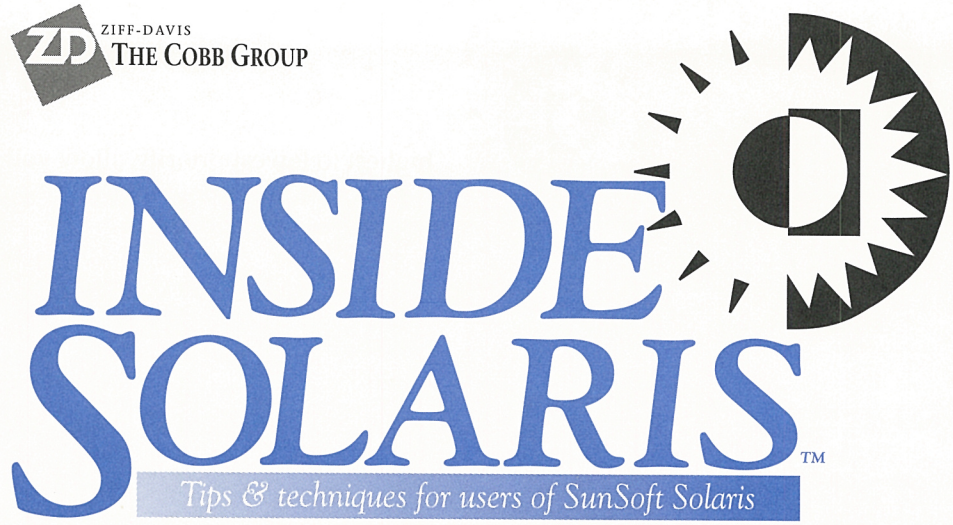
April 1998 • $11.50
Vol. 4 No. 4

ZIFF-DAVIS
THE COBB GROUP



IN THIS ISSUE

Visit our Web site at http://www.cobb.com/sun

Capturing error messages

Effective system administration means that you have to keep the machines in your purview running at their best efficiency. Practically, this means that you must find out about error conditions whenever possible. You don't want the information to be lost in the weeds. To help you out, Solaris provides a system-logging facility to help you log error messages.

Solaris' system-logging facility helps you manage log messages first by filtering the messages according to type, then formatting the messages into a consistent format:

```
Jan 12 23:02:40 Devo unix: fd0 at
➥fdc0
```

Each line shows the date, hostname (Devo, in this case), the process that emitted the message, followed by the actual message body.

In this article, we'll show you how to configure the system logger. Then, we'll show you how you can send messages to it in your programs and script files.

Types of system messages

Solaris consists of several independent subsystems, each of which may emit logging information. If all the messages were mixed together, it could be more difficult than necessary to scan through your log files to detect and correct problems. For this reason, the system-logging facility groups messages by facility and severity. This helps you manage the error messages, ignoring those that don't concern you.

Coverage of subsystems is inconsistent. Some subsystems have specific facility codes, such as cron and mail. Other subsystems are grouped by class, such as daemon and user. Table A shows the facility codes currently in use by Solaris.

The severity groupings, shown in Table B on page 2 in order of

Table A: *Solaris facility codes*

Facility code	Generated by
user	User processes
kern	Solaris' kernel
mail	The mailing system
daemon	System daemons
auth	The authorization system
lpr	Line printer spooling system
news	News servers
cron	The cron and at services scheduling
local0—local7	Reserved for site-specific uses
mark	Internal timestamp

ZIFF-DAVIS
a SOFTBANK company

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
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@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: cobb_customer_relations@zd.com

Staff
Editor-in-Chief ..... Marco C. Mason
Print Designer ..... Margueriete Winburn
Editors ..... Karen S. Shields
Joan McKim
Michael E. Jones
Publications Coordinator ..... Linda Recktenwald
Product Group Manager ..... Michael Stephens
Circulation Manager ..... Mike Schroeder
Publisher ..... Jon Pyles

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.

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

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

highest to lowest priority, allow you to narrow your focus considerably. If you don't care about informational messages from the mail system, you're free to ignore them.

Table B: *Severity codes in descending priority*

Severity code	Generated because
emerg	Panic conditions may cause the system to shut down.
alert	Situations occur that need immediate correction.
crit	A critical error occurred.
err	A normal error occurred.
warn	A process is warning about a potentially serious situation.
notice	A process noticed something important that's not an error.
info	Information about a process.
debug	Information useful to author/programmer of the subsystem.

Configuring the system logger

Since you can select messages by their facility and severity, you can see exactly those messages you want. You can view some messages when they occur, save some in one or more files for viewing later, or send some to one or more operators.

You can configure the system logger to do what you want by editing the */etc/syslog.conf* file. Each time the system logging daemon (`syslogd`) starts, it reads */etc/syslog.conf* to find out how to treat any incoming messages. The file consists of records of the form

```
selector action
```

where *`selector`* specifies the list of error messages you want, and *`action`* tells `syslogd` what to do with them.

The basic form of the *`selector`* field is:

```
facility.severity
```

where the facility section is a comma-separated list of the facilities you want from the list in Table A, and *`severity`* specifies the minimum severity level on which you want to take action. So if the *`selector`* field happens to be

```
mail,news:err
```

then `syslogd` will perform the specified action whenever either the mail or news subsystem submits a log message with emerg, alert, crit, and/or err severity levels.

You can also connect multiple *`selector`* fields with a semicolon, so that you can perform the same action with multiple types of log messages. If you want to perform the same action with critical kernel errors

as you do with the mail and news errors, you can use this `selector` field:

```
mail,news:err;kernel.crit
```

The */etc/syslog.conf* file offers a convenient shorthand: You can use an asterisk (*) as a facility name to specify all facilities except the mark facility. (This helps prevent the log file from growing too quickly.)

The ***action*** field specifies where to send the message. You may specify a filename to store the message, the name of another computer, or a list of users to notify. If you want to specify a filename, it must start with a forward slash (/). Give the full path to the file where you want to store the messages. Store all mail and news errors in */etc/mail+news.errs* with the line

```
mail,news:err                /etc/mail+news.errs
```

If you want messages forwarded to another computer for storage, specify that computer by preceding the name with an at (@) symbol. To send all kernel messages to the host named sd_log, you can use the line

```
kernel.debug                 @sd_log
```

If you want to specify one or more users, just list their names. To specify all users, use an asterisk (*). Please note that only users who are actually logged in will be notified. If you want everyone to be notified of a critical error, and the operator to be notified on any emergency messages and line printer notices, you could use these lines:

```
*.crit                       *
*.emerg;lpr.notice           operator
```

Whenever you change the configuration file, be sure to tell `syslogd` to reread it by sending it the `HUP` signal, like this:

```
# ps -e ¦ grep syslog
   184 ?       0:01 syslogd
# kill -HUP 184
```

Logging errors in your scripts

Since a centralized facility exists to log error messages, you may as well use it for your own programs and scripts. This way, you can track the usage and any errors in your scripts and programs. For script files, it's very easy to use the system-logging facility—the `logger` command sends a message to the logger for you with the following line:

```
$ logger -p facility.severity message
```

Just replace ***facility*** with the facility you want to log to and ***severity*** with the error level appropriate to the message. The remainder of the line contains the message to send to ***syslogd***.

Quick Tip: The local0 through local7 facilities are reserved for site-specific uses, but there may not be enough facilities for all your purposes. In order to extend the range, you may want to group your programs into categories and put a prefix on each message body that specifies the actual script or program that generated the message.

For example, we might use local0 for various disk-administration scripts, local1 for user-administration scripts, etc. If the script named *FindDups* detects an error, it might post a message like this one:

```
logger -p local0.err FindDups: Can\'t allocate
➥space on /tmp
```

If you're writing programs rather than scripts, you should read the `man` page (section 3) for the `openlog()`, `syslog()`, `closelog()`, and `setlogmask()` functions. Briefly, you start using the logging service in your program with `openlog()`, in which you tell it the name of the process that's sending the message and the facility. You can also specify some options (described in */usr/include/sys/syslog.h*) with the syntax

```
void openlog(char *name, int options, int
➥facility)
```

where `name` specifies the name of the process, `options` contains the list of options, and `facility` is the facility ID that you want to use in your program.

When you actually want to write a message to the logging service, you use the `syslog()` function, where you specify the severity and the text string containing the log message. The syntax of the `syslog()` function is as follows:

```
void syslog(int severity, char *message)
```

The `setlogmask()` allows you to tell the logging facility which messages to process and which to discard. This is a great feature because

it allows you to log plenty of information when you're debugging a program. Then, when your program is working, you can deactivate some of your debugging messages without changing any code. Later, when you make improvements, you can bring back the debugging messages—again without changing your program. The syntax for `setlogmask()` is

```
void setlogmask(int severity)
```

The `setlogmask()` function lets you set the minimum level of messages you want your program to emit. If you want to see debugging messages, call `setlogmask()` with the severity of `LOG_DEBUG`. Later, when your program works and you only want to emit warnings and more severe messages, call it with `LOG_WARNING`. Please note that the values `LOG_DEBUG`, etc., are defined in the include file (*/usr/include/sys/syslog.h*).

When your program is finished, you can close the logging service with a call to `closelog()`. Listing A shows a simple C program that shows how you can use these functions in your programs.

Listing A: *LogDemo.c*

```
/* LogDemo.c - System logging demo */
#include <stdio.h>
#include <syslog.h>
int main(int argc, char *argv[])
  {
  int iLogLevel=LOG_WARNING;
  char acBuf[200];

  openlog("LogDemo", LOG_PID, LOG_LOCAL0);
  if ( argc > 1 ) iLogLevel = atoi( argv[1]);
  setlogmask(iLogLevel);

  syslog(LOG_EMERG,"emergency");
  syslog(LOG_ALERT,"alert");
  syslog(LOG_CRIT,"critical");
  syslog(LOG_ERR,"error");
  syslog(LOG_WARNING,"warning");
  syslog(LOG_NOTICE,"notice");
  syslog(LOG_INFO,"info";
  syslog(LOG_DEBUG,"debug");

  closelog();
  return 0;
  }
```

Conclusion

Since Solaris provides a standard method for managing logging messages, you ought to become familiar with it. Not only do many of the programs you work with use the logging service, but you can also press it to your advantage in your own scripts and programs. ❖

QUICK TIP

Taking advantage of the directory sort order

By now, you've no doubt noticed that when you list a directory, `ls` arranges the names in a particular order. You can use `ls` to simplify managing file collections by using particular prefixes or suffixes for filenames.

The `ls` sorting method puts numerals before uppercase letters, which appear before lowercase. Various punctuation symbols are mixed throughout the list. (Since the sort order varies by locale, you might want to verify the sort order, or you can experiment with filenames and investigate on your own.)

If you want to place a file near the beginning of a directory listing so that it will be noticed immediately, you may want to prefix it with a ! symbol. A file named *!README!*, for example, will appear near the top of the list. We often use this trick in the */etc* and */etc/rc?.d* directories. We prefix a disabled script or configuration file with *!DIS*, so that such files will be grouped together at the front of the directory listing.

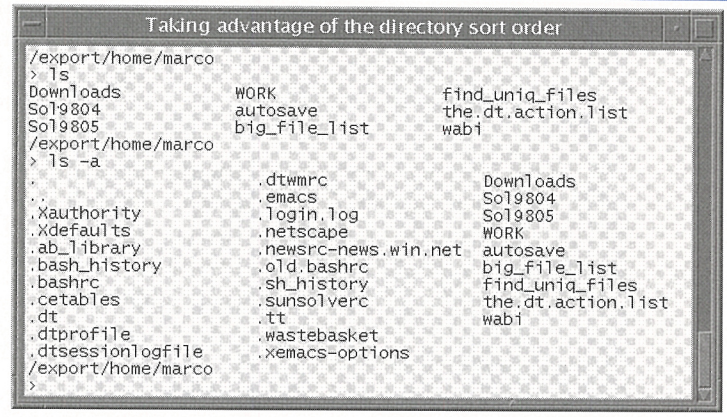
If you write programs that generate log files, you can name your log files starting with the date, such as *971225.logfile*. This way, directory listings will show your log files in chronological order. Alternatively, you can place the date at the end of your log file, so

files will be grouped by the application that generated them.

Don't forget that using a period (.) as a prefix normally makes the file invisible: It will appear in directory listings only when you use the `-a` option. This is a handy way to keep particular files out of the way, so they don't clutter your directory listings. Just take a look at Figure A—even though this is a small directory, which one would *you* want to look through to find a file?

There are many ways to take advantage of a sorted directory listing. Since `ls` goes to the trouble of sorting your directory listings for you, you may as well find ways to use it to make your life a little simpler! ❖

Figure A



Name little-used files and directories with a period (.) prefix to minimize clutter in your directory listings.

FILE MANAGEMENT

Splitting large files for distribution

If some of your offices aren't connected to the Internet, you probably know what a hassle it is to send files from one location to another on floppies. But what happens when your files become too large to fit on a floppy disk?

One option is to use a magnetic tape similar to those you use for backing up your system. However, tapes can be expensive (which is especially bad if the other office delays returning them). Also, this option mandates that all the sites where you distribute files must have the same type of tape drive available.

On the other hand, maybe the other office *is* on the Internet, but the mailer is set to pass only text files of a particular size or smaller. In this case, you must be sure that your file is in ASCII format and that it meets the size requirements.

In either case, it would be nice to be able to break up your file into multiple chunks. Well, Solaris provides a command that's ideal for this purpose—the `split` command. This command can break up files into whatever size you select and will sequentially name the resulting files so that you can conveniently reassemble them.

Convert files to ASCII and split them

The `split` command will break your files into chunks based on the number of lines or on the particular size you select. If you'd like to `split` a file into chunks 5,000 lines long, you'd use this form of the command:

```
$ split -5000 BigFile
```

The `-5000` parameter tells `split` to break the file into 5,000 line chunks. However, you can specify fixed-size pieces like this:

```
$ split -b 10k BigFile
```

Here, we told `split` to break the file into 10KB pieces with the `-b 10k` switch. If you use `m` instead of `k`, you specify megabytes; if you use neither, the result will indicate the chunk size in bytes. If you don't specify the chunk size with either method, then `split` defaults to a chunk size of 1,000 lines.

Quick Tip: You'll want to keep in mind that a line count really has meaning only for a text file. In a binary file, a line may be arbitrarily long, because the newline characters may appear *anywhere* in the binary file—if they appear at all! If you're going to split a binary file, you should use the -b method.

If you need the file in ASCII form, though, you'll want to use `uuencode` to change your file

into ASCII characters; then you can `split` the file. In this case, you can use the line-count method. You can `uuencode` a file like this:

```
$ uuencode InFile DestFile >OutFile
```

InFile specifies the name of the file that you want to convert to ASCII, *DestFile* specifies the name you want the file to be called *on the remote system*, and *OutFile* is the name you want to call the ASCII file on your own system. To reverse the process, use `uudecode` to convert the ASCII file back to binary, with the line

```
$ uudecode FileName
```

Splitting your files for floppy drives

If you're using a 1.44MB floppy disk drive, you'll want to split your files into 1.4MB chunks. Suppose that you want to send the file *BigFile* to one of your sites, but it's too large (at about 4MB) to fit on a floppy. You can `split` the file with the command:

```
$ split -b 1400k BigFile
$ ls -ls
total 15968
7968 BigFile
2816 xaa
2816 xab
2368 xac
```

Since we didn't specify a base name, the `split` command just uses `x`, then it adds the sequence ID (`aa`, `ab`, `ac`, etc.) to each chunk that's been split off until it reaches the end of the file. If you don't like to use `x` as the base filename, you can specify anything you want by adding it to the end of the command line, using the following code:

```
$ split -b 1400k BigFile BigFile_chunk.
$ ls -ls
total 15968
7968 BigFile
2816 BigFile_chunk.aa
2816 BigFile_chunk.ab
2368 BigFile_chunk.ac
```

Now you can copy the smaller files to floppy disks and send them to the other offices.

Reassembling the file

Once the recipient gets the files, it's a simple matter to reassemble them for use. Simply use the `cat` command to combine them, like this:

```
$ cat BigFile_chunk.aa BigFile_chunk.ab
BigFile_chunk.ac >BigFile
```

This `cat` command sends the output of each chunk, in the specified order, to `BigFile`. However, you needn't type all the filenames: When you use wild cards, your shell automatically sorts the matching names. Since you want all the chunks to be assembled in the same order, you can just use this command:

```
$ cat BigFile_chunk.* >BigFile
```

This method is a lot simpler, especially when you're contending with multiple chunks. ❖

BEGINNER'S TIP

Using the su command for quick problem diagnosis

Has this ever happened to you? You're on your way to troubleshoot a problem on a machine in the network you administer. As you approach the problem computer, half a dozen people stop you with questions about their own systems. Before you know it, you've spent two hours trying to fix a problem that should have taken 10 minutes.

Fortunately, Solaris offers a solution that allows you to attend to the needs of your users without exposing yourself to the dangers of hallway questioners. In many instances, you can fix client machines by remote control. In this article, we'll show you a way to diagnose and correct problems in client machines from the security and anonymity of your office.

Rise and shine

It's much easier to schedule time and stick to your priorities if you can solve problems from your own desk. First, you might try to `rlogin` to the user's machine to see if you can see the problem. Since many problems are centered around file permissions, you may have to log into the specific account to see the problem:

```
/root> rlogin pinky -l user
Password:
```

Uh-oh! You don't know the user's password, and you shouldn't ask for it, as it could compromise security. What should you do?

Bypassing passwords

One way you can avoid the necessity for using a password is to either change or remove the password altogether. However, when you're finished with your task, the user must then select a new password for the account, so removing the password isn't the easiest solution.

Fortunately, as a system administrator, you don't *need* others' passwords to log on to the system. First, go ahead and `rlogin` into the machine as yourself. As you may know, if you then `su` to the user's account, `su` will ask for a password, as follows:

```
/root> rlogin pinky -l user
Password:
Last login: Mon Jan  5 11:06:06 from Zappa
bash$ su user
Password:
```

As a matter of fact, you can easily get into the user's account *without* bothering with a password by using `su` from the root account to any other account. So, `su` to the root account, *then* `su` to the user's account with these lines of code:

```
bash$ su root
Password:
# su user
#
```

If the user's problem deals only with permissions, you may be able to reproduce the problem and find the solution. However, you might be unable to reproduce the problem because you haven't run the program as the other user did. The missing piece of the puzzle is the state of the environment variables, such as `PATH`.

The su command's -l option

You can avoid this headache by adding the `-l` option to the `su` command. The `-l` option tells the `su` command to act as if you were the specified user just logging into the system. In this case, `su` starts a new login shell for the user's account. The new shell then runs the shell startup scripts, configures the environment variables, etc., just as though you were the specified user. As a result, when you try to reproduce a problem reported by a user, you'll have a much simpler time doing so.

```
bash$ su root
Password:
# su user
Hello user!
Devo:~>
```

Conclusion

Administering a network can be difficult enough without having to run around the building solving various and sundry user problems. By using `rlogin` in conjunction with the `su -l` command, you should be able to rectify many problems without leaving your office. ❖

PPP PERFORMANCE TIP

Speeding up your PPP link

If you're using Solaris' PPP facility, you can probably squeeze a little extra performance out of the link. By properly setting the *ipcp_async_map* for your communications paths, you can transmit the same amount of information over the network in fewer bytes, thereby shaving some time off your file transfers.

So what is ipcp_async_map?

The PPP configuration file (*/etc/asppp.cf*) allows you to specify the characteristics for each communications pathway you'll use for PPP. One of the parameters you can specify is the obscure *ipcp_async_map*. Reading the `man` page for *aspppd* doesn't shed much light on the

purpose of this parameter, as you can see in Figure A. After some digging, we discovered what this option means.

Figure A

```
ipcp_async_map hex-number
    Specifies the async control character map
    for the current path. The hex-number is
    the natural (i.e., big endian) form repre-
    sentation of the four octets that comprise
    the map. The default value is ffffffff.
```

This excerpt from aspppd's man page doesn't really tell us how and when to use the ipcp_async_map *option.*

First, let's look at some background information. As you know, a byte can hold 256 values. You probably also know that the first 32 bytes are often used to represent control characters in the ASCII character set, e.g., backspace, carriage return, escape, newline, and tab.

It turns out that the 32-bit number set by *ipcp_async_map* tells the PPP link which control characters to treat specially. If a bit is on, then PPP must treat the corresponding control character in a special way—where the lowest order bit maps to ^@, the next bit maps to ^A, then ^B, etc. Since the default value is 0xffffffff (i.e., all bits are on), all the control characters receive special treatment.

Just what special treatment does the PPP link give to a control character? When PPP is told to treat a control character specially, it doesn't send that character at all. Instead, PPP sends a two-byte sequence that represents the character. So, if you're sending a file where all byte values are equally distributed, then you'll send about 12 percent more data than you want. (32 possible byte values are sent as pairs of bytes, rather than single bytes.)

Why give a character special treatment?

Ideally, you *wouldn't* treat the control characters in a special way. In a perfect communications path, equipment would simply pass all bytes straight through to their destination. However, you must take care with control characters because some equipment in a communications path between the client and host computer may perform one or more special actions on receipt of particular control characters.

Unfortunately, we live in a less-than-perfect world. Some terminal controllers, when they receive a ^O, will flush the data in the buffer. If you use a modem with a cheap cable and are forced to use software handshaking, then you'll use ^S and ^Q to stop and restart communications. (You'd save yourself untold headaches by turning off software handshaking and using a good cable with your modem instead.)

Table A: *Control characters*

Hex Code	Symbol	ASCII Name	Description
00	^@	NUL	
01	^A	SOH	Start of header
02	^B	STX	Start text
03	^C	ETX	End text
04	^D	EOT	End of transmission
05	^E	ENQ	Enquire
06	^F	ACK	Acknowledge
07	^G	BEL	Bell
08	^H	BS	Backspace
09	^I	HT	Tab
0a	^J	LF	Linefeed (\n)
0b	^K	VT	Vertical tab
0c	^L	FF	Formfeed
0d	^M	CR	Carriage return (\r)
0e	^N	SO	Shift out
0f	^O	SI	Shift in
10	^P	DLE	Data link escape
11	^Q	DC1	Device control 1 (XON, resume xmit)
12	^R	DC2	Device control 2
13	^S	DC3	Device control 3 (XOFF, pause xmit)
14	^T	DC4	Device control 1
15	^U	NAK	Negative acknowledge
16	^V	SYN	Synchronous idle
17	^W	ETB	End transmission block
18	^X	CAN	Cancel
19	^Y	EM	End of medium
1a	^Z	SUB	Substitute
1b	^[	ESC	Escape
1c	^\	FS	File separator
1d	^]	GS	Group separator
1e	^^	RS	Record separator
1f	^_	US	Unit separator

Setting ipcp_async_map

If you're using a perfect communications pathway, then you can set *ipcp_async_map* to 0 so none of the control characters receive special treatment. This can shave more than 10 percent off your file transfer times on your PPP link. (You probably won't attain that result, since other sources of communications delays will also affect your communications.)

If you know which control characters are treated specially in your communications path, you can still save some time on your transfers by telling the PPP facility to treat only those characters specially. Let's suppose that you're using software handshaking (i.e., ^S and ^Q are special) on your modem, and your ISP is using a terminal controller in which ^O flushes the buffer. Now we just find out which bits correspond to ^O, ^Q, and ^S and set only those bits. Table A provides a list of all the control characters.

At this point, all we need to do is build a hex number with bits 0x0f, 0x11, and 0x13 on, and the rest off. Since the bits are numbered from right to left, we come up with a binary number, which we then convert to hex. To make the process simpler to see, the first line shows the control character symbol, and the next line shows the binary number we made by setting ^O, ^Q, and ^S to one, and setting the rest to zero. The third line shows the hexadecimal representation of the number.

```
_^]\ [ZYX WVUT SRQP ONML KJIH GFED CBA@
0000 0000 0000 1010 1000 0000 0000 0000
  0    0    0    A    8    0    0    0
```

We arranged the number in groups of four bits since it simplifies converting from binary to hexadecimal. To convert back and forth, use Table B. Now we can edit the appropriate path block in */etc/asppp.cf* to set the *ipcp_async_map* option to our newly computed value, as shown in Figure B.

Don't hit the books quite yet...

Since Sun has no idea what communications equipment lies between your host and your ISP's host, it sets *ipcp_async_map* to a conservative value that won't fail. While this saves Sun a lot of service calls, this preset value costs us time each time we transfer binary files over our PPP link.

Table B: *Converting back and forth from binary to hexadecimal*

Hex	Binary	Hex	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Figure B

Speeding up your PPP link

```
# /etc/asppp.cf - Dialup PPP link configuration

#####
# Dialout to WinNET
#####

ifconfig ipdptp0 plumb 198.70.147.210 up metric 1

path
   interface            ipdptp0          # We connect via /dev/ipdptp0
   peer_system_name     WinNET           # /etc/uucp/Systems record key
   ipcp_async_map       a8000            # Escape ^O, ^Q, and ^S
```

We modified this path block to specify an ipcp_async_map *value of a8000.*

If your setup is relatively standard, then you shouldn't need special treatment on any control characters. So, before digging out all the manuals on your equipment, go ahead and try 0, and send a binary file or two over your PPP link. Be sure that your test file has every possible character in it. If you can successfully transmit this file, then you're finished.

If your file fails to transmit, *then* hit the books. You can read the manuals to ascertion which control characters must be treated specially, or you can figure it out by trial and error.

Conclusion

Since the default value of *ipcp_async_map* is so conservative, there's an opportunity for you to improve the performance of your PPP link when you transmit binary files. Text files tend to use few control characters, and those (tab, carriage return, newline, formfeed, tab) are so commonly-used in text that communications equipment shouldn't require any special handling for them. Thus, you can achieve a modest performance gain by turning off just those control characters. ❖

Monitoring an active log file

Many programs write log files to assist with the debugging process. So, when you're installing a system, you run it and examine the log files to try to find out what went wrong. Did you know that you can examine the contents of a logging file, even while it's in use? What's more, you can feed those contents to another program for processing.

What's the secret?

As long as you have read permissions on a file, you can read a file that's still open for writing by another process. Since many programs write data to log files, you needn't wait for the program to finish running before you start reading it. However, once the program that's reading the file encounters the end of the file, it will then terminate—unless it's specially written.

It turns out, though, that you needn't write your own special program. Solaris provides an option (`-f`) on the `tail` command that will, when it reaches the end of the file, wait a little while, then look for more input. The `tail` command will stay in this loop until you explicitly terminate it.

As an example, when we were working on the PPP article, we needed to watch */etc/log/asppp.log* to find out the source of dialing problems, IP negotiation, etc. To do so, we used the simple command

```
# tail -f /etc/log/asppp.log
```

This way, we could see the logging information as it's written.

Processing the log

You can use this trick to allow other programs to work on the log file, too. Since the tail program prints its output on the standard output, you can use it to start a command pipeline. For example, when we were looking over the IP negotiation phase, we didn't want to look at all the other messages. So we used the command

```
# tail -f /etc/log/asppp.log | grep IP_NCP
```

Since we chose this method, `grep` could throw away all messages that didn't involve the IP_NCP (IP negotiation) protocol.

Monitoring multiple streams

Have you ever wanted to watch both the standard output and standard error streams at the same time? If so, you probably resorted to something like the line

```
# your_program 2>&1
```

which tells the Korn or Bourne shell to send the standard error stream (2) to the same place as the standard output stream (1).

However, since this mixes the streams, the output can get confused. If you would like to see both streams separately in different windows, then in one window, you can run your program piping the error stream to a file with this line:

```
# your_program 2>std.err.out
```

In another window, you can view the error stream using the following `tail -f` command:

```
# tail -f std.err.out
```

Conclusion

The next time you're installing a program that generates log files, try the trick we just described. Real-time feedback can be a big help in debugging, because you can see what's happening on your system at the time the log message is generated. The ability to then pipe the results through other programs for processing is just the icing on the cake! ❖

Are you a good tipper?

Do you have any great Solaris tips that you've discovered? If so, send them our way! If we use your tip, it will appear on our weekly online ZDTips service. (Visit www.zdtips.com to check out all our available tip services.) We may also publish it here in Inside Solaris. Your byline will appear with the tip, along with your E-mail and/or Web addresses.

Send your tips to inside_solaris@zd.com, fax them to "Solaris tips" at (502) 491-4200, or mail them to

Inside Solaris
The Cobb Group
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Dial-on-demand connections with aspppd

Internet connectivity seems to become more important each day, with most companies requiring a presence on the Internet. Small companies may choose to have an Internet Service Provider (ISP) host a Web site for them so they can transact business over the Web. Large companies might use leased lines to supply a high-speed, full-time connection to the Internet. And of course, there's the large gray area between having no connection and maintaining a full-time connection.

Your company may need to establish a connection to the Internet. Perhaps you must upgrade your E-mail services so that you get frequent mail delivery. Or maybe the vice president wishes to browse the Web in order to check on stock prices during lunch.

Using a full-time connection via a leased line is expensive. However, you can get high-speed access at any time, day or night. You'll have to pay for the leased line, the equipment you'll use on your premises, and (via the setup and monthly charges) the equipment on your ISP's site.

Part-time connectivity

For many smaller companies, there just isn't enough benefit to justify the expense of a leased line and the requisite extra equipment. In such cases, you must find lower-cost alternatives. One alternative is to use a standard modem to connect to your ISP only during certain times of the day. This way, you can use a standard telephone line and basic modem, which is a cost-effective solution. This solution is especially tempting, since every copy of Solaris ships with a dialup PPP facility built-in.

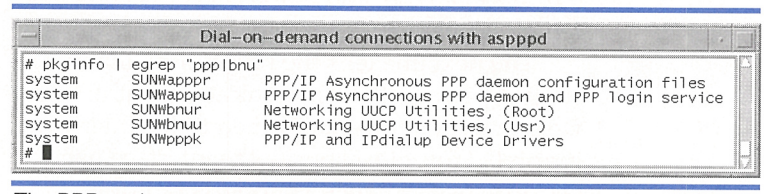
In this article, we're going to show you two methods in which you can use the dialup PPP facility for a part-time connection. The first method uses a static IP address with automatic dial-on-demand, and the second uses a dynamic IP address with manual connect and disconnect.

However, both methods are based on the standard PPP installation, so before we show you how to create a part-time connection, let's briefly review basic PPP setup for dial-out connections. (Since we covered this in the article "Configuring A Remote PPP Dial-Up Client For Solaris 2.X" in our February 1996 issue, we'll skip over some of the details.)

Basic PPP configuration review

First, you must ensure that you have all the required packages, which are listed in Figure A, installed on your machine. Please note that the uucp packages (SUNWbnur, SUNWbnuu) are required because the PPP connection is established with the standard uucp dialer services. The first line of Figure A shows the command you use to determine whether the required packages are installed. If they're not, you'll need to install them from your distribution CD.

Figure A



The PPP package requires these five packages.

Once you have these packages installed, you must configure your serial port and modem to communicate properly with one another. If you use an external modem, be sure you use a quality cable, because some cheap cables omit some of the modem signals, making it difficult to maintain a good connection. (This topic is complex enough to warrant its own article, so if you have any difficulty configuring your serial ports or modem, then read Stokely Consulting's excellent Web page on the subject "Celeste's Tutorial on Solaris 2.x Modems and Terminals," which you'll find at www.stokely.com.)

Quick Tip: We advise you to use an external modem, so you'll be able to reset it easily by turning it off and on. While you learn about serial ports, it's possible to get the modem stuck in a state from which it's difficult to recover. Internal modems, while cheaper, are more difficult to reset since you must turn off your computer. To do so until you gain enough experience with serial port and modem configuration, you should use an external modem.

Now we're ready to configure the files. For the basic PPP configuration, we need to configure only three files. The file */etc/asppp.cf* configures the PPP configuration; */etc/uucp/Systems* configures the chat script for dialing the modem; and */etc/uucp/Devices* configures the modem you'll use to connect to the modem. You'll also want to configure */etc/gateways* to prevent routing information from consuming bandwidth on your PPP link.

Configuring /etc/asppp.cf

Figure B shows a typical */etc/asppp.cf* file that we're using to connect to our ISP. The line starting with `ifconfig` specifies how to set up the interface. Specifically, we're telling `ifconfig` to set up device ipdptp0 (the dialup link) using PPP address 198.70.147.210, that the link is up (i.e., it's enabled), and that it's a connection to a remote machine.

The next line tells the PPP facility that we're starting a new path (communications path). First, we specify that this path is tied to the interface ipdptp0. (You must specify the interface name because the PPP facility supports multiple incoming and outgoing paths—this way you can tie the path to the right `ifconfig` statement.) Then, we specify the name of the record in the */etc/uucp/Systems* file that we use to log into our ISP. Since we use WinNET for our ISP, we'll call the record *WinNET*.

When you create the `ifconfig` command, you must select an IP address to tie the interface. If you're using a static IP address, you'll need to specify the address your ISP gives you. If you're using a dynamic IP address, you can use any value (other than 0.0.0.0) because, during login, it will change to the value that the ISP gives it for that login.

Configuring /etc/uucp/Systems

Now, let's add the WinNET record to our */etc/uucp/Systems* file. To do so, we simply add the following line to the end of the file. You'll have to change this record because the sequence of operations used to log into your ISP will be different from ours. At the very least, you must change telephone, user name, and password to reflect those that you use with your ISP with the line

```
WinNET Any ACU 38400 telephone "" P_ZERO
 "" \r\n gin: username  ord: password
```

Please note that the previous code is all one line. The first word, WinNET, is the record name. Any refers to the day and time the record may be used, meaning any day or time. The following field is the device type, where ACU specifies an *Automatic Call Unit*, which is what you'll probably need. You can refer to the file */etc/uucp/Devices* for information about other device types that are available. Next comes the baud rate you want to use for the connection followed by the telephone number to call.

The rest of the line is the `uucp` chat script that logs you into the ISP. The basic structure of the chat script is a set of expect/send pairs. First you specify what you're expecting to read from the modem. Once it's read, the script sends the send value. The process repeats until it reaches the end of the last pair. If the modem is still connected at the end of this sequence, then the hosts are connected. (Please note that the PPP layer hasn't yet been established.)

We're expecting an empty string ("") from the modem, which means that it will immediately send the next part, P_ZERO, to the serial port. This is actually a special value that tells `uucp` to program the serial port to use eight data bits with no parity, which is the setup required to get a PPP link working. (Similarly, P_ONE sets parity to one, P_EVEN sets even parity, and P_ODD sets odd parity.)

Now we try to log into our ISP. To do this, we need to send a carriage return and line feed so it will issue a login prompt for us. Again, we expect an empty string to force an unconditional send of a carriage return (`\r`) followed by a linefeed (`\n`).

Once our ISP receives the carriage return and line feed, it emits the login prompt "login:". We'll expect the phrase "gin:", and once we see it, we'll send our *username*. Then the ISP will prompt us for our password with the phrase "password:", so we'll expect "word:"; once we see it, we'll send our password. Now we should be logged in.

Figure B

Dial-on-demand connections with aspppd

```
# /etc/asppp.cf - Dialup PPP link configuration

#####
# Dialout to WinNET
#####

ifconfig ipdptp0 plumb 198.70.147.210 up metric 1

path
   interface            ipdptp0          # We connect via /dev/ipdptp0
   peer_system_name     WinNET           # /etc/uucp/Systems record key
```

The basic configuration has no frills, but it will get the job done.

Configuring /etc/uucp/Devices

The last file we must configure is named */etc/uucp/Devices*. You use this file to tell `uucp` which serial port is connected to the modem you want to use to establish the link. This file contains entries in the form

```
Type  Port  Unused  Speed  Dialer
```

In this line, *Type* specifies the type of connection you're using. Since you're connecting through a modem, use ACU. The *Port* parameter specifies the name of the device, without the directory prefix. Solaris doesn't use the *Unused* field, so just insert a hyphen (-). You specify the speed of your connection to the modem with the *Speed* word. Finally, with *Dialer* you specify the modem type. You must look at the file */etc/uucp/Dialers* for a complete list of all the legal *Dialers*.

At our site, we're using a Hayes-compatible modem on port */dev/cua/b* with a connection speed of 38400 bps, so our entry is

```
ACU cua/b - 38400 hayes
```

Please note that the speed of the connection to the modem is independent of the modem's baud rate. Your computer communicates to the modem over a serial cable at one speed, the speed as specified by *Speed*. The modem talks to the modem on the other end of the phone line at another speed, the connection speed. For example, a 14.4Kbps modem sends 14,400 bits per second over the phone lines. In this case, you might choose a connection speed of 19,200Bps.

If the modem uses compression, it may actually send data over the line at an effectively higher rate, perhaps 20Kbps. In order to accommodate this higher rate, you'll want to select a faster connection speed. In general, it's best to choose the highest connection speed the modem supports.

Configuring /etc/gateways

If your system has another network interface on it (as most do), then it will act as a router when connected to the Internet. However, you don't want your machine to continually exchange routing information with your ISP. If it does, then you'll waste bandwidth over your PPP channel (as well as CPU time) for the routing information.

If you plan to use the dial-on-demand configuration, then you'll encounter yet another problem: Routers tend to exchange information frequently enough to keep your link active, even when no one's using it. So once the link activates, it will stay active even if no one's using it. To prevent this situation, add the line

```
norip ipdptp0
```

in the file */etc/gateways*. If the file doesn't exist, then create it and put this line in it.

Testing your connection

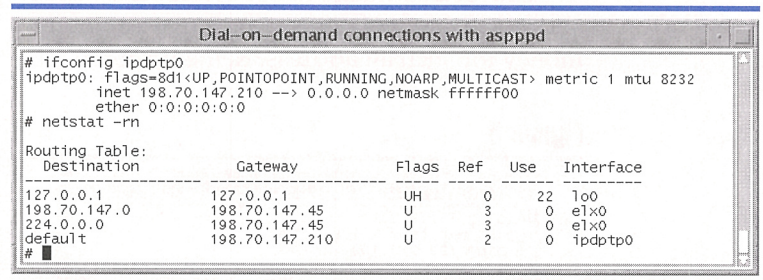
Now that you've set up your PPP link, you're ready to test it to see if it works. First, start `aspppd`, the PPP daemon. To do so, you can use the command

```
# /etc/init.d/asppp start
```

This command starts the `aspppd` daemon, which manages your PPP link. If it sees a valid configuration, it configures the PPP interface (ipdptp0) and enters it into the routing table. You can examine the port configuration and routing table to see if the basic configuration is correct. Figure C shows how to do this—the `ifconfig` command displays the setup for ipdptp0, if it exists. If the command didn't work, you'd see a "no such interface" error.

We can use the `netstat` command to see whether the interface was properly registered into the router table. Here, you're looking for a default route to the ipdptp0, as shown in the last line in Figure C.

Figure C



You can use the ifconfig and netstat commands to check whether your configuration is reasonable.

If your configuration looks valid, it's time to test your phone number and `uucp` chat script. To start the connection, just `ping` an address outside of your network. Since the default route goes to ipdptp0, the `aspppd` daemon will bring up the PPP link for you to try to route the packet.

Please note that establishing the connection may take a long time, as far as your software is concerned, so the first access to the Internet may fail. For example, in our test setup, we're using the `ping` command to bring up the link; it times out in about 20 seconds with a "no answer" error. The connection takes about 40 seconds to establish. If we wait a little while and try the command again, it succeeds, as shown in Figure D. Now you can terminate the connection using the command

```
# /etc/init.d/asppp stop
```

At this point, you're ready to move on to the next step. If you're going to use a dial-on-demand configuration with a static IP address, proceed to the next section. For a manually controlled connection with dynamic IP addresses, skip to the Manual Connections section.

Using a dial-on-demand with a Static IP address

The easiest way to use the Internet is with a static IP address. The main advantage is that you needn't do anything special to make and break the connection. All you really need is a couple of minor configuration changes. Then, whenever you attempt to access something that's not local to your network, `aspppd` will establish the PPP connection with your ISP. After a (configurable) period of inactivity, `aspppd` automatically breaks the link.

One disadvantage to using a static IP address is that most ISPs will charge you more money for such an address. Remember, the provider can't allow others to use the address when you're not connected.

In order to configure the PPP facility to use a static IP address, make two changes to the */etc/asppp.cf* file. First, specify the IP address your ISP gives you in the `ifconfig` statement. Then, add the following statement to your path (just add this line after the line that specifies the peer_system_name):

```
inactivity_timeout 300
```

This line tells `aspppd` to disconnect the PPP link after 300 seconds (five minutes) of inactivity. You can change the timeout length to anything you prefer.

Now you're ready to use your PPP link. Whenever you attempt to access an address that's outside your network, `aspppd` will start the PPP link, if required. After the link remains idle for a long enough time, the link will stop. You needn't think any more about it.

Manual connections

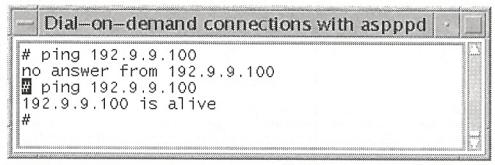
If you don't want to spend the extra monthly cost for a static IP address, then you must use dynamic IP addressing. In this scheme, the ISP can save money because it provides you with an IP address from a shared pool each time you log in. Since everyone shares this pool of IP addresses, the ISP needs fewer lines than would otherwise be required. However, this shared pool mandates that we configure our machine slightly differently. It also forces us to manually start and stop the link.

In order to accommodate dynamic IP addresses, you must add the following line to the path in your */etc/asppp.cf* file. This line tells `aspppd` that it should accept whatever IP address the ISP offers:

```
negotiate_address  on
```

We don't try to use dial-on-demand on a dynamically allocated IP address because the routing table changes after the connection starts. Thus, we can't establish the correct default route for the PPP interface. Even if we could, an application that opens a persistent connection may fail since the IP address may be different on each login. (The PPP connection could terminate and later restart because the connection might open with a different IP address.)

Figure D



Be patient when starting the link, since it must dial the phone, build the PPP connection, establish routes, etc.

Therefore, we must rely on manual connection control instead. Since we're not specifying the `inactivity_timeout` parameter, the `aspppd` daemon starts the link whenever it starts and tries to keep the link active. When you install the `SUNWapppr` package, it installs the script `asppp` in the */etc/init.d* directory. Thus, you can start the link with this command:

```
# /etc/init.d/asppp start
```

Similarly, you can terminate the link with the following command:

```
# /etc/init.d/asppp stop
```

The installation procedure also creates links to the `asppp` script as */etc/rc0.d/K47asppp*, */etc/rc1.d/K47asppp*, and */etc/rc2.d/S47asppp*. Because you always want to shut down the link when you're stopping the system, you'll probably want to leave the first two links alone. However, the third symbolic link tells Solaris to start the `aspppd` daemon when it boots into multiuser mode. Since you may not want to start the PPP link every time you boot your computer, you can disable */etc/rc2.d/S47asppp*.

Quick Tip: When you want to disable a script in the */etc/rc?.d* directories, just rename it to something that doesn't start with an S or a K. (We typically just add the prefix _DIS to segregate disabled files near the front of the directory as we described in "Taking Advantage of the Directory Sort Order.") In these directories, any script beginning with an S is automatically started, and any script starting with a K is automatically stopped when the system goes to the appropriate run level.

Keeping the link alive

One disadvantage of using dynamic IP addresses on your PPP link is that when your PPP link drops unexpectedly (such as when the link is inactive too long), an application won't know that the link has died. Then when `aspppd` restarts the link, chances are good that you'll have a different IP address. If the application is working under the assumption that the link will have the same IP address and attempts to communicate with the remote computer, it will probably fail. An example of this situation occurs when you're transferring a file. If the connection drops and comes back, the file transfer will hang.

Miscellaneous notes

When you're first configuring or otherwise experimenting with your PPP configuration, you should turn on the debugging log so you can see what the PPP connection is doing. The debugging log shows you what the link is doing, making it easier to track down any problems you may experience.

Table A: *Seven levels of debugging information*

Level	Meaning
0	Only log errors
1	Minimal connection information
4	Abbreviated uucp chat-script messages
5	Complete uucp chat-script messages
7	All uucp messages
8	Log PPP message tracing
9	Log the complete IP packets

With the debugging log enabled, the `aspppd` daemon writes debugging information to the log file */etc/log/asppp.log*. You can select from among seven levels of debugging information, as shown in Table A. You turn on the debugging log by adding the following statement to the path in your */etc/asppp.cf* file, where *x* is the level of debugging information you want:

```
debug_level x
```

Be sure to remove this line from the path when you're finished debugging, as this file can grow extremely quickly—especially at level 9.

Conclusion

If you occasionally need to connect your network to the Internet, but don't want to pay for a dedicated line, then a dial-on-demand PPP link with a static IP address is probably your best solution. For cost-conscious businesses, using a dynamically allocated IP address is definitely the cheapest way to go. Either way, with just a little work, you can create a part-time link to the Internet for your company. ❖

PERIODICALS MAIL

SunSoft Technical Support
(800) 786-7638

C:7461905 00002096 02/99
[redacted]
CLINTON TOWNSHIP MI 48035-4218

Please include account number from label with any correspondence.

UTILITIES

Using vi to reverse a file

Every once in a while you may need to reverse the order of lines in a file. For instance, you may be searching for data with a `find` and `grep` pipeline that you put through sort and into a file, only to discover that you should have added the *-r* flag to the sort command. Or you may have created a script to rename a set of files, changing *file00* through *file99* to *file01* through *file100*, only to realize just in time that you have the commands in the wrong order and could have ended up with one file that had been renamed 100 times, clobbering 99 others on the way. Unfortunately, there's no reverse utility that will quickly fix the problem.

Here's a simple trick with `vi` that will let you reverse the order of lines in a file. It also illustrates one of the important details of `vi`'s global search syntax.

To reverse a file within `vi`, simply type

```
:g/^/m0
```

In "spoken-`vi`," you might read the command as "Globally search for the beginning of each line, and for each line found, move it to immediately follow line 0." The seemingly similar command

```
:1,$m0
```

won't do what you want because it treats the entire block of lines as a single unit.

The first command works the way it does because a global search is actually a two-step process. First, `vi` searches the file for the pattern and marks each matching line. Second, `vi` steps through each marked line in turn, beginning at the top of the file, and applies the command to each marked line.

In this case, every line is marked because even empty lines will be matched by the /^/ pattern. Then, for each line, the command moves the current line (.) to line 0. When line 1 moves, it'll be in the same place it started. When line 2 moves, it will be above line 1. Line 3 will move up one line, and so on. You have reversed the lines in a file with fewer than 10 keystrokes.

Because the same global command we used above in `vi` will also work in `ed`, you can put the command into a *reverse* script, as shown in Figure A.

Figure A

```
#!/bin/sh --
#
# reverse - print to stdout all lines
#           in reverse order
#
PATH=/bin:/usr/bin

case $# in
   1)   ed - "$1" << "EOF"
g/^/m0
1,$p
EOF
        ;;
   *)   echo Usage: $0 file >&2
        exit 1
        ;;
esac
```

This script uses ed to reverse the order of lines in a file.

This command, like the `vi` example above, will reverse only a file that meets the limitation of `vi`'s and `ed`'s maximum file size. ❖

Printed in the USA.
This journal is printed on recyclable paper.