case, the keyword `this` translates into a reference to the *Color_HelloWorld* applet. And so, the foreground and background colors of this *Color_HelloWorld* applet are set.

We use the `setBackground()` and the `setForeground()` methods to change the background and foreground colors of the Applet window. (For some color definitions, see the sidebar "Standard Java Colors" on page 4.) The *java.awt.Component* package defines the `setBackground()` and `setForeground()` methods. All GUI widgets, except menu components like menubar and menuitems, descend from the Component class. Thus, these two operations work on all these widgets, including applets.

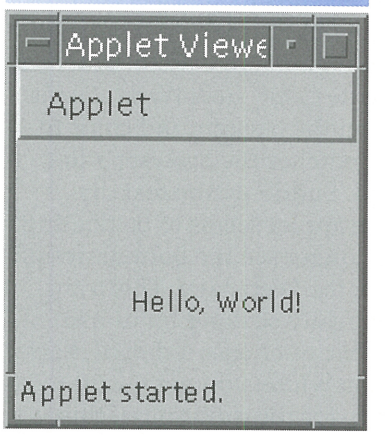
The subclass-superclass relationship uses the ISA hierarchy model. In the ISA hierarchy model, a subclass is a (ISA) specialization of its superclass. For example, if you have the class hierarchy shown in Figure E, you can see

Figure B

```
<HTML>
<APPLET code="HelloWorld.class" width=150 height=100>
</APPLET>
</HTML>
```

This HTML document tells the browser to run the applet code found in HelloWorld.class *in a 150x100 window.*

Figure C



Here's the result of our HelloWorld *applet.*

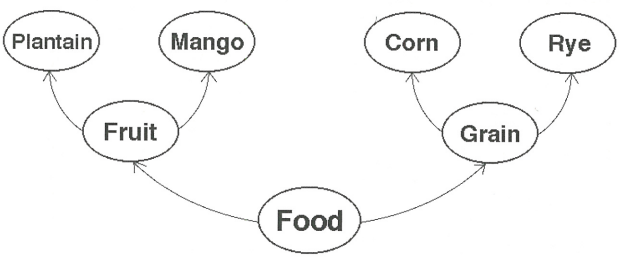
Figure D

```
import java.applet.* ;
import java.awt.* ;

public class Color_HelloWorld extends HelloWorld {
  // Set the background and foreground colors
  public void init() {
   // match your browser background
   this.setBackground(Color.white);
   // whatever foreground color suits your fancy
   this.setForeground(Color.blue);
  }
}
```

The Color_HelloWorld *applet modifies* HelloWorld *by changing the foreground and background colors.*

Figure E



This hierarchy of edibles illustrates the ISA hierarchy model.

that a Mango is a Fruit. So the Fruit class (collection of all fruits) is a superclass of the Mango class (collection of all mangoes). The Mango class is a subclass of the Fruit class.

Similarly, an Applet ISA Panel (defined as a Container that's nested inside another Container) ISA Container ISA Component. This is the subclass-superclass chain that leads from an Applet up to a Component. This example gives you an idea of the hierarchical structure of the elements of the GUI interface. It extends to all aspects of Java, since it is an object-oriented language.

After this diversion into class hierarchies, let's get back to the *Color_HelloWorld* applet. Notice how we set the background color to white, so that it matches the browser's default background. Suddenly the text displayed inside the applet seems to be floating, since we can no longer see the boundaries of the applet within the Web page. This transparency effect is really striking for animated applets. (If your browser uses a default background other than white, you may want to change it to white for the duration of this demonstration.)

Building our first Java application

Now that we've built our *HelloWorld* applet and extended it, it's time to see how to create a standalone application. Let's build a standalone version that can be directly executed by the Java interpreter, outside the browser. To start, create the text file named *HelloWorldApp.java*, as shown in Figure F.

Figure F

```
// Note the absence of import packages
class HelloWorldApp {

    // We must define main() which gets
    // executed by default.

    public static void main (String args[]) {

        System.out.println ("Hello, World!" ) ;
    }
}
```

HelloWorldApp.java *is our first Java application.*

For our application, we forgo the niceties of a windowing system. This Java application is much closer to the traditional "Hello, World!" application.

`System.out` is analogous to the standard output stream (STDOUT) used in C/C++. `println()` starts a new line after the text, just as if you used `printf()` in C and appended a newline (`\n`) to the format string. We compile the application using `javac` as before. However, we run it differently, by executing the Java interpreter:

```
$ java HelloWorldApp
Hello, World!
```

As you can see, the result is that the text "Hello, World!" gets printed. Here we've built a text-mode application, but this shouldn't lead you to conclude that Java applications are restricted to text mode. They can have graphical user interfaces as well. In fact, we'll build one in the next section.

Adding an input text field

We'll now build a Java application that allows you to type text in a field, which it then displays in a different pane. To do so, we implement a special kind of frame for our application. A frame is defined as an optionally resizable top-level application window. We'll define a subclass of the Frame class called TextFieldApp.

Each such frame will contain a text field for the user to type in text and a text area for the text to be displayed. We create the frame, then

Standard Java colors

The package *java.awt.Color* provides access to all the color capabilities of the Applet environment. While you can create any colors you want, *java.awt.Color* provides a few predefined colors for you:

Color.black	Color.green	Color.pink
Color.blue	Color.lightGray	Color.red
Color.cyan	Color.magenta	Color.white
Color.darkGray	Color.orange	Color.yellow
Color.gray		

You can look up these constants, as well as all the methods for the Color class, in the API documentation available from *http://java.sun.com/java.sun.com/newdocs.html*. Please note that these colors may appear slightly different on various video card/monitor combinations, as few are calibrated.

add the text area and text field to it, after having customized them both for our purposes. We add them so that the text area is at the top of the frame, and the text field is at the bottom. Java will automatically do the layout for us. There are more complicated layout managers in Java, which allow the specification of elaborate layout constraints. These are useful for laying out windows containing several components.

Since this is a standalone Java application, we must define a `main()` method, in which we create, resize, and paint the frame. Finally, we need to write an event handler. In this case, the event handler is simpler than usual, since we're not concerned about mouse-clicks, keyboard events, or windowing events (like the window being moved). If we'd been interested in those events, we'd most probably have redefined the `handle-Event()` method. Since we're interested only in copying the text entered in the text field to the text area, we can take a shortcut and redefine the `action()` method to copy the text when necessary.

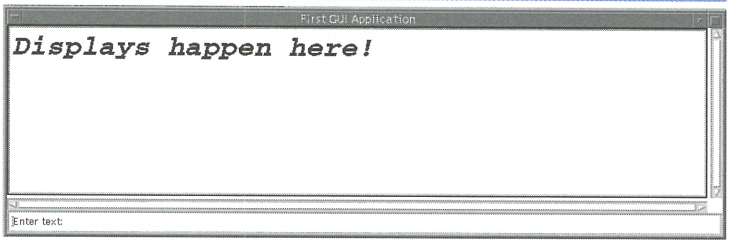
The code for our *TextFieldApp.java* program is shown in Figure G. Notice that we've left out, for the purposes of this exercise, elementary components of the window, such as a menu bar that would allow us to choose to quit the application.

Now you can compile and run the *TextFieldApp.java* program using the commands

```
$ javac TextFieldApp.java
$ java TextFieldApp
```

When you do so, you'll see the window shown in Figure H. To play with the application, type in some text in the lower text field, and press the [Enter] key. You should see the text in bigger letters echoed in the text area above. You'll have to kill this window manually, since we haven't implemented an OK button or a Quit menu item.

Figure H



Here's what our Java application looks like when it first starts running.

Figure G

```
import java.awt.* ;

public class TextFieldApp extends Frame {
    TextField textfield ; // For typing in text
    TextArea textarea ; // and displaying it...
    Font   textareafont ;

    public TextFieldApp (String title) {

    // Create a Frame with the title
    super(title) ;

    // Add text field and textarea
    textfield = new TextField ("Enter text:", 40);
    textarea = new TextArea ("Displays happen here!",
                5, 40) ;
    textarea.setEditable(false); // Don't allow user to
                                 // modify it!
    textarea.setBackground(Color.white) ;
    textarea.setForeground(Color.black) ;
    textfield.setForeground(Color.black) ;

    // Set BIG font for display of textarea.
    textareafont = new Font("Courier", Font.BOLD +
       Font.ITALIC, 36) ;
    textarea.setFont(textareafont) ;

    // Add these elements to the current frame.
    this.add("North", textarea) ;
    this.add("South", textfield); // very simple layout
    } // end TextFieldApp() method

    // All applications must have the main() method.
    public static void main(String args[]) {

      Frame f = new TextFieldApp ("First GUI
       ➥Application");

      f.pack(); // Resize to appropriately small size.
      f.show();

    } // end main() method

    // Handle text being typed in, and transfer it to
    // the text area.
    // The more generic method called handleEvent()
    // can handle keyboard & mouse events and would
    // be needed for more complex windows.
    public boolean action (Event event, Object argument) {

      // Text typed into text field ?
      if (event.target == textfield) {
        // Display text
        textarea.setText((String)argument + "\n" ) ;
        return true;
      }
      else {  return false ; }

    } // end action() method

} // end TextFieldApp class
```

This Java application copies any text entered into the text field to the text area.

Information resources

You can find many auxiliary sources of information about Java on the Internet. If you're interested in the cross-platform aspects of Java development, you can keep abreast of the JVM porting efforts for different platforms in the document *http://java.sun.com/java.sun.com/Mail/external_lists.html*.

For a registry of many existing Java applets (along with the source code for many of them), you'll want to visit *http://www.gamelan.com/*. Also, check out *http://java.sun.com/java.sun.com/applets/index.html* for an index of many Java applets you can tinker with.

Don't forget to check your local bookstore for books on Java. Unless your bookstore is horribly out-of-date, there'll be about a trillion different books on Java. Be careful before you buy a Java book: Since Java is the current fad language, many books on it are junk.

Conclusion

We've shown you how to build simple Java applets and standalone applications. We also demonstrated how to extend an existing applet for which we had the Java source code. Finally, we explained how to create a Java application. Since the JDK is freely available on the Internet, and there are versions for both Sparc and Intel, you have no excuse not to download it and start playing with it! ❖

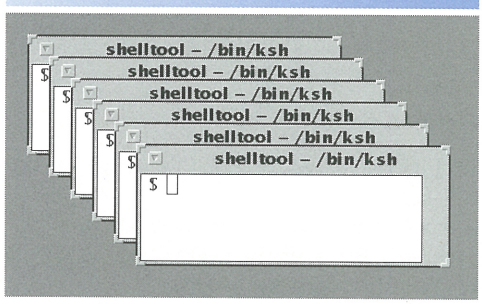
Probal Shome is a Senior Technical Analyst with Oracle Worldwide Support. You can reach him at pshome@us.oracle.com.

CUSTOMIZE YOUR ENVIRONMENT

Changing the title bar of an xterm window

If you use OpenWindows, you've probably had lots of Command Tool and Shell Tool windows open at the same time. If you have enough of them on the screen, it can get pretty confusing. The more windows you have open, the less of each window you can see before they overlap. Which one are you using for which project? Figure A shows the windows overlapping with only the title bars showing.

Figure A



While it's handy to have the ability to use multiple windows to do your job, it can be confusing.

As you can see, all the title bars show the same information: shelltool - /bin/ksh. The title bar is supposed to be informative, but it's telling us only that it's a Shell Tool window running */bin/ksh*. The Command Tool and Shell Tool programs, among others, both use an `xterm` window for I/O. In this article, we'll show you how to change the title bar for an `xterm` window to anything else you want.

Changing the text of an xterm window

It turns out that an `xterm` window treats the title bar and the normal window area as two different windows. The `xterm` provides a special character sequence you can use to put text in the alternate window (i.e., the title bar) and another character sequence to tell the `xterm` when to go back to the original window.

The sequence of characters you use to tell the `xterm` to start putting text in the title bar is [Esc],], then l. You then may send it the text you want on the title bar. When you're finished, send the `xterm` an [Esc] character followed by \.

For example, if you're using OpenWindows, start a Shell Tool and type the following command:

```
$ echo "^[]lNew Titlebar Text^[\\"
```

When you press the [Esc] key, the `xterm` window displays ^[. Also, the shell interprets the \ character as an escape, which tells it to treat the next character as if it had no special meaning. In order to send the \ character to the `echo` command, you need to use two \ characters: The first tells it the next character has no special meaning; the second is the character itself.

A quick shell script

Actually remembering and typing the appropriate escape sequences is no great hardship. However, it makes things simpler to make a shell script to do the job for you. Figure B shows the shell script *WinName*, which will name a window for you.

Keep in mind that ^[is the [Esc] key, not the keys [Ctrl] and [. If you're using `vi` to enter the shell script, just press [Ctrl]-V before pressing the [Esc] key to enter it. One last note: We're comparing the `TERM` environment variable to `sun-cmd` because the `termcaps` entry for an `xterm` is `sun-cmd`.

If you use this script, you won't have to remember the escape sequences, and you won't have any problems if you mistype something. If you mistype the [Esc]\ part, for example, you'll lose some of your output from successive commands, as the output will go to the title bar. However, the `xterm` doesn't actually change the title bar until it receives the [Esc]\ character sequence, so you won't see a change.

Once you enter the shell script, don't forget to make it executable with the command

```
$ chmod +x WinName
```

When you use this script, you need to keep something in mind: If you want spaces in your text, enclose the new window title in quotes. Otherwise this script will set the window title to the first word only.

Figure B

```
if [ $TERM = "sun-cmd" ]; then
  echo "^[]l" $1 "^[\\"
else
  echo "Sorry, you can only use this in a
    ➥sun-cmd window"
fi
```

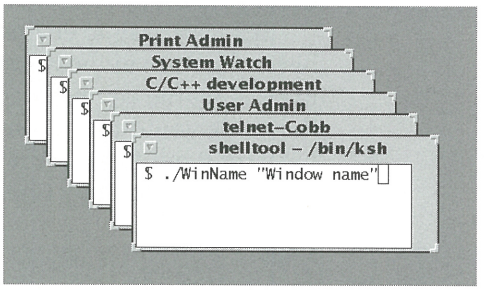
This short shell script will change the title bar in a sun-cmd (i.e., `xterm`) window.

Conclusion

Remember the ugly screen shown in Figure A? Figure C shows the same screen, but with informative title bars.

You can use this technique to put any information in the title bar of your `xterm` windows. Once you do so, you'll find it much easier to locate the window you want, since the title bar now describes the window. ❖

Figure C



It's much easier to find the window you want when the title bars have useful information in them.

CUSTOMIZING YOUR SYSTEM

Using command-line history in the Korn shell

By Al Alexander

Recently, someone cornered me and asked why UNIX doesn't provide command-line history like DOS does. In DOS, after you execute the DOSKEYS command, you can recall and edit command lines. Actually, UNIX does provide a similar capability. As I spend most of my day working at the command line in the Korn shell, life with Solaris would be far more difficult without the command-line history feature.

However, command-line history works differently depending on which shell you're using. In this article, we'll show you how to enable and use command-line history in the Korn shell. Next month, we'll show you how to use command-line history in the C shell.

Enabling command-line history

The Korn shell provides a very sophisticated command-line history mechanism. Using it, you can retrieve previously entered commands, as well as edit and execute them. You can use two primary modes, `vi` and `emacs` mode. (There's also a `gmacs` mode, but it's nearly identical to `emacs` mode.)

These modes allow you to edit the command line as if you were using the editor named. For example, in `vi` mode, the Korn

shell uses insert and command modes for the editor, just like vi..To enter command mode, you press the [Esc] key. Then you can use the normal cursor-movement keys for moving around and editing text. You can go back to insert mode by pressing *i*, *a*, or *A*, just as you'd expect. Table A shows a subset of the editing commands you can use in vi and emacs modes.

If you use either vi or emacs as your editor-of-choice, it's easy to select which mode to use. If you use another editor, like the CDE text editor, you'll just have to choose the one you like better. It may be worthwhile checking them both out.

It's very easy to enable the command-line history feature in the Korn shell. All you need to do is set the environment variable VISUAL to the mode you'd like to use: emacs, gmacs, or vi. Let's try it out. If you're not already in the Korn shell, enter it by typing

```
$ ksh
```

Now let's enable the command-line editing mode. For this demonstration, we'll use vi mode.

```
$ VISUAL=vi
```

To demonstrate the re-use of previously entered commands, let's first create a brief three-command history. Type in the following three UNIX commands, and then we'll show how to easily re-use these commands within the Korn shell:

```
$ ls -al
$ ps -ef
$ who
```

Now that we've put some commands in your history, let's call up the last command we entered, who. To do so, press the [Esc] key, then press *k*. You should then see the who command, with the cursor on the w. At this point, you're in a one-line version of the vi editor. The history list is treated like a file, and your command line is like a one-line view port into this file. You begin in vi's insert mode, so pressing the [Esc] key puts the editor into command mode. Pressing the *k* key moves you up a line to the previous command. If you press *k* again, you'll go up to another command, in this case ps -ef. Press *k* again, and you'll see the ls -al command again. Press *j* to go to the next command, and you'll be back at the ps -ef command.

As in the regular vi editor, typing a capital *A* allows you to append to the end of the current line. Go ahead and type a capital letter *A*, then type

```
| more
```

Now your command line should look like this:

```
$ ls -al | more
```

You can execute the command by pressing the [Enter] key.

The capabilities you just saw are some of the things that make this feature so desirable. The vi editor offers many other capabilities, such as searching for previous commands with the / character, that make the command-history editing feature very powerful. Just as the vi editing mode is powerful, emacs mode provides similar features.

Table A

Description	vi (insert mode)	vi (command mode)	emacs
Retrieve next command	*n/a*	j	[Ctrl]-N
Retrieve previous command	*n/a*	k	[Ctrl]-P
Move left one character	*n/a*	h	[Ctrl]-B
Move right one character	*n/a*	l	[Ctrl]-F
Move left one word	*n/a*	b	[Esc]b
Move right one word	*n/a*	w	[Esc]f
Enter insert mode	*n/a*	i, a, A	*n/a*
Exit insert mode	[Esc]	*n/a*	*n/a*
Go to start of line	*n/a*	^	[Ctrl]-A
Go to end of line	*n/a*	$	[Ctrl]-E
Execute command	[Enter]	[Enter]	[Enter]
Delete current character	*n/a*	x	[Ctrl]-D
Delete current word	*n/a*	dw	[Esc]d

These are some of the most-commonly used command-line editing commands in vi and emacs.

Once you enable the command-line history editing mode, you not only gain the ability to retrieve previous commands and edit them, you also get the ability to edit the line you're currently working on. If you've entered 30 or so characters and notice that the first character is wrong, you can simply go to the start of the line, edit the character, then go back to the end and continue typing.

Terminal requirements

For the in-place command-line editing to work, your terminal must support two basic features. First, your terminal must not automatically advance to the next line when it receives a carriage return without a new line. Second, when your terminal prints a space, it must overwrite the character that was originally at the location. These two features allow you to use in-place command-line editing by allowing the terminal to update the command line as it's being edited.

Editing a very long line

As a result of the modest requirements the in-place command-line editing feature places on your terminal, the Korn shell uses only one physical line when you edit a command line. As you notice, the terminal isn't required to have a character used to move the cursor up a line. Since the cursor motion is limited to left (via the carriage return), right (by printing characters), and down (via the new line), the Korn shell uses a single line on your terminal. Otherwise, in-place editing for long command lines would be impossible. When you're editing a command line larger than the width of your terminal, the Korn shell displays a special character at the right margin to tell you so.

If you're at the start of the long command line, the Korn shell displays a > at the right margin to show you that there's more command line to the right. If you're at the end of the long command line, you'll see a <, showing you that there's more command line to the left of the left margin. If your command line is so large that it extends past both the left and right margins, the Korn shell will display an * at the right margin.

Unless you tell it otherwise, the Korn shell assumes that your terminal is 80 characters wide. You can specify the size by setting the `COLUMNS` variable. Let's try an example just to see how the Korn shell handles a very long line. We're going to use the `vi` editing mode in this example. First, let's start the Korn shell, set `vi` mode for command-line editing, and set the column width to 40 so we don't have to use an exorbitantly long command line:

```
$ ksh
$ VISUAL=vi
$ COLUMNS=40
```

Now, let's enter a relatively long command line:

```
$ now is the time for all good men to come to
➥the aid of their party.
```

When you enter this command line, you'll see it in its entirety, even if it wraps to the next line. Now press the [Esc] key, then the *k* key to invoke the command-line editor. When you do so, you'll see

```
$ now is the time for all good men to>
```

The > symbol indicates that the command line extends past the right margin. If you use the *L* key to advance the cursor to the space after the word *to*, you'll see

```
$ r all good men to come to the aid o*
```

Now, press *A* to start editing at the end of the line. When you do, you'll see

```
$ id of their party.          <
```

The command-line history

If you'd like to examine the last few commands you've entered, you can type

```
$ history
```

This will print the last few commands that you've entered.

Each time you start the Korn shell, it loads the command-line history from the previous session. This makes life much simpler if you tend to do the same operations over and over. Rather than having to type them in again, you can access the commands you used in a previous session.

Unless you override it, UNIX stores your command-line history in the file *.sh_history*. You may override it by setting the `HISTFILE` environment variable to the name of the file you'd rather use. You can use this feature to set up different command-line histories for

different terminal windows. Thus, if you tend to work on multiple projects at once, like most of us, you can have a different command-line history for each project. You can even create a text file of commands you like to use, then load them.

You can also tell UNIX just how much history you want to store by setting the `HISTSIZE` variable. This tells how many commands will be available to you between sessions. Normally, this defaults to 128 commands.

Conclusion

UNIX provides three different, but related, mechanisms for giving you the ability to work with a command-line history. In this article, we've explored command-line history for the Korn shell. Next month, we'll take a look at command-line history in the C shell. ❖

SYSTEM CONFIGURATION

Orderly shutdown during extended power failures

By Jerry L. M. Phillips, M.S.

Do you depend upon your organization's auxiliary service departments or individual users to call you when your Sun equipment loses AC power during the night or on weekends? Do the calls ever come too late to permit an orderly shutdown of your Solaris operating systems? Worse yet, do you discover power losses after you return to work? If so, a time-saving alternative is available.

Following the last prolonged weekend power outage at my campus, which I heard about 10 hours after the fact, I spent two aggravating days stabilizing the file systems on a crashed Usenet News-World Wide Web server. This prompted my search for a solution that offered timely, dependable notification of long power failures and an orderly shutdown and remote reboot of the affected systems.

In this article, I recommend a hardware and software combination that solved my problems. The combination includes an American Power Conversion (APC) Smart-UPS 700, an APC Call-UPS II remote control accessory, a Measure-UPS II supplementary surveillance accessory, and APC PowerChute Plus v4.2.1 for Solaris software.

Here's how it works

The process is fairly straightforward. The Smart-UPS 700, attached to your Solaris/SPARC system, registers a power disturbance and switches to battery operation. If AC power doesn't resume within a few minutes, the Call-UPS II initiates a paging sequence. It uses a modem that you attach to the Call-UPS to dial a pager number. Your pager receives the call and displays the site ID with a Smart-UPS event status code indicating that the Smart-UPS is on battery.

You may then dial into the Call-UPS II from a remote workstation to examine the event log and also check the load parameters on the Smart-UPS. In the meantime, the UPS status changes to low battery. Realizing that you can't respond onsite in time, you may tell the Call-UPS II to shut down the system gracefully and turn off the UPS. When power resumes, your system reboots automatically. Now, let's go through the steps necessary to configure the Call-UPS II and PowerChute Plus software.

Call-UPS II configuration

Following installation and testing of the Smart-UPS, connect the Call-UPS cable (marked UPS on the connector) to the computer interface port on the UPS. Next, connect the */dev/ttya* serial port on your Solaris/SPARC computer to the Call-UPS management port using the null modem cable provided. This temporary connection will allow you to configure various parameters on the Call-UPS.

In our example, you'll configure the management port to send pages via modem and accept dial-in calls. Enter the command

```
$ /usr/sbin/eeprom | grep ttya
```

and confirm that the default values of the serial port selected are

```
ttya-rts-dtr-off=false
ttya-ignore-cd=true
ttya-mode=9600,8,n,1,h
```

If your results do not agree with the parameters on the third line above, enter the command

```
$ eeprom ttya-mode=9600,8,n,1,h
```

This will configure your port so it can communicate with the Call-UPS. Check your */etc/remote* file—you need to be sure that you have the serial port configured properly. You may need to edit the file and insert the following line before executing the `tip` command:

```
cuaa:dv=/dev/cua/a:br#9600
```

If everything is ready, you can access the Call-UPS through the port by entering the command

```
$ tip cuaa
```

Now, press [Ctrl]-P, and the Call-UPS should respond with the *Enter Password>* prompt. Type in the uppercase letters *APC* and press the [Enter] key. The Smart-UPS 700 banner page should appear. Press any key to continue, and the Smart-UPS 700 should display the MAIN MENU, shown in Figure A.

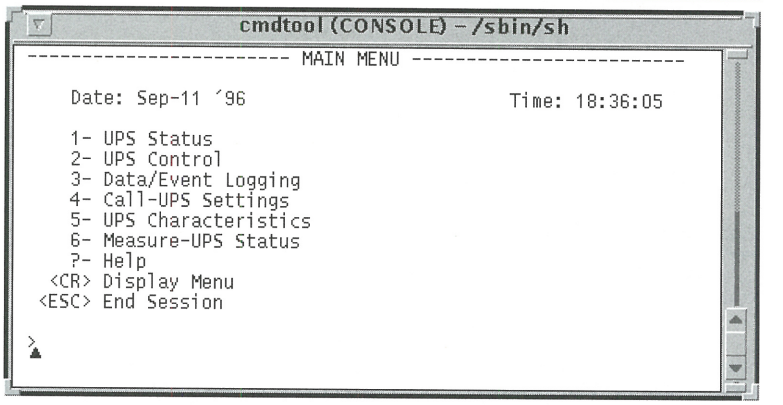
Select Call-UPS Settings (item 4), and you should see the CALL-UPS SETUP screen shown in Figure B. Follow the Call-UPS user's manual instructions to enter the Date, Time, and Location settings. Change the Answer Ring setting to 1 or 2. This setting represents the number of rings your modem waits before answering an incoming call.

For the purposes of this article, we'll ignore the Dial Back, Dial Back Str, and Answer Lockout settings. These manage dial-back security and control communication sessions over multiple devices using the same telephone line. After you get your system working, you may want to explore these settings for added security for your UPS. (You probably wouldn't like a spiteful worker or student hacking into your UPS and turning off your system.)

Now select item 11, the Pager Setup Menu. When you do so, the Call-UPS will display the PAGING SETUP screen, shown in Figure C. Here, you configure the Call-UPS to page you if there is a change in the Smart-UPS status. You enable paging by changing the Paging setting to ON.

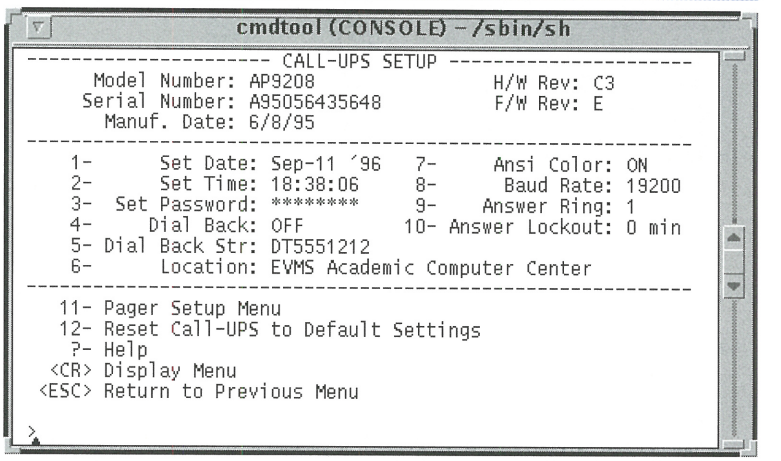
The Dial String 1 setting stores the pager number and modem commands to dial your pager. This will depend on the type of modem you use. Our example uses a standard Hayes-compatible modem, so the DT means dial with touch tone; then we send a 9 to get an outside line. The comma tells the modem to pause for two seconds. Then comes our pager number.

Figure A



Once you're communicating properly with the Call-UPS, you should see the MAIN MENU.

Figure B



You use the CALL-UPS SETUP screen to set up the date, time, and communications settings.

Finally, we use the @ to tell the Call-UPS to wait five seconds before continuing.

Whenever the UPS status changes, the Call-UPS will call the number specified by Dial String 1 and then send the site ID number, the space character (specified by item 9), followed by the numerical code corresponding to the UPS status. You can also configure a second pager number into Dial String 2 to make the system call another pager.

At the bottom of the PAGING SETUP screen, you can configure which error conditions will trigger a page and on which pager. You may choose to have some conditions call both pagers, some call only pager 1, and others call only pager 2. In this instance, we're sending all events to pager 1 only.

If you're paranoid about your pagers, you can also tell Call-UPS to repeat your pages, along with an interval between pages. This way, if your pager line just happened to be busy the first time it attempts to call, it should get through the second (or third…) time.

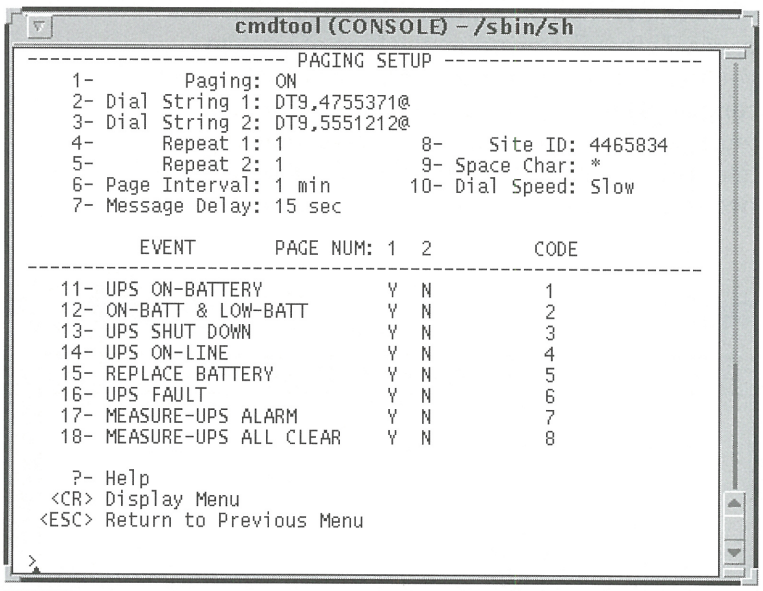
Now that you've configured the pager settings, press the [Esc] key twice to return to the MAIN MENU. Essentially, the Call-UPS II configuration is complete.

Other items on the MAIN MENU allow you to configure some advanced options, as well as examine many details about the UPS operation, such as utility line input, UPS output, UPS load, run time, UPS temp, freq, last UPS transfer to battery, voltage, battery capacity, date of last battery change, and results of last self test.

When you call in from a remote location, you'll use the UPS Control menu. From this menu, you can turn the UPS on and off, run a self test, simulate a power failure, and gracefully shut down the attached computer.

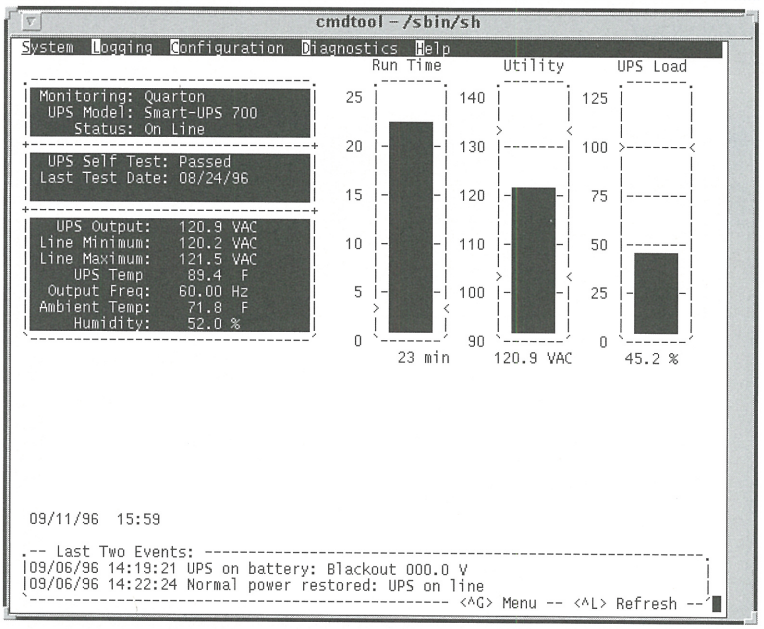
To end the session, press the [Esc] key, type *yes* to confirm and save your configuration changes, and press [Enter]. Type ~. (tilde period) to terminate the `tip` program. Disconnect the null modem cable from your serial port and from the Call-UPS Management port.

Figure C



You can choose which events are important enough to page you with the PAGING SETUP screen.

Figure D



The PowerChute software allows you to monitor the status of the UPS at any time.

PowerChute Plus configuration

The next step is to install the PowerChute Plus software. Its features include scheduled server shutdown, interactive/scheduled battery testing, detailed power quality logging, and real-time graphical displays that show battery voltage, battery capacity, UPS load, utility line voltage, and run time remaining.

The PowerChute software consists of two modules. The first module is a UPS monitoring module daemon (upsd) that executes as a background process and communicates with the UPS via a serial port. The second is a user interface module that provides system monitoring,

logging, configuration, and diagnostic capabilities from the computer.

The software comes with cables and serial port adapters. Connect serial port */dev/ttya* to the UPS monitor port on the Call-UPS. Follow the detailed PowerChute Plus user's guide to install the software. It requires some familiarity with two UNIX port management tools—`pmadm` and `sacadm`. You'll have to disable logins on the serial port that PowerChute uses to communicate with the UPS.

First, we'll disable the default port monitor on */dev/ttya* by executing the command

```
$ sacadm -d -p zsmon
```

Now we'll disable the service that controls the */dev/ttya* port monitor with the command

```
$ pmadm -d -p zsmon -s ttya
```

Once you're finished, you should list all the port monitors and services on the system to ensure that you've properly turned them off for */dev/ttya*. You can list all port monitors on the system by using the command

```
$ sacadm -l
```

Similarly, you can list all services on the system by using the command

```
$ pmadm -l
```

Disabling serial port logins will remain in effect only until you shut down the computer. You may choose to remove the port monitors and services or create a file in */etc/rc2.d* that disables port logins at startup.

The installation process modifies the file */sbin/rc0* and also creates the file */etc/rc2.d/S98upsd* that starts and stops the upsd daemon. The daemon will start automatically during system boot, or you can issue the commands below as needed:

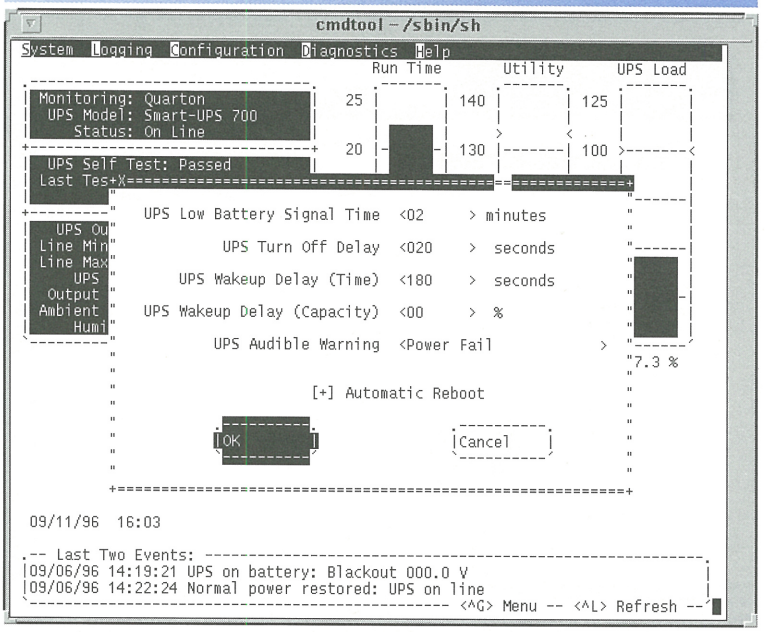
```
$ /etc/rc2.d/S98upsd start
$ /etc/rc2.d/S98upsd stop
```

You start the character-based user interface module by typing

```
$ /usr/lib/powerchute/powerchute
```

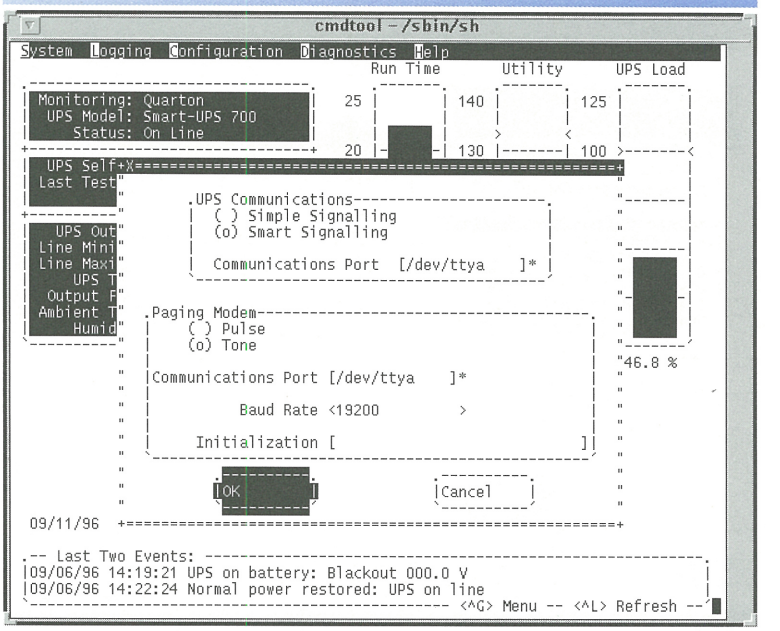
Press the spacebar to accept the UPS online prompt. Then, enter the password for the account you're using and press the [Enter] key. The PowerChute user interface module will appear as shown in Figure D. Please note the *Ambient Temp* and *Humidity* parameters: The Measure-UPS performs temperature sensing, humidity sensing, and contact monitoring, and provides this data to the Smart-UPS. Type [Ctrl]-G to access the different menu options.

Figure E



You use this screen to configure the basic parameters for the UPS.

Figure F



You configure the communications port the UPS is connected to with this configuration screen.

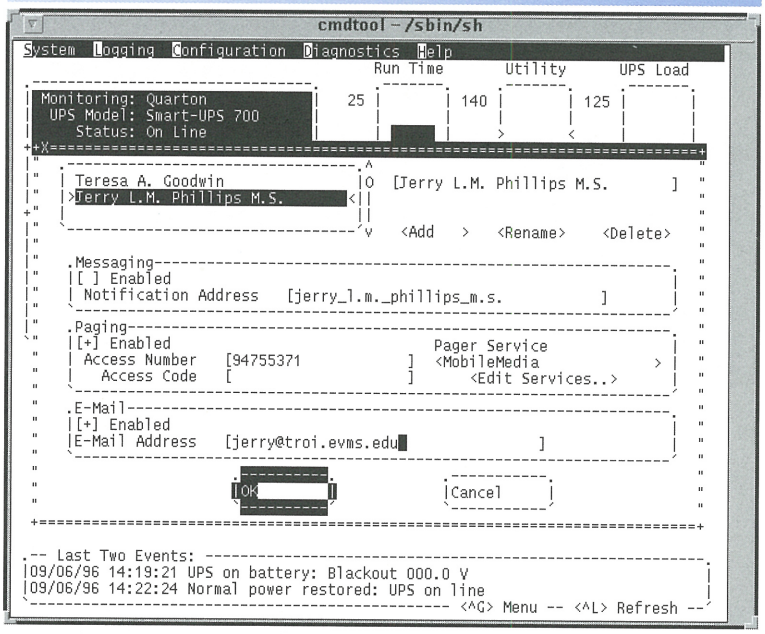
Press the right arrow key to highlight the Configuration menu item. Press the down arrow to highlight UPS Shutdown Parameters, then press the spacebar. When you do so, you'll see the screen shown in Figure E on the previous page. Now use the [Tab] key to move within the dialog box. Please note that we set the configuration as shown in Figure E. When you finish, press [Tab] until you get to the OK button, then press the spacebar.

Next, we need to tell PowerChute which communications port the UPS is connected to. Select Configuration Communications Parameters and use the settings shown in Figure F on the previous page. Please note: The paging modem function didn't work for me! APC technical support recommended using the Call-UPS II instead. A future version of PowerChute may support the paging modem option without the Call-UPS.

You can tell PowerChute to inform people about changes in the UPS status with the Configuration Event Users menu item, shown in Figure G. Select Configuration Event Users from the MAIN MENU, and enter settings appropriate to your site. As you can see, you can enable E-mail and paging.

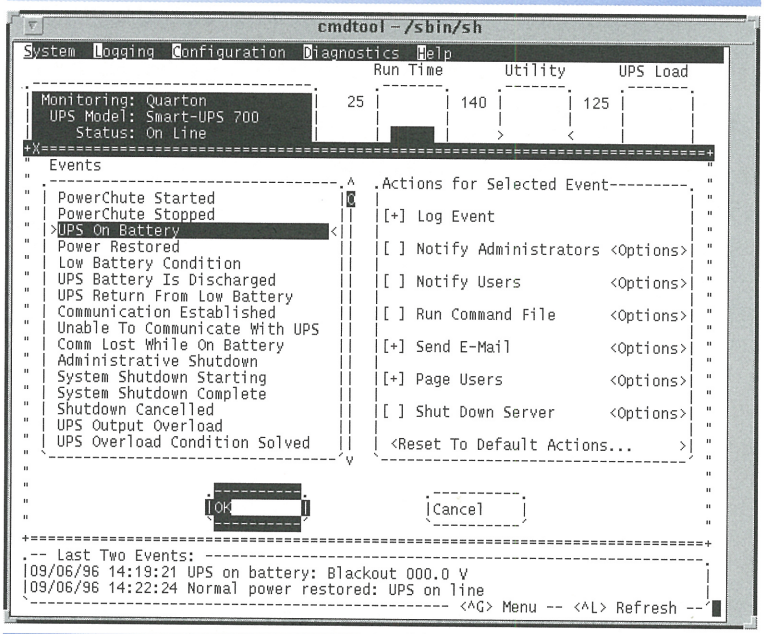
Finally, we need to tell PowerChute what to do when a specific event occurs. To do so, select Configuration Event Actions from the MAIN MENU. When you do, you'll see the screen shown in Figure H. On the left side of the screen is a list of events, and on the right is a set of actions you can trigger when the selected event occurs. Figure H shows that the software will write a log entry and send E-mail in response to the UPS On Battery event.

Figure G



The Configuration Event Users screen allows you to set the E-mail addresses to notify about UPS changes.

Figure H



PowerChute allows you to execute a wide variety of actions on any or all events.

Test your setup

Once you get your system set up, you should test it. Make sure that you're not running anything critical, and unplug your UPS. You should receive a page that shows the site ID and event status code. You should also receive E-mail if you configured your system to send it. Finally, you should call into the Call-UPS module to test its configuration and modem connection.

Conclusion

If you can't afford the time it takes to rebuild your file systems, you need to ensure that your system doesn't go down unexpectedly. While you can't ensure 100 percent up time, a UPS with good support software can go a long way towards giving you peace of mind. ❖

Who has an account on your system?

If you're like me, you sometimes need to see who has an account on a particular machine. Obviously, you can just take a look at the */etc/passwd* file and see. Figure A shows an */etc/passwd* file. However, you have to ignore the entries that don't correspond to a user account, and let's face it, the */etc/passwd* file is ugly. In this article, we're going to present a small script that will print a list of the user accounts on your machine in a nicer format, sorted alphabetically by the login name.

Figure A

```
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
smtp:x:0:0:Mail Daemon User:/:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/
  ➥usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
marco:x:100:10:Main user:/export/home/marco:/
  ➥bin/ksh
fred:x:101:10:Test user:/export/home/fred:/bin/
  ➥csh
```

The /etc/passwd *file contains all the information you need, but it's not easy to read.*

As you can see in Figure A, the entries in the */etc/passwd* file are in no particular order. We also see that the first entry in each record is the login name of the user. Since we want to alphabetize our list, the first step is to sort the */etc/passwd* file, like this:

```
$ sort </etc/passwd
```

Now we want to take the resulting file and print it out in a prettier format. The `awk` tool is the best choice for the job. It's designed to read a text file one record at a time, where a record is a single line, and break it into fields. Then you can process the record according to the rules you hand `awk`. (If you're unfamiliar with `awk`, see the article "An Introduction to awk" in our May issue.)

However, `awk` normally breaks a record into fields at white space. Thus, it interprets the line

```
now is the time
```

as a record containing four fields. Unfortunately, the */etc/passwd* file doesn't lend itself to processing like this. A record doesn't necessarily contain spaces, and when it does, they're not at field boundaries. For example, this line has two spaces, but they both appear in the comment field:

```
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
```

The */etc/passwd* file is arranged such that each line contains seven fields, separated by colons. The field definitions are shown in Table A.

Table A

Field	Description
1	The user's login name
2	The encrypted password (archaic, now used for `pwconv` maintenance)
3	User ID
4	Group ID
5	Comment field, it often holds the user's full name
6	Home directory location
7	Login shell to use

These are the field definitions for the /etc/passwd *file.*

If we could tell `awk` to split records apart using a colon rather than white space, printing our report would be a simple task. Luckily, you can tell `awk` the character to use to split a record apart with the `-F` switch. Simply use `-F?`, where `?` is the character to use as a field separator, like this:

```
$ sort </etc/passwd | awk -F: ' { print
      ➥$1,$5,$7 } '
```

Figure B

```
# ISOLusers - Display the username, comment
# and shell each user on the system.

sort </etc/passwd | awk -F: '

# Print column headers
BEGIN {
  printf "%-8.8s ", "UserName"
  printf "%-25.25s ", "Comment"
  printf "%-24.24s\n", "Shell"
  printf "%8.8s ", "--------"
  printf "%25.25s ", "-------------------------"
  printf "%24.24s\n", "-----------------------"
}

# Process each record that has a shell entry
# (This is how we differentiate a user from a
#  daemon or other entry in /etc/passwd.)
length($7)>0 {
  printf "%-8.8s %-25.25s %-24.24s\n",$1,$5,$7
}

'
```

The ISOLusers *script prints an alphabetized list of all user accounts on your system.*

This command prints out the user's login name, comment field, and login shell. Now all we need to do is fine-tune it to make it prettier, and we're done. To do so, we use `awk`'s `printf` command, so we can force the columns to be fixed width. We also added column headings to make the report nicer. The last thing we did was tell `awk` to only print records for the users who had a shell defined. This way, you don't have to look through the daemon processes and other entries that don't correspond to user accounts. Figure B shows the finished script, named *ISOLusers*.

Conclusion

We've presented a simple script you can use to find out who has an account on your system. For our purposes, we needed only the user name, comment, and shell, but you can modify it to suit your own needs. Now you don't have to weed through the */etc/passwd* file in search of a user's account. The script `ISOLusers` will print a neat list for you. ❖

LETTERS

More directory switching with the Korn shell

Your quick tip entitled "Quickly Switch Between Directories" in the July issue was good, but it might have been worth mentioning another feature of the Korn shell. In particular, you might have discussed the ability to define `CDPATH` and the use of `cd -` to jump to the previous directory.

Glen
via the Internet

Glen, thanks for pointing that out. Each time you change directories in the Korn shell, it stores your new working directory in the PWD environment variable. It also stores your previous working directory in the `OLDPWD` variable.

So one way to change to your previous directory is to use the command

```
$ cd $OLDPWD
```

As you noticed, however, the Korn shell gives you an even simpler method of switching back to your previous directory. The implementers of the Korn shell gave the shorthand notation of `cd -`, which tells the Korn shell to change to the directory stored in the `OLDPWD` environment variable. ❖