User manual for old pbm functions

pbm Functions

Updated: 22 July 2004
Table Of Contents

NAME

libpbm - libnetpbm functions to read and write PBM image files

SYNOPSIS

#include <pbm.h>

bit **pbm_allocarray(int *cols*, **int** *rows*);

bit *pbm_allocrow(int *cols*);

pbm_freearray(bit ****bits*, **int *rows*);

**pbm_freerow(bit ***bitrow*);

void pbm_readpbminit(FILE * *fp*, **int ***colsP*, **int ***rowsP*, **int ***formatP*);

void pbm_readpbmrow(FILE * *fp*, **bit ***bitrow*, **int** *cols*, **int** *format*);

void pbm_readpbmrow_packed(FILE * *fp*,
unsigned char * const *packed_bits*, **const int** *cols*, **const int** *format*);

void bit pbm_readpbm(FILE *** *fp*, **int ***colsP*, **int ***rowsP*);

void pbm_writepbminit(FILE * *fp*, **int** *cols*, **int** *rows*, **int** *forceplain*);

void pbm_writepbmrow(FILE * *fp*, **bit ***bitrow*, **int** *cols*, **int** *forceplain*);

void pbm_writepbmrow_packed(FILE * *fp*,
unsigned char * const *packed_bits*, **const int** *cols*, **const int** *forceplain*);

void pbm_writepbm(FILE * *fp*, **bit ****bits*, **int** *cols*, **int** *rows*, **int** *forceplain*);

#define pbm_packed_bytes(*cols***) ...**

void pbm_nextimage(FILE ***file*, **int * const *eofP*);

void pbm_check(FILE * *file*, **const enum pm_check_type** *check_type*, **const int** *format*, **const int** *cols*,
const int *rows*, **enum pm_check_code * const** *retval*);

DESCRIPTION - PBM-SPECIFIC ROUTINES

These library functions are part of Netpbm.

TYPES AND CONSTANTS

typedef ... bit;

#define PBM_WHITE ...

#define PBM_BLACK ...

Each **bit** should contain only the values of **PBM_WHITE** or **PBM_BLACK**.

#define PBM_FORMAT ...

#define RPBM_FORMAT ...

#define PBM_TYPE PBM_FORMAT

#define PBM_FORMAT_TYPE(*f*) ...

These are for distinguishing different file formats and types.

INITIALIZATION

pbm_init() is identical to **pm_init()**.

MEMORY MANAGEMENT

pbm_allocarray() allocates an array of bits. **pbm_allocrow()** allocates a row of the given number of bits. **pbm_freearray()** frees the array allocated with **pbm_allocarray()** containing the given number of rows. **pbm_freerow()** frees a row of bits.

READING PBM IMAGE FILES

pbm_readpbminit() reads the header from a PBM image in a PBM file, filling in the rows, cols and format variables. **pbm_readpbmrow()** reads a row of bits into the *bitrow* array. Format and cols were filled in by **pbm_readpbminit()**. **pbm_readpbmrow_packed()** is like **pbm_readrow()** except instead of returning a **bits** array, it returns an array *packed_bits* of bytes with the pixels of the image row packed into them. The pixels are in order from left to right across the row and from the beginning of the array to the end. Within a byte, the bits are in order from the most significant bit to the least significant bit. If the number of pixels in the row is not a multiple of 8, the last byte returned is padded on the least signficant bit side with undefined bits. White is represented by a **PBM_WHITE** bit; black by **PBM_BLACK**.

pbm_readpbm() reads an entire bitmap file into memory, returning the allocated array and filling in the rows and cols variables. This function combines **pbm_readpbminit()**, **pbm_allocarray()** and **pbm_readpbmrow()**.

pbm_readpbminit() and **pbm_readpbm** abort the program with a message to Standard Error if the PBM image header is not syntactically valid, including if it contains a number too large to be processed using the system's normal data structures (to wit, a number that won't fit in a C 'int').

ppm_readppminit() and **ppm_readppm** abort the program with a message to Standard Error if the PPM image header is not syntactically valid, including if it contains a number too large to be processed using the system's normal data structures (to wit, a number that won't fit in a C 'int').

WRITING PBM IMAGE FILES

pbm_writepbminit() writes the header for a PBM image in a PBM file. *forceplain* is a boolean value specifying that a plain format (text) file to be written, as opposed to a raw format (binary) one. **pbm_writepbmrow()** writes a row to a PBM file. **pbm_writepbmrow_packed()** is the same as **pbm_writepbmrow()** except that you supply the row to write as an array of bytes packed with bits instead of as a **bits** array. The format of *packed_bits* is the same as that returned by **pbm_readpbmrow()**.

pbm_writepbm() writes the header and all data for a PBM image to a PBM file. This function combines **pbm_writepbminit()** and **pbm_writepbmrow()**.

MISCELLANEOUS

pbm_nextimage() positions a PBM input file to the next image in it (so that a subsequent **pbm_readpbminit()** reads its header).

Immediately before a call to **pbm_nextimage()**, the file must be positioned either at its beginning (i.e. nothing has been read from the file yet) or just after an image (i.e. as left by a **pbm_readpbmrow()** of the last row in the image).

In effect, then, all **pbm_nextimage()** does is test whether there is a next image or the file is positioned at end-of-file.

If **pbm_nextimage()** successfully positions to the next image, it returns ***eofP* false (0). If there is no next

image in the file, it returns ***eofP** true . If it can't position or determine the file status due to a file error, it issues an error message and exits the program with an error exit code.

pbm_check() checks for the common file integrity error where the file is the wrong size to contain all the image data. **pbm_check()** assumes the file is positioned after an image header (as if **pbm_readpbminit()** was the last operation on the file). It checks the file size to see if the number of bytes left in the file are the number required to contain the image raster. If the file is too short, **pbm_check()** causes the program to exit with an error message and error completion code. Otherwise, it returns one of the following values (enumerations of the **enum pm_check_code** type) as ***retval**:

PM_CHECK_OK The file's size is exactly what is required to hold the image raster.

PM_CHECK_UNKNOWN_TYPE *format* is not a format whose size **pbm_check()** can anticipate. The only format with which **pbm_check()** can deal is raw PBM format.

PM_CHECK_TOO_LONG The file is longer than it needs to be to contain the image raster. The extra data might be another image.

PM_CHECK_UNCHECKABLE The file is not a kind that has a predictable size, so there is no simple way for **pbm_check()** to know if it is the right size. Only a regular file has predictable size. A pipe is a common example of a file that does not.

check_type must have the value **PM_CHECK_BASIC** (an enumerated value of the **pm_check_type** enumerated type). Otherwise, the effect of **pbm_check()** is unpredictable. This argument exists for future backward compatible expansion of the function of **pbm_check()**.

SEE ALSO

libpgm, **libppm**, **libpnm**, **pbm**

AUTHOR

Copyright (C) 1989, 1991 by Tony Hansen and Jef Poskanzer.

Table Of Contents

DESCRIPTION - PBM-SPECIFIC ROUTINES
TYPES AND CONSTANTS
INITIALIZATION
MEMORY MANAGEMENT
READING PBM IMAGE FILES
WRITING PBM IMAGE FILES
MISCELLANEOUS
SEE ALSO
AUTHOR