Error Handling

Netpbm Programming Library Errors

As part of Netpbm's mission to make writing graphics programs quick and easy, Netpbm recognizes that no programmer likes to deal with error conditions. Therefore, very few Netpbm programming library functions return error information. There are no return codes to check. If for some reason a function can't do what was asked of it, it doesn't return at all.

Netpbm's response to encountering an error is called "throwing an error."

The typical way a Netpbm function throws an error (for example, when you attempt to open a non-existent file with **pm_openr()**) is that the function writes an error message to the Standard Error file and then causes the program to terminate with an exit() system call. The function doesn't do any explicit cleanup, because everything a library function sets up gets cleaned up by normal process termination.

In many cases, that simply isn't acceptable. If you're calling Netpbm functions from inside a server program, you'd want the program to recognize that the immediate task failed, but keep running to do other work.

So as an alternative, you can replace that program exit with a longjmp instead. A longjmp is a classic Unix exception handling concept. See the documentation of the standard C library **setjmp()** and **longjmp()** functions.

In short, you identify a point in your programs for execution to hyperjump to from whatever depths of whatever functions it may be in at the time it detects an exception. That hyperjump is called a longjmp. The longjmp unwinds the stack and puts the program in the same state as if the subroutines had returned all the way up to the function that contains the jump point. A longjmp does not in itself undo things like memory allocations. But when you have a Netpbm function do a longjmp, it also cleans up everything it started.

To select this form of throwing an error, use the **pm_setjmpbuf()** function. This alternative is not available before Netpbm 10.27 (March 2005).

Issuing of the error message is a separate thing. Regardless of whether a library routine exits the program or executes a longjmp, it issues an error message first.

You can customize the error message behavior too. By default, a Netpbm function issues an error message by writing it to the Standard Error file, formatted into a single line with the program name prefixed. But you can register your own error message function to run instead with **pm_setErrorMsgFn()**.

pm_setjmpbuf()

pm_setjmpbuf() sets up the process so that when future calls to the Netpbm programming library throw an error, they execute a longjmp instead of causing the process to exit as they would by default.

This is not analogous to **setjmp()**. You do a setjmp() first, then tell the Netpbm programming library with **pm_setjmpbuf()** to use the result.

Example:

```
#include <setjmp.h>
#include <pam.h>
jmp_buf jmpbuf;
int rc;
rc = setjmp(jmpbuf);
if (rc == 0) {
  struct pam pam;
  pm_setjmpbuf(&jmpbuf);

  pnm_readpam(stdin, &pam, PAM_STRUCT_SIZE(tuple_type));
  printf("pnm_readpam() succeeded!0);
 } else {
  printf("pnm_readpam() failed.  You should have seen "
       "messages to Standard Error telling you why.0);
 }
```

This example should look really strange to you if you haven't read the documentation of **setjmp()**. Remember that there is a hyperjump such that the program is executing the **pnm_readpam()** and then suddenly is returning a second time from the setjmp()!

Even **pm_error()** works this way -- if you set up a longjmp with **pm_setjmpbuf()** and then call **pm_error()**, **pm_error()** will, after issuing your error message, execute the longjmp.

pm_setjmpbuf() was new in Netpbm 10.27 (March 2005). Before that, Netpbm programming library functions always throw an error by exiting the program.

User Detected Errors

The Netpbm programming library provides a function for you to throw an error explicitly: **pm_error()**. **pm_error()** does nothing but throw an error, and does so the same way any Netpbm library function you call would. **pm_error()** is more convenient than most standard C facilities for handling errors.

If you don't want to throw an error, but just want to issue an error message, use **pm_errormsg()**. It issues the message in the same way as **pm_error()** but returns normally instead of longjmping or exiting the program.

Note that **libnetpbm** distinguishes between an error message and an informational message (use **pm_errormsg()** for the former; **pm_message()** for the latter). The only practical difference is which user message function it calls. So if you don't register any user message function, you won't see any difference, but a program is still more maintainable and easier to read when you use the appropriate one of these.

pm_error()

Overview

void pm_error(char * *fmt***,**

Example

```
if (argc-1 < 3)
   pm_error("You must specify at least 3 arguments.  "
         "You specified" only %d", argc-1);
```

pm_error() is a **printf()** style routine that simply throws an error. It issues an error message exactly like **pm_errormsg()** would in the process.

pm_errormsg()

Overview

void pm_errormsg(char * *fmt***,**

Example

```
if (rc = -1)
   pm_errormsg("Could not open file.  errno=%d", errno);
   return -1;
```

pm_errormsg() is a **printf()** style routine that issues an error message. By default, it writes the message to Standard Error, but you can register a user error message routine to be called instead, and that might do something such as write the message into a log file. See **pm_setusererrormsgfn**.

There is very little advantage to using this over traditional C services, but it issues a message in the same way as **libnetpbm** library functions do, so the common handling might be valuable.

Note that the arguments specify the message text, not any formatting of it. Formatting is handled by **pm_errormsg()**. So don't put any newlines or tabs in it.

pm_setusererrormsgfn()

Overview

void pm_setusererrormsgfn(pm_usererrormsgfn * *function***);**

Example

```
   static pm_usererrormsgfn logfilewrite;
   static void
   logfilewrite(const char * const msg) {
      fprintf(myerrorlog, "Netpbm error: %s", msg);
   }
   pm_setusererrormsgfn(&logfilewrite);

   pm_errormsg("Message for the error log");
```

pm_setusererrormsg() registers a handler for error messages, called a user error message routine. Any library function that wants to issue an error message in the future will call that function with the message as an argument.

The argument the user error message routine gets is English text designed for human reading. It is just the text of the message; there is no attempt at formatting in it (so you won't see any newline or tab characters).

You can remove the user error message routine, so that the library issues future error messages in its default way (write to Standard Error) by specifying a null pointer for *function*.

The user error message routine does not handle informational messages. It handles only error messages. See **pm_setusermessagefn()**.

Error Handling In Netpbm Programs

Most Netpbm programs respond to encountering an error by issuing a message describing the error to the Standard Error file and then exiting with exit status 1.

Netpbm programs generally do not follow the Unix convention of very terse error messages. Conventional Unix programs produce error messages as if they had to pay by the word. Netpbm programs tend to give a complete description of the problem in human-parseable English. These messages are often many terminal

lines long.