AUUG 2002: Imaging Tutorial

Michael Still

AUUG 2002: Imaging Tutorial

by Michael Still

Table of Contents

Chapter 1. Introduction

This document is the manual associated with my tutorial on imaging programming presented at the *Australian Unix User's Group* 2002 Winter Conference in Melbourne Australia. It is intended to serve as the basis for discussions during this day long tutorial, as well as being a reference for the attendees once they return to their every day lives.

About the author

Michael has been working in the image processing field for several years, including a couple of years managing and developing large image databases for an Australian government department. He currently works for TOWER Software, who manufacture a world leading EDMS and Records Management package named TRIM. Michael is also the developer of Panda, an open source PDF generation API, as well as being the maintainer of the comp.text.pdf USENET frequently asked questions document. You can contact Michael at mikal@stillhq.com.

Michael also has a whole bunch of code (most of which relates to imaging) at his website: http://www.stillhq.com

Motivation for this tutorial

This tutorial started life as a series of articles about the TIFF image format, which were published by IBM DeveloperWorks (http://www.ibm.com/developerworks) in April and June 2002. This logically grew into the tutorial you see before you today. This tutorial is based on my several years experience as an imaging developer, and the common mistakes that people seem to make over and over. Perhaps this tutorial will go some way to correcting some common misconceptions.

Assumed knowledge

There are some things which I assume you know, and which are outside the scope of this tutorial.

C

This tutorial discusses code. Almost all of the code discussed is written in C. It is therefore safe to assume that if you don't have a good working knowledge of C, then you're likely to get a lot less out of this tutorial as those who do know some C. On the other hand, don't worry about the more esoteric syntax, I'll explain this as we need it.

It should also be noted that the code samples in this tutorial are not optimal. They have been written to be as readable as possible, and not necessarily the most efficient possible. Please bear this in mind before blindly copying them.

How to compile and link on your chosen operating system

It is outside the scope of this document to teach you how to compile and link source code into an executable form on your chosen architecture and operating system. You will need to understand this before you will be able to use any of the code in this document.

For those of you using gcc on a unix (or unix-like) operating system, then the following points might be all you need to know. If you need more information, then a http://www.google.com search will serve you well.

1. Libraries are added to the link command using the -l command line option. For instance, to compile and link the source file foo, with the tiff library, you would use a command line along the lines of **gcc foo.c -o foo -ltiff -lm**.
2. You need to include -lm almost always. When you compile a c program using gcc without specifying any libraries, you get a -lm for free. As soon as you start specifying any libraries (for instance in this case -ltiff), then you also need to explicitly specify the math library as well.
3. You will almost certainly also need to add the library and include paths for the installed version of the relevant library to the compile command line as well. Directories are added to the library search path using the -L command line option. Include directories are added with the -I option.
4. The make files included with the samples in this tutorial a probably a bad place to look for introductory compile examples, as they use automake and autoconf to try to detect where the various required libraries are installed...

Credits

There are many people who need to be thanked when one attempts a project of this size. I would specifically like to thank my wife and son, who put up with me being geeky so very often.

Figure 1-1. Catherine and Andrew

I should also thank the following people:

- Doug Jackson (doug_jackson@citadel.com.au), for proof reading and getting me interested in this whole topic to start with
- Tony Green (greeno@bandcamp.tv), for occasional DocBook wrangling
- Michael Smith (smith@xml-doc.org), for DocBook hints
- Greg Lehey, for giving me access to his docbook environment when I was having jade pain
- Lenny Muellner, of O'Reilly, for helping me with my gmat problems

How this document was produced

This tutorial was written in DocBook SGML using xemacs. This was then converted into PDF using the jade SGML tools. Diagrams were developed in a combination of the gimp, Adobe Illustrator, and custom developed code. Diagrams were converted to EPS as required by jade using ImageMagick.

A series of DocBook generation scripts was also used to automate some of the generation of this document. These scripts can be found in my online CVS repository at: http://www.stillhq.com/cgi-bin/cvsweb/docbooktools/

The source for this tutorial is available at: http://www.stillhq.com/cgi-bin/cvsweb/tutorial-imaging/

License

License for text (OPL)

Commercial redistribution of Open Publication-licensed material is permitted.

Any publication in standard (paper) book form shall require the citation of the original publisher and author. The publisher and author's names shall appear on all outer surfaces of the book. On all outer surfaces of the book the original publisher's name shall be as large as the title of the work and cited as possessive with respect to the title.

II. COPYRIGHT

The copyright to each Open Publication is owned by its author(s) or designee.

III. SCOPE OF LICENSE

The following license terms apply to all Open Publication works, unless otherwise explicitly stated in the document.

Mere aggregation of Open Publication works or a portion of an Open Publication work with other works or programs on the same media shall not cause this license to apply to those other works. The aggregate work shall contain a notice specifying the inclusion of the Open Publication material and appropriate copyright notice.

SEVERABILITY. If any part of this license is found to be unenforceable in any jurisdiction, the remaining portions of the license remain in force.

NO WARRANTY. Open Publication works are licensed and provided "as is" without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement.

IV. REQUIREMENTS ON MODIFIED WORKS

All modified versions of documents covered by this license, including translations, anthologies, compilations and partial documents, must meet the following requirements:

- 1) The modified version must be labeled as such.
- 2) The person making the modifications must be identified and the modifications dated.
- 3) Acknowledgement of the original author and publisher if applicable must be retained according to normal academic citation practices.
- 4) The location of the original unmodified document must be identified.
- 5) The original author's (or authors') name(s) may not be used to assert or imply endorsement of the resulting document without the original author's (or authors') permission.

V. GOOD-PRACTICE RECOMMENDATIONS

In addition to the requirements of this license, it is requested from and strongly recommended of redistributors that:

- 1) If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.

- 2) All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.

  Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy and CD-ROM expression of an Open Publication-licensed work to its author(s).

VI. LICENSE OPTIONS

The author(s) and/or publisher of an Open Publication-licensed document may elect certain options by appending language to the reference to or copy of the license. These options are considered part of the license instance and must be included with the license (or its incorporation by reference) in derived works.

- A. To prohibit distribution of substantively modified versions without the explicit permission of the author(s). "Substantive modification" is defined as a change to the semantic content of the document, and excludes mere changes in format or typographical corrections.

  To accomplish this, add the phrase 'Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.' to the license reference or copy.

- B. To prohibit any publication of this work or derivative works in whole or in part in standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from the copyright holder.

  To accomplish this, add the phrase 'Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.' to the license reference or copy.

  OPEN PUBLICATION POLICY APPENDIX:

  (This is not considered part of the license.)

  Open Publication works are available in source format via the Open Publication home page at <URL:tbd>.

  Open Publication authors who want to include their own license on Open Publication works may do so, as long as their terms are not more restrictive than the Open Publication license.

  If you have questions about the Open Publication License, please contact TBD, and/or the Open Publication Authors' List at <TBD>, via email.

License for source code (GPL)

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's

software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications

or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the

same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later

version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

License for the libtiff man pages

Copyright (c) 1988-1997 Sam Leffler Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Chapter 2. Imaging concepts

There are some core terms which are used throughout this tutorial. This section defines these terms -- it's probably therefore worth bookmarking this section for reference throughout the rest of the day. This section presents these things in alphabetical order, for ease of reference.

These terms are presented in alphabetical order. Some of them might seem a bit odd, but they're here because they are interesting...

Anti-aliasing

Imagine that you are drawing a triangle across an image (or on the screen for that matter). The triangle is sometimes going to cross pixels in a way which makes them not totally turned on. Have a look at the figure below to see what I mean...

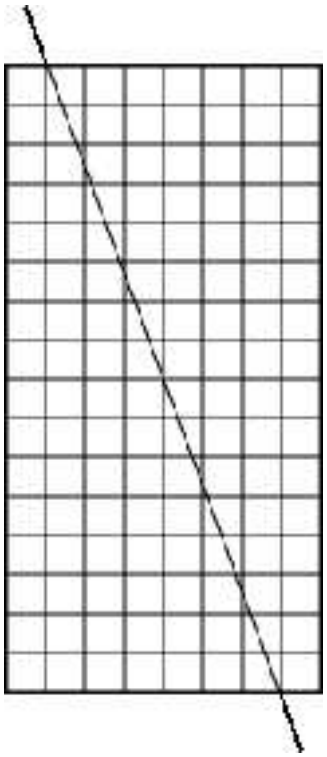

Figure 2-1. Drawing a triangle

If we only have a black and white image, then we'll end up with an image like the one in the figure below. I am sure you'll agree that this isn't a very good representation of the side of the triangle.

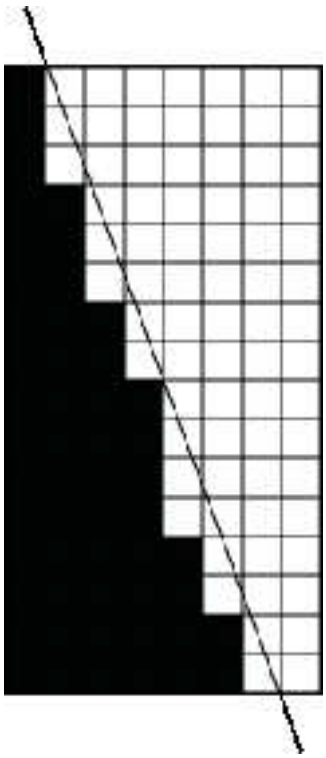

Figure 2-2. A triangle with only black pixels

Anti-aliasing is when we try to correct for this problem by inserting some gray pixels. In the figure below, we have given some of the pixels a gray value which is based on how much of the pixel is “filled” with the triangle.

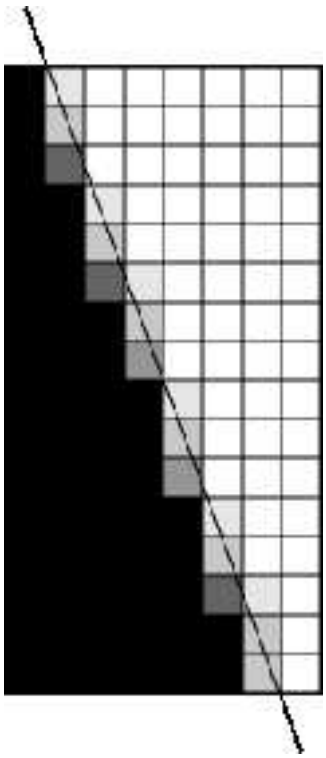

Figure 2-3. A triangle with gray scale pixels

The triangle might be a little clearer without the grid lines.

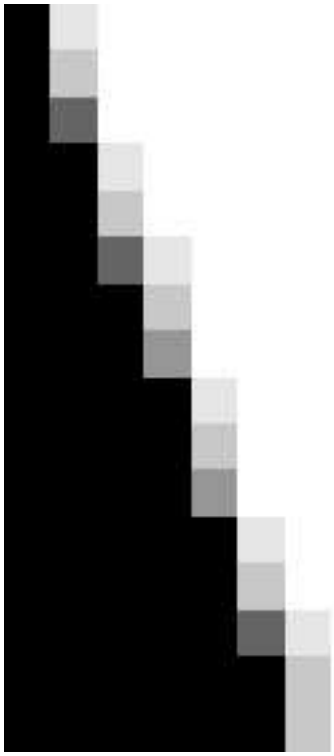

Figure 2-4. Anti-aliased triangle without grid lines

For comparison, there is the triangle we started with...

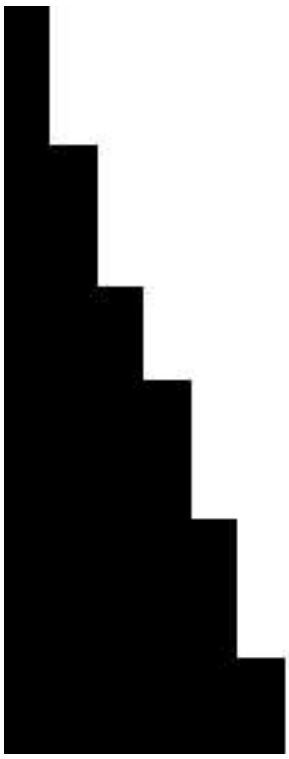

Figure 2-5. The original triangle

So, in summary, anti-aliasing is the process of turning on some extra gray scale pixels to improve the look of shapes we are drawing...

Encrypting images

DES in Electronic Code Book (ECB) mode is a particularly poor choice of cryptography for image files. This is because ECB mode implements a look up table between the unencrypted value and the encrypted value. This results in a known input value turning into the same output value over and over. This can have some interesting blurring effects, but wont obscure the image contents. An example will help this make more sense -- the first figure is the logo for the company I am currently working for. I took this image, and ran it through some DES ECB code and produced the second figure.

Figure 2-6. The TOWER corporate logo

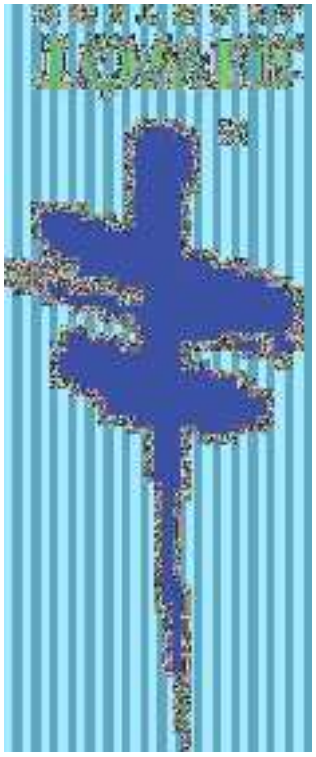

Figure 2-7. After ECB encryption

You can see that whilst the image has certainly changed, the contents of the image has not really been obscured.

We should note that ECB mode is not commonly used anyway. If you use something like PGP or blowfish, then you should be much happier...

Gray scale conversion

How do you convert color images to gray scale? Well, my instant answer when I first had to do this was to just average the red, green and blue values. That answer is wrong. The reality is that the human eye is much better at seeing some colors than others. To get an accurate gray scale representation, you need to apply different coefficients to the color samples. Appropriate coefficients are 0.299 for red, 0.587 for green and 0.114 for blue.

Below I have included several pictures that illustrate this concept. The first figure is a color image [1] , the second is the color image converted to gray scale without the coefficients applied, and the third image is a correct gray scale rendition.

Figure 2-8. The original image

Figure 2-9. An average of the color values for each pixel

Figure 2-10. A correct conversion to gray scale

You can see that the sensible algorithm is better, have a look at the shadows on the hill and the trees in the background. The contrast and darkness of the second picture is might better than the averaging algorithm...

The source code to generate these example images may be found in the TIFF chapter.

Pixel samples

Each pixel needs to have a value associated with this. Most practitioners call this a sample. For a black and white image we would only have one sample per pixel, whilst for a RGB color image, we would have three samples per pixel -- the red, green and blue components.

Pixel

A pixel is the lowest unit of raster image description -- in other words, a raster image is defined in terms of pixels. Example pixels can be seen in the discussion of rasters, later in this chapter.

Rasters

The most common image format (and the focus of the majority of this tutorial) is raster image formats. Raster image formats are those which store the picture as a bitmap [2] describing the state of pixels, as opposed to recording the length and locations of primatives such as lines and curves. Common examples include the output of photoshop, the gimp, scanners and digital cameras. Effectively, your monitor and printer are raster output devices.

Figure 2-11. A sample raster image

Raster image formats include TIFF, GIF, PNG, and most of the other formats in this tutorial. If in doubt, assume that it's a raster format unless I tell you otherwise.

If we zoom in enough on a portion of this image, we can see that it is made up of descrete elements -- the pixels.

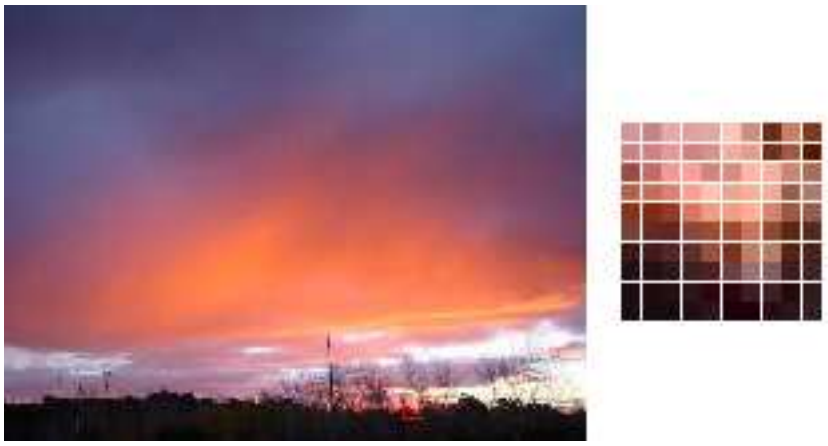

Figure 2-12. Zooming in on a portion of the raster image

The code to produce this zoomed image is presented in the TIFF chapter of this tutorial.

Theory of color and gray scale storage

There are several ways that the value of a pixel can be stored within a raster image. This is a pretty fundamental concept, so it is best to get it out of the way early on before we get caught up in the actual format of images.

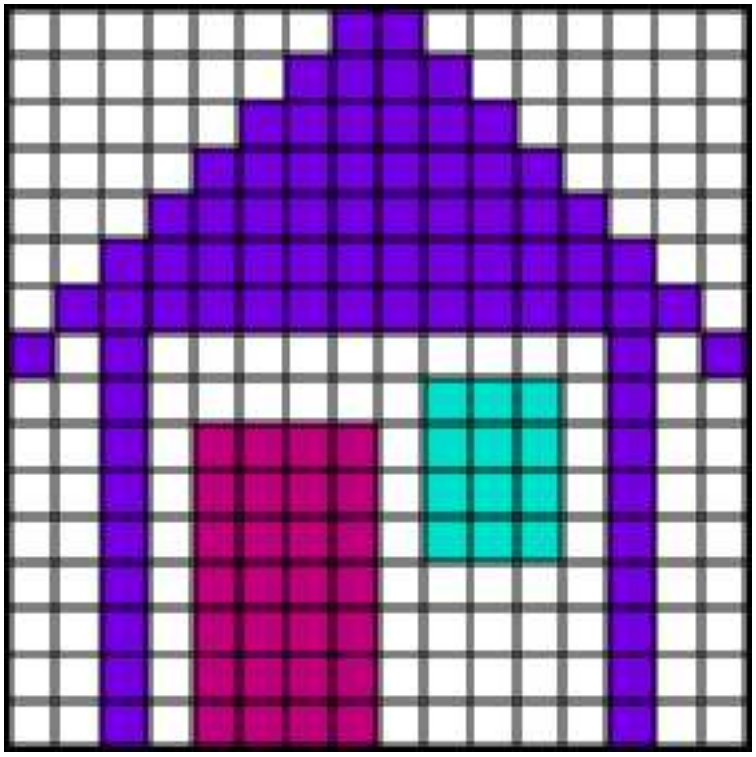

Figure 2-13. The worst picture of a house you have ever seen

I will use the simple image above to demonstrate the ways that the pixel values can be stored...

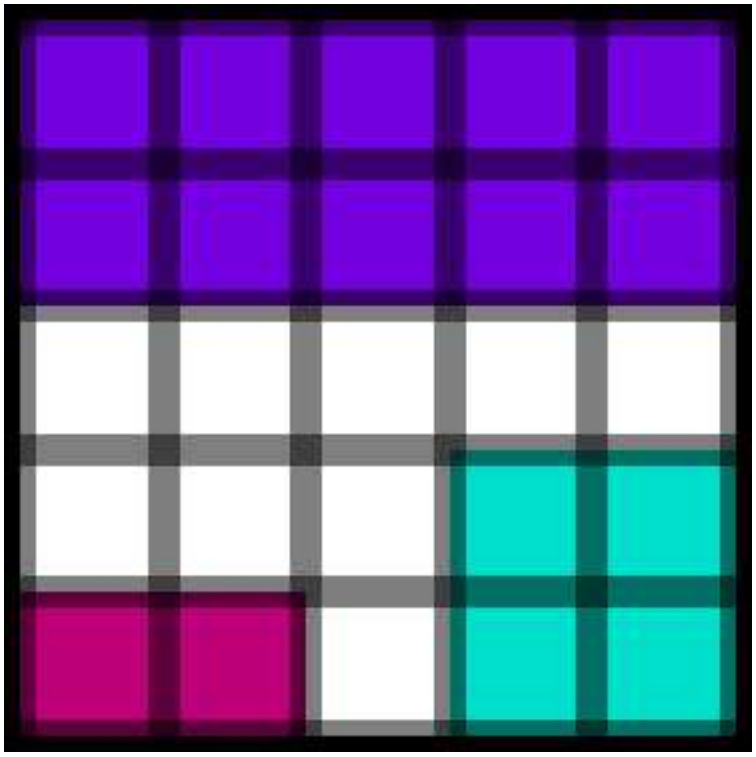

Figure 2-14. A zoom in on the house

We'll in fact only use a zoomed in portion of the image so that what is happening is clearer.

Direct storage of black and white

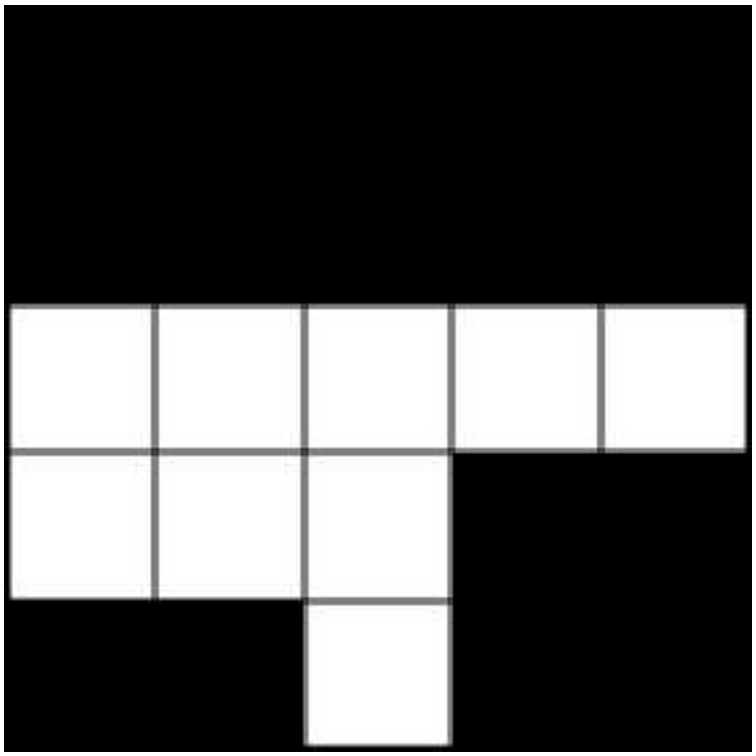

Figure 2-15. A black and white zoom in

Black and white images only really have one pixel value storage option. This is to store the value of the pixel directly at the pixel location in the image data itself. This is a good option for black and white data, because it only takes on bit per pixel anyway.

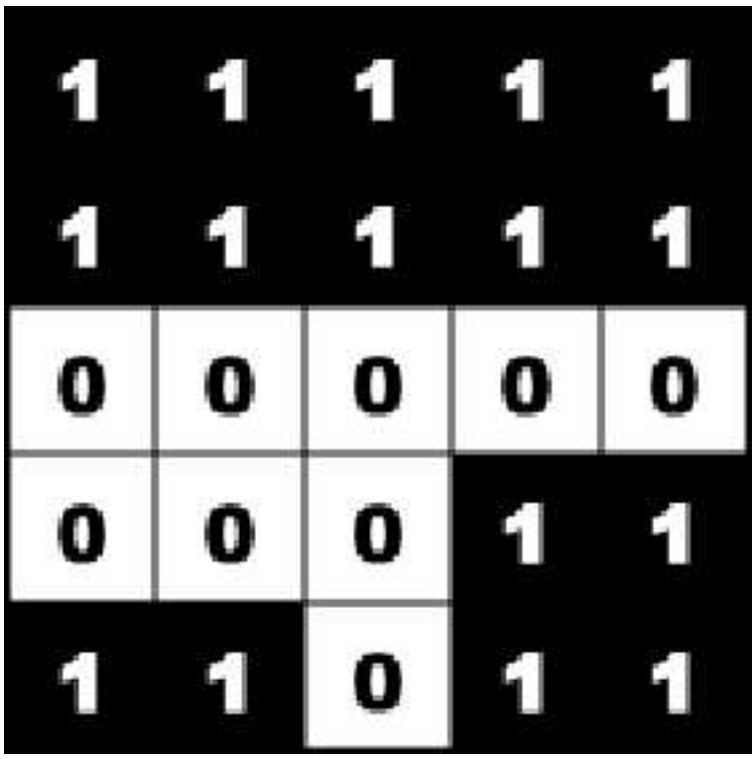

Figure 2-16. Black and white data

Whilst this is a very simple example, make sure you understand how the diagram works, because it gets more complex from here. Each square represents the storage space in the image data, and in this example the values representing the pixel values is stored inside the image data.

Direct gray scale storage

Figure 2-17. A gray scale zoom in

In this example, we are storing the gray scale values of the pixels within the image data.

Direct RGB storage

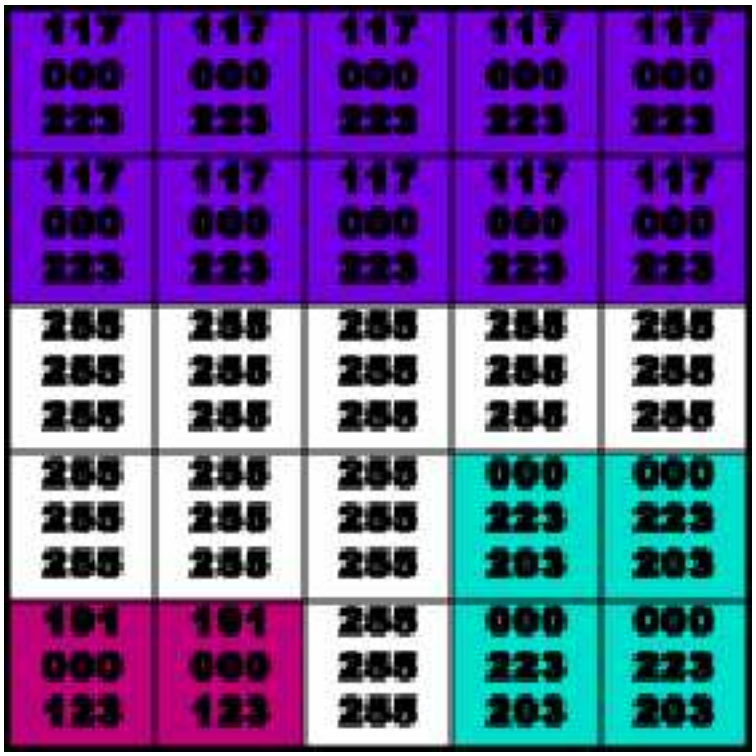

Figure 2-18. A RGB zoom in

Here we are storing the red green and blue values for each pixel within the image data.

Paletted RGB storage

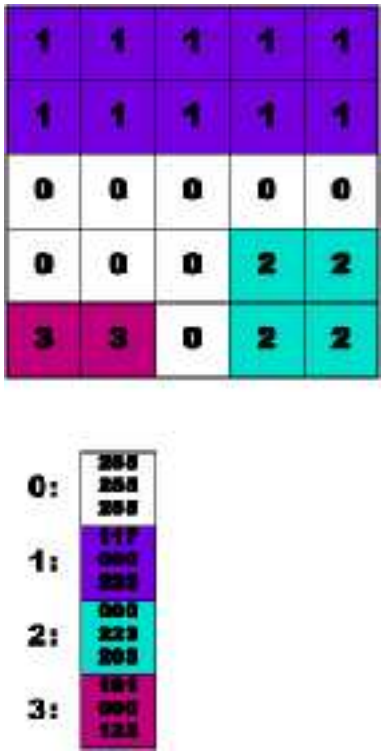

Figure 2-19. A paletted RGB zoom in

The other option is to instead store within the image data itself a number which uniquely identifies the color at that pixel. We can then have a table elsewhere in the image file which defines the color that is that unique value. This table is a palette, and this is a very common way of storing RGB data (it is in fact the only option with some formats such as GIF).

You can also use palettes for gray scale images, I just haven't provided an example of that here.

The advantage of paletting

The big advantage of paletting an image is that the final file is going to be much smaller. For example, an A4 page is 1754 by 2479 pixels. That's 4,348,166 pixels. Now, let's assume for this example that the image is 24 bit color (that is 8 bits per color per pixel), and that there are seven colors in the image. That means that unpaletted, we have 13,044,498 bytes of image data (uncompressed). If we palette the image data, then we only need three bits of data in the image data -- 1,630,562 bytes (uncompressed). We'll also need to store the palette itself, which will need another 21 bytes (uncompressed). We don't need to store the color indices, as they are just an offset into an array.

This means that the total saving on an uncompressed image is 11,413,936 bytes. The disadvantage of course is that using the image is almost certainly going to be slightly slower...

Vector

Some images aren't raster images. These images are normally composed of primitive drawing commands, such as draw a line, draw a circle, et cetera. The advantage of vector formats is that it is an exact representation of the image, they also scale much better. Examples of vector formats include Scalar Vector Graphics (SVG), Adobe Illustrator files, Windows Meta Files, and to a certain extent PDF documents. A plotter is a vector output device.

Notes

1. Which may or may not give you joy, depending on if you are viewing this document in color or not
2. You can think of a bitmap as being simply an array of pixels. We'll talk more about this later

Chapter 3. TIFF

"With great power comes great responsibility" -- Spiderman

In this chapter we will discuss the Tagged Image File Format, which is one of the most common raster image formats for professional imaging -- especially scanned images and faxes. Unfortunately, it is not supported by web browsers, so isn't used as much as it might otherwise be.

Introduction

TIFF (Tagged Image File Format) is a raster image format which was originally produced by Aldus and Microsoft. Aldus was later acquired by Adobe, who manage the TIFF specification to this day. At the time of writing of this chapter, the current version of the TIFF specification is TIFF version 6.0. The TIFF specification can be downloaded from http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf. The TIFF technical notes at http://partners.adobe.com/asn/developer/technotes/main.html

Libtiff is a standard implementation of the TIFF specification, which is free and works on many operating systems. All of the examples in this chapter refer to libtiff.

Installation

The source for libtiff can be downloaded from http://www.libtiff.org/ -- the current version of the library being version 3.5.7 at the time of writing. This version can be downloaded from ftp://ftp.remotesensing.org/pub/libtiff, look for the files named either tiff-v3.5.7.tar.gz or tiff-v3.5.7.zip.

Unix

Once you have downloaded the tarball for the library as described above, then you need to follow the steps below to install libtiff [1].

1. tar xvzf tiff-v3.5.7.tar.gz
2. cd tiff-v3.5.7
3. ./configure
4. make
5. make install

Not too hard at all...

win32

The instructions for how to compile the library are as follows:

- Uncompress the libtiff distribution
- Change directory into the libtiff directory inside the distribution
- Copy ..\contrib\winnt\fax3sm.c to fax3sm_winnt.c
- Copy ..\contrib\winnt\libtiff.def to libtiff.def
- Remove the line for TIFFModeCCITTFax3 from the libtiff.def file
- Change the line for TIFFFlushdata1 in the libtiff.def file to TIFFFlushData1

- nmake /f makefile.vc all
- Done!

To compile the tools once you have the library, just change into the tools directory and do a nmake /f makefile.vc

For the Visual Studio challenged, there is a precompiled version of libtiff for win32 available from http://www.stillhq.com. Have a look at the Panda documentation page...

The TIFF on disc format

It is useful for use to know something about how TIFF images are laid out on disc. For a fuller discussion of this, have a look at the TIFF specification version 6, which is referenced in the further reading section of this chapter [2] .

File header

Table 3-1. TIFF on disc: the file header

Bytes	Description
0 - 1	Byte order for this file: ll (ASCII) for little endian, or MM (ASCII) for big endian
2 - 3	Magic number = 42, in the byte order specified above
4 - 7	Offset in bytes to the first IFD [a]
Notes: a. IFD: Image File Directory	

Image File Directory

Table 3-2. TIFF on disc: the image file directory

Bytes	Description
0 - 1	Number of entries in the IFD
???	The entries in the IFD (all are 12 bytes long)
4 bytes at end	Offset in bytes to the next IFD (four zero bytes if this is the last IFD)

Image File Directory Entries

Table 3-3. TIFF on disc: an image file directory

Bytes	Description
0 - 1	Tag that identifies the entry. Have a look at tiff.h
2 - 3	Field type

Bytes	Description
4 - 7	Count of the number of type size fields used. For instance, if the type is an ASCII string, then this field will store the length of the string, including the NULL terminating byte that C strings have. The count does not include padding (if any).
8 - 11	The number of bytes inside the file that the value is stored at. Because the value must be stored on a word boundary, it will always be an even number. This file offset may point anywhere in the file, even after the image data.

These entries have a type associated with them, possible types are:

- 1: BYTE 8 bit unsigned integer
- 2: ASCII 8 bit byte that contains a 7 bit ASCII code
- 3: SHORT 16 bit (2 byte) unsigned integer
- 4: LONG 32 bit (4 byte) unsigned integer
- 5: RATIONAL Two LONGs: the first represents the numerator of a fraction; the second, the denominator

TIFF version 6 added the following fields:

- 6: SBYTE An 8 bit signed integer
- 7: UNDEFINED An 8 bit byte that may contain anything
- 8: SSHORT A 16 bit (2 byte) signed integer
- 9: SLONG A 32 bit (4 byte) signed integer
- 10: SRATIONAL Two SLONGs: the first represents the numerator of a fraction; the second, the denominator
- 11: FLOAT Singled precision (4 byte) IEEE format
- 12: DOUBLE Double precision (8 byte) IEEE format

Sign is implemented using two's complement notation. New field types may be added later, although it seems unlikely at this stage, given the TIFF specification hasn't changed in quite a long time. Image readers should ignore types they don't understand [3] .

Possible field entries

Discussing the many different possible field entries is out of the scope of this document. Refer to tiff.h and the TIFF specification for more information.

So where's the image data?

Interestingly, the image data itself is just stored as another tag value... The tag value StripOffsets stores where in the file the image data strips start. The tag value StripByteCounts stores the size of each strip.

Coding for TIFF can be hard

Most file format specifications define some basic rules for the representation of the file. For instance, PNG (a competitor to TIFF) documents are always big endian. TIFF doesn't mandate things like this though, here is a list of some of the seemingly basic things that it doesn't define:

1. The byte order -- big endian, or little endian
2. The fill order of the bit within the image bytes -- most significant bit first, or least significant
3. The meaning of a given pixel value for black and white -- is 0 black, or white?
4. ...and so on

This means that creating a TIFF can be very easy, because it is rare to have to do any conversion of the data that you already have. It does mean, on the other hand, that being able to read in random TIFFs created by other applications can be very hard -- you have to code for all these possible combinations in order to be reasonably certain of having a reliable product.

So how do you write an application which can read in all these different possible permutations of the TIFF format? The most important thing to remember is to *never make assumptions about the format of the image data you are reading in*.

Writing Black and White TIFF files

The first thing I want to do is show you how to write a TIFF file out. We'll then get onto how to read a TIFF file back into your program.

Infrastructure for writing

It is traditional for bitmaps to be represented inside your code with an array of chars. This is because on most operating systems, a char maps well to one byte. In the block of code below, we will setup libtiff, and create a simple buffer which contains an image which we can then write out to disc.

```
#include <stdio.h>
#include <tiffio.h>

int
main (int argc, char *argv[])
{
  char buffer[32 * 9];
}
```

Code: write-infrastructure.c

The code above is pretty simple. All you need to use libtiff is to include the tiffio.h header file. The char buffer that we have defined here is going to be our black and white image, so we should define one of those next...

Writing the image

To make up for how boring that example was, I am now pleased to present you with possibly the worst picture of the Sydney Harbor Bridge ever drawn. In the example below, the image is already in the image buffer, and all we have to do is save it to the file on disc. The example first opens a TIFF image in write mode, and then places the image into that file.

Please note, that for clarity I have omitted the actual hex for the image, this is available in the download version of this code for those who are interested.

```
#include <stdio.h>
#include <tiffio.h>

int main(int argc, char *argv[]){
  // Define an image -- this is 32 pixels by 9 pixels
  char buffer[25 * 144] = { ...boring hex omitted... };

  TIFF *image;

  // Open the TIFF file
  if((image = TIFFOpen("output.tif", "w")) == NULL){
    printf("Could not open output.tif for writing\n");
    exit(42);
  }

  // We need to set some values for basic tags before we can add any data
  TIFFSetField(image, TIFFTAG_IMAGEWIDTH, 25 * 8);
  TIFFSetField(image, TIFFTAG_IMAGELENGTH, 144);
  TIFFSetField(image, TIFFTAG_BITSPERSAMPLE, 1);
  TIFFSetField(image, TIFFTAG_SAMPLESPERPIXEL, 1);
  TIFFSetField(image, TIFFTAG_ROWSPERSTRIP, 144);

  TIFFSetField(image, TIFFTAG_COMPRESSION, COMPRESSION_CCITTFAX4);
  TIFFSetField(image, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_MINISWHITE);
  TIFFSetField(image, TIFFTAG_FILLORDER, FILLORDER_MSB2LSB);
  TIFFSetField(image, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);

  TIFFSetField(image, TIFFTAG_XRESOLUTION, 150.0);
  TIFFSetField(image, TIFFTAG_YRESOLUTION, 150.0);
  TIFFSetField(image, TIFFTAG_RESOLUTIONUNIT, RESUNIT_INCH);

  // Write the information to the file
  TIFFWriteEncodedStrip(image, 0, buffer, 25 * 144);

  // Close the file
  TIFFClose(image);
}
```

Code: write-nohex.c

There are some interesting things to note in this example. The most interesting of these is that the output image will not display using the xview command on my Linux machine. In fact, I couldn't find an example of a group 4 fax compressed black and white image which would display using that program. See the sidebar for more detail.

Problems with xview

Xview is part of the xloadimage package written by Jim Frost, which comes with X windows.

It's a good example of how hard it can be to handle TIFF images well. If you have trouble viewing the output of the sample code, then try using some other program, like the GIMP.

The sample code shows the basics of using the libtiff API. The following interesting points should be noted...

1. The buffers presented to and returned from libtiff each contain 8 pixels in a single byte. This means that you have to be able to extract the pixels you are

interested in. The use of masks, and the right and left shift operators come in handy here.

2. The TIFFOpen function is very similar to the fopen function we are all familiar with.
3. We need to set the value for quite a few fields before we can start writing the image out. These fields give libtiff information about the size and shape of the image, as well as the way that data will be compressed within the image. These fields need to be set before you can start handing image data to libtiff. There are many more fields for which a value could be set, I have used close to the bar minimum in this example.
4. TIFFWriteEncodedStrip is the function call which actually inserts the image into the file. This call inserts uncompressed image data into the file. This means that libtiff will compress the image data for you before writing it to the file. If you have already compressed data, then have a look at the TIFFWriteRawStrip instead.
5. Finally, we close the file with TIFFClose.

More information about the libtiff function calls

If you need more information about any of the libtiff function calls mentioned in this chapter, then checkout the extensive man pages which come with the library. Remember that case is important with man pages, so you need to get the case in the function names right -- it's TIFFOpen, not tiffopen.

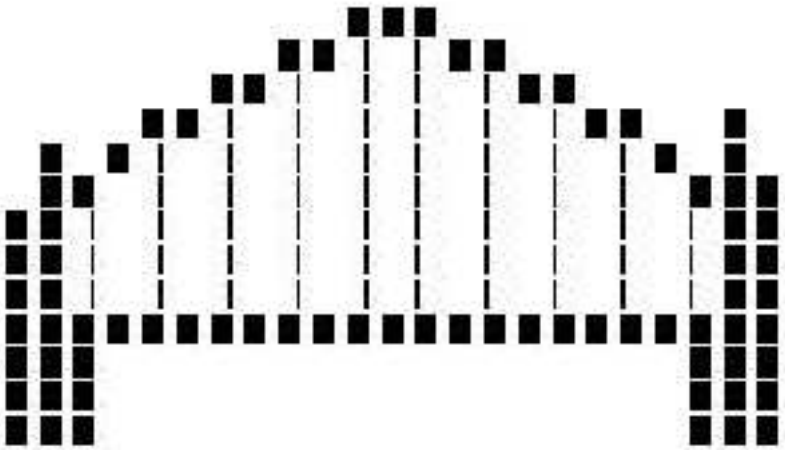

Figure 3-1. The Sydney Harbor Bridge, by Michael Still

Reading Black and White TIFF files

Reading TIFF files reliably is much harder than writing them. The issue that complicates reading black and white TIFF images the most is the several different storage schemes which are possible within the TIFF file itself. libtiff doesn't hold your hand much with these schemes, so you have to be able to handle them yourself. The three schemes TIFF supports are single stripped images, stripped images, and tiled images.

1. A single strip image is as the name suggests -- a special case of a stripped image. In this case, all of the bitmap is stored in one large block. I have experienced reliability issues with images which are single strip on Windows machines. The general recommendation is that no one strip should take more than 8 kilobytes uncompressed which with black and white images limits us to 65,536 pixels in a single strip.
2. A multiple strip image is where horizontal blocks of the image are stored together. More than one strip is joined vertically to make the entire bitmap. Figure 2 shows this concept.
3. A tiled image is like your bathroom wall, it is composed of tiles. This representation is show in Figure 3, and is useful for extremely large images -- this is especially true when you might only want to manipulate a small portion of the image at any one time.

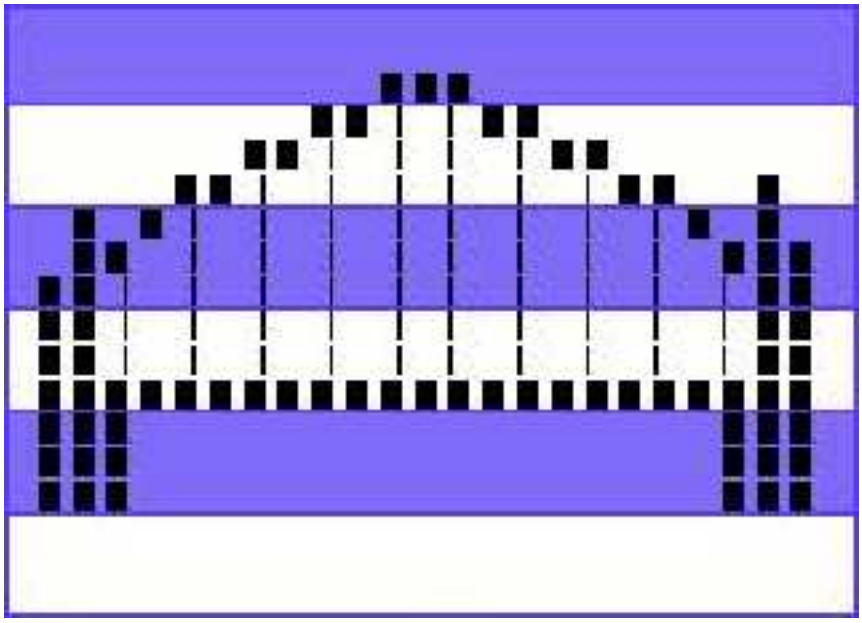

Figure 3-2. The Sydney Harbor Bridge, in strips

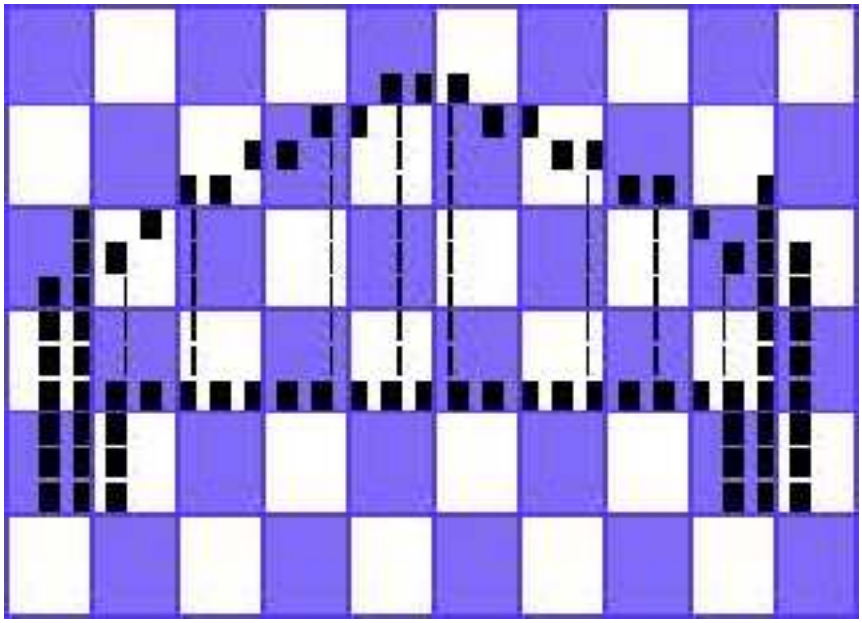

Figure 3-3. The Sydney Harbor Bridge, in tiles

Tiled images are comparatively uncommon, so I will focus on stripped images in this chapter. Remember as we go along, that the single stripped case is merely a subset of a multiple strip images.

Infrastructure for reading

The most important thing to remember when reading in TIFF images is to be flexible. The example below has the same basic concepts as the writing example above, with the major difference being that it needs to deal with many possible input images. Apart from stripping and tiling, the most important thing to remember to be flexible about is photo metric interpretation. Luckily, with black and white images there are only two photo metric interpretations to worry about (with color and to a certain extent gray scale images there are many more).

What is photo metric interpretation? Well, the representation of the image in the buffer is really a very arbitrary thing. I might code my bitmaps so that 0 means black (PHOTOMETRIC_MINISBLACK), whilst you might find black being 1 (PHOTOMETRIC_MINISWHITE) more convenient. TIFF allows both (in the TIFFTAG_PHOTOMETRIC tag), so our code has to be able to handle both cases. In the example below, I have assumed that the internal buffers need to be in PHOTOMETRIC_MINISWHITE so we will convert images which are in PHOTOMETRIC_MINISBLACK.

The other big thing to bear in mind is fill order (whether the first bit in the byte is the highest value, or the lowest). The example below also handles both of these correctly. I have assumed that we want the buffer to have the most significant bit first. TIFF images can be either big endian or little endian, but libtiff handles this for us. Thankfully, libtiff also supports the various compression algorithms without you having to worry about those. These are by far the scariest area of TIFF, so it is still worth your time to use libtiff.

```
#include <stdio.h>
#include <tiffio.h>

int main(int argc, char *argv[]){
```

```
TIFF *image;
uint16 photo, bps, spp, fillorder;
uint32 width;
tsize_t stripSize;
unsigned long imageOffset, result;
int stripMax, stripCount;
char *buffer, tempbyte;
unsigned long bufferSize, count;

// Open the TIFF image
if((image = TIFFOpen(argv[1], "r")) == NULL){
  fprintf(stderr, "Could not open incoming image\n");
  exit(42);
}

// Check that it is of a type that we support
if((TIFFGetField(image, TIFFTAG_BITSPERSAMPLE, &bps) == 0) || (bps != 1)){
  fprintf(stderr, "Either undefined or unsupported number of bits per sample\n");
  exit(42);
}

if((TIFFGetField(image, TIFFTAG_SAMPLESPERPIXEL, &spp) == 0) || (spp != 1)){
  fprintf(stderr, "Either undefined or unsupported number of samples per pixel\n");
  exit(42);
}

// Read in the possibly multile strips
stripSize = TIFFStripSize (image);
stripMax = TIFFNumberOfStrips (image);
imageOffset = 0;

bufferSize = TIFFNumberOfStrips (image) * stripSize;
if((buffer = (char *) malloc(bufferSize)) == NULL){
  fprintf(stderr, "Could not allocate enough memory for the uncompressed image\n");
  exit(42);
}

for (stripCount = 0; stripCount < stripMax; stripCount++){
  if((result = TIFFReadEncodedStrip (image, stripCount,
                    buffer + imageOffset,
                    stripSize)) == -1){
    fprintf(stderr, "Read error on input strip number %d\n", stripCount);
    exit(42);
  }

  imageOffset += result;
}

// Deal with photometric interpretations
if(TIFFGetField(image, TIFFTAG_PHOTOMETRIC, &photo) == 0){
  fprintf(stderr, "Image has an undefined photometric interpretation\n");
  exit(42);
}

if(photo != PHOTOMETRIC_MINISWHITE){
  // Flip bits
  printf("Fixing the photometric interpretation\n");

  for(count = 0; count < bufferSize; count++)
    buffer[count] = ~buffer[count];
}

// Deal with fillorder
if(TIFFGetField(image, TIFFTAG_FILLORDER, &fillorder) == 0){
  fprintf(stderr, "Image has an undefined fillorder\n");
  exit(42);
```

```
  }

  if(fillorder != FILLORDER_MSB2LSB){
    // We need to swap bits -- ABCDEFGH becomes HGFEDCBA
    printf("Fixing the fillorder\n");

    for(count = 0; count < bufferSize; count++){
      tempbyte = 0;
      if(buffer[count] & 128) tempbyte += 1;
      if(buffer[count] & 64) tempbyte += 2;
      if(buffer[count] & 32) tempbyte += 4;
      if(buffer[count] & 16) tempbyte += 8;
      if(buffer[count] & 8) tempbyte += 16;
      if(buffer[count] & 4) tempbyte += 32;
      if(buffer[count] & 2) tempbyte += 64;
      if(buffer[count] & 1) tempbyte += 128;
      buffer[count] = tempbyte;
    }
  }

  // Do whatever it is we do with the buffer -- we dump it in hex
  if(TIFFGetField(image, TIFFTAG_IMAGEWIDTH, &width) == 0){
    fprintf(stderr, "Image does not define its width\n");
    exit(42);
  }

  for(count = 0; count < bufferSize; count++){
    printf("%02x", (unsigned char) buffer[count]);
    if((count + 1) % (width / 8) == 0) printf("\n");
    else printf(" ");
  }

  TIFFClose(image);
}
```

Code: read.c

This code works by first opening the image and checking that it is one that we can handle. It then reads in all the strip for the image, and appends them together in one large memory block. If required, it also flips bits until the photo metric interpretation the one we handle, and deals with having to swap bits if the fill order is wrong. Finally, our sample outputs the image as a series of lines composed of hex values. Remember that each of the values represents 8 pixels in the actual image.

Compression algorithms in libtiff

There are several compression algorithms available within libtiff. How do you select which one is right for your imaging needs?

Please note that all the tags in this table should be preceded by COMPRESSION, for example COMPRESSION_CCITTFAXG4.... The compression option is stored in the tag TIFFTAG_COMPRESSION.

Table 3-4. Libtiff compression algorithms

Compression algorithm	Well suited for	COMPRESSION

Compression algorithm	Well suited for	COMPRESSION
CCITT Group 4 Fax and Group 3 Fax	If your coding for black and white images, then you're probably using the CCITT fax compression methods. These compression algorithms don't support color.	_CCITTFAX3, _CCITTFAX4
JPEG	JPEG compression is great for large images such as photos. However, the compression is normally lossy (in that image data is thrown away as part of the compression process). This makes JPEG very poor for compressing text which needs to remain readable. The other thing to bear in mind is that the loss is accumulative -- see the loss section below for more information about this.	_JPEG
LZW	*This is the compression algorithm used in GIF images. Because of the licensing requirements from Unisys, support for this compression codec has been removed from libtiff. There are patches available if you would like to add it back, but the majority of programs your code will integrate with no linger support LZW.*	_LZW
Deflate	This is the gzip compression algorithm, which is also used for PNG. It is the compression algorithm I would recommend for color images.	_DEFLATE

Accumulating loss?

Why does the loss in lossy compression algorithms such as JPEG accumulate? Imagine that you compress an image using JPEG. You then need to add say a bar code to the image, so you uncompress the image, add the bar code, and recompress it. When the recompression occurs, then a new set of loss is introduced. You can imagine that if you do this enough, then you'll end up with an image which is a big blob.

Whether this is a problem depends on the type of your data. To test how much of a problem this is, I wrote a simple libtiff program which repeatedly uncompresses and recompresses an image. What I found was that with pictures, the data is much more resilient to repeated compression.

Figure 3-4. The picture before we compressed it

```
<sidebar>
<heading refname="" type="sidebar" toc="no">Accu
<p>Why does the loss in lossy compression algori
 add say a barcode to the image, so you uncompre
et of loss is introduced. You can imagine that i

<p>Whether this is a problem depends on the type
repeatedly uncompresses and recompresses an imag
on.</p>

<figure>
<heading refname="picture-start" type="figure" t
<img src="xxx.jpg" width="500" height="200" alt=
</figure>

<figure>
<heading refname="text-start" type="figure" toc=
<img src="xxx.jpg" width="500" height="200" alt=
</figure>
```

Figure 3-5. The sample text before we compressed it

The code I used had a 'quality' rating of 25% on the JPEG compression, which is a way of tweaking the loss of the compression algorithm. The lower the quality, the higher the compression ratio. The default is 75%.

Figure 3-6. The picture after it has been recompressed 200 times

```
<sidebar>
<heading refname="" type="sidebar" toc="no">Accu
<p>Why does the loss in lossy compression algori
 add say a barcode to the image, so you uncompre
et of loss is introduced. You can imagine that i

<p>Whether this is a problem depends on the type
repeatedly uncompresses and recompresses an imag
on.</p>

<figure>
<heading refname="picture-start" type="figure" t
<img src="xxx.jpg" width="500" height="200" alt=
</figure>

<figure>
<heading refname="text-start" type="figure" toc=
<img src="xxx.jpg" width="500" height="200" alt=
</figure>
```

Figure 3-7. The text after it has been recompressed 200 times

The code for the repeated compression example is at the end of this chapter.

Writing a color image

It time to show you how to write a color image to disc. Remember that this is a simple example, which can be elaborated on greatly.

```
#include <tiffio.h>
#include <stdio.h>

int main(int argc, char *argv[]){
```

```
  TIFF *output;
  uint32 width, height;
  char *raster;

  // Open the output image
  if((output = TIFFOpen("output.tif", "w")) == NULL){
    fprintf(stderr, "Could not open outgoing image\n");
    exit(42);
  }

  // We need to know the width and the height before we can malloc
  width = 42;
  height = 42;

  if((raster = (char *) malloc(sizeof(char) * width * height * 3)) == NULL){
    fprintf(stderr, "Could not allocate enough memory\n");
    exit(42);
  }

  // Magical stuff for creating the image
  // ...

  // Write the tiff tags to the file
  TIFFSetField(output, TIFFTAG_IMAGEWIDTH, width);
  TIFFSetField(output, TIFFTAG_IMAGELENGTH, height);
  TIFFSetField(output, TIFFTAG_COMPRESSION, COMPRESSION_DEFLATE);
  TIFFSetField(output, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);
  TIFFSetField(output, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_RGB);
  TIFFSetField(output, TIFFTAG_BITSPERSAMPLE, 8);
  TIFFSetField(output, TIFFTAG_SAMPLESPERPIXEL, 3);

  // Actually write the image
  if(TIFFWriteEncodedStrip(output, 0, raster, width * height * 3) == 0){
    fprintf(stderr, "Could not write image\n");
    exit(42);
  }

  TIFFClose(output);
}
```

Code: write.c

You can see from this code some of the things that we have discussed in theory. The image has three samples per pixel, each of eight bits. This means that the image is a 24 bit RGB image. If this was a black and white or gray scale image, then this value would be one. The tag PHOTOMETRIC_RGB says that the image data is stored within the strips themselves (as opposed to being paletted) -- more about this in a second.

Other values for samples per pixel?

In this example, we have three samples per pixel. If this was a black and white image, or a gray scale image, then we would have one sample per pixel. There are other valid values for here as well -- for instance, sometimes people will store a transparency value for a given pixel, which is called an alpha channel. This would result in having four samples per pixel. It is possible to have an arbitrary number of samples per pixel, which is good if you need to pack in extra information about a pixel. *Note that doing this can break image viewers which make silly assumptions -- I once had to write code for a former employer to strip out alpha channels and the like so that their PDF generator wouldn't crash.*

The other interesting thing to discuss here is the planar configuration of the image. Here we have specified PLANARCONFIG_CONTIG, which means that the red green and blue information for a given pixel is grouped together in the strips of image data. The other option is PLANARCONFIG_SEPARATE, where the red samples for the image are stored together, then the blue samples, and finally the green samples.

Writing a paletted color image

So how do we write a paletted version of this image? Well, libtiff makes this really easy -- all we need to do is change the value of TIFFTAG_PHOTOMETRIC to PHOTOMETRIC_PALETTE. It's not really worth including an example in this chapter, given it's a one word change.

Reading a color image

Now all we have to do is work out how to read other people's color and gray scale images reliably, and we're home free. Initially I was very tempted to not tell you about the TIFFReadRGBAStrip() and TIFFReadRGBBSTile() calls, which hide some of the potential ugliness from the caller. These functions have some limitations I'm not astoundingly happy with. To quote the TIFFReadRGBAStrip() man page:

TIFFReadRGBAStrip reads a single strip of a strip-based image into memory, storing the result in the user supplied RGBA raster. The raster is assumed to be an array of width times rowsperstrip 32-bit entries, where width is the width of the image (TIFFTAG_IMAGEWIDTH) and rowsperstrip is the maximum lines in a strip (TIFFTAG_ROWSPERSTRIP).

The strip value should be the strip number (strip zero is the first) as returned by the TIFFComputeStrip function, but always for sample 0.

*Note that the raster is assume to be organized such that the pixel at location (x,y) is raster[y*width+x]; with the raster origin in the lower-left hand corner of the strip. That is bottom to top organization. When reading a partial last strip in the file the last line of the image will begin at the beginning of the buffer.*

Raster pixels are 8-bit packed red, green, blue, alpha samples. The macros TIFFGetR, TIFFGetG, TIFFGetB, and TIFFGetA should be used to access individual samples. Images without Associated Alpha matting information have a constant Alpha of 1.0 (255).

See the TIFFRGBAImage(3T) page for more details on how various image types are converted to RGBA values.

NOTES

Samples must be either 1, 2, 4, 8, or 16 bits. Colorimetric samples/pixel must be either 1, 3, or 4 (i.e. SamplesPerPixel minus ExtraSamples).

Palette image colormaps that appear to be incorrectly written as 8-bit values are automatically scaled to 16-bits.

TIFFReadRGBAStrip is just a wrapper around the more general TIFFRGBAImage(3T) facilities. It's main advantage over the similar TIFFReadRGBAImage() function is that for large images a single buffer capable of holding the whole image doesn't need to be allocated, only enough for one strip. The TIFFReadRGBATile() function does a similar operation for tiled images.

There are a couple of odd things about this function -- it defines (0, 0) to be in a different location from all the other code that we have been writing. In all the previous code we have written, the (0, 0) point has been in the top left of the image. This call defines (0, 0) to be in the bottom left. The other limitation to be aware of is that not all valid values for bits per sample are supported. If you find these quirks unacceptable, then remember that you can still use TIFFReadEncodedStrip() in the same manner that we did for the black and white images in the previous chapter...

```
#include <stdio.h>
#include <tiffio.h>

int main(int argc, char *argv[]){
  TIFF *image;
  uint32 width, height, *raster;
  tsize_t stripSize;
  unsigned long imagesize, c, d, e;

  // Open the TIFF image
  if((image = TIFFOpen(argv[1], "r")) == NULL){
    fprintf(stderr, "Could not open incoming image\n");
    exit(42);
  }

  // Find the width and height of the image
  TIFFGetField(image, TIFFTAG_IMAGEWIDTH, &width);
  TIFFGetField(image, TIFFTAG_IMAGELENGTH, &height);
  imagesize = height * width + 1;

  if((raster = (uint32 *) malloc(sizeof(uint32) * imagesize)) == NULL){
    fprintf(stderr, "Could not allocate enough memory\n");
    exit(42);
  }

  // Read the image into the memory buffer
  if(TIFFReadRGBAStrip(image, 0, raster) == 0){
    fprintf(stderr, "Could not read image\n");
    exit(42);
  }

  // Here I fix the reversal of the image (vertically) and show you how to get the colo
ues from each pixel
  d = 0;
  for(e = height - 1; e != -1; e--){
    for(c = 0; c < width; c++){
      // Red = TIFFGetR(raster[e * width + c]);
      // Green = TIFFGetG(raster[e * width + c]);
      // Blue = TIFFGetB(raster[e * width + c]);
    }
  }

  TIFFClose(image);
}
```

Code: read.c

Storing TIFF data in places other than files

All of these examples that I have included to this point have read and written with files. There are many scenarios when you wouldn't want to store your image data in a file, but would still want to use libtiff and TIFF. For example, you might have customer pictures for id cards, and these would be stored in a database.

The example which I am most familiar with is PDF documents. In PDF files, you can embed images into the document. These images can be in a subset of TIFF if desired, and TIFF is quite clearly the choice for black and white images.

An expanded example

If you need more information about hooking the file input and output functions within libtiff than this chapter allows, then have a look at the images.c file in Panda, my PDF library. The web pages for Panda can be found at http://www.stillhq.com [4] .

libtiff allows you to replace the file input and output functions in the library with your own. This is done with the TIFFClientOpen() method. Here's an example (please note this code wont compile, and is just to describe the main concepts:

```
// Please note that this code wont compile, and is intended to only show you
// the sturcture of TIFFClient* calls

#include <tiffio.h>
#include <pthread.h>

// Function prototypes
static tsize_t libtiffDummyReadProc (thandle_t fd, tdata_t buf, tsize_t size);
static tsize_t libtiffDummyWriteProc (thandle_t fd, tdata_t buf, tsize_t size);
static toff_t libtiffDummySeekProc (thandle_t fd, toff_t off, int i);
static int libtiffDummyCloseProc (thandle_t fd);

// We need globals because of the callbacks (they don't allow us to pass state)
char *globalImageBuffer;
unsigned long globalImageBufferOffset;

// This mutex keeps the globals safe by ensuring only one user at a time
pthread_mutex_t convMutex = PTHREAD_MUTEX_INITIALIZER;

TIFF *conv;

// Lock the mutex
pthread_mutex_lock (&convMutex);

globalImageBuffer = NULL;
globalImageBufferOffset = 0;

// Open the dummy document (which actually only exists in memory)
conv = TIFFClientOpen ("dummy", "w", (thandle_t) - 1, libtiffDummyReadProc,
            libtiffDummyWriteProc, libtiffDummySeekProc,
            libtiffDummyCloseProc, NULL, NULL, NULL);

// Setup the image as if it was any other tiff image here, including set-
ting tags


// Actually do the client open
TIFFWriteEncodedStrip (conv, 0, stripBuffer, imageOffset);

// Unlock the mutex
pthread_mutex_unlock (&convMutex);

//...

////////////////// Callbacks to libtiff

static tsize_t
libtiffDummyReadProc (thandle_t fd, tdata_t buf, tsize_t size)
{
  // Return the amount of data read, which we will always set as 0 because
  // we only need to be able to write to these in-memory tiffs
  return 0;
```

```
}

static tsize_t
libtiffDummyWriteProc (thandle_t fd, tdata_t buf, tsize_t size)
{
  // libtiff will try to write an 8 byte header into the tiff file. We need
  // to ignore this because PDF does not use it...
  if ((size == 8) && (((char *) buf)[0] == 'I') && (((char *) buf)[1] == 'I')
     && (((char *) buf)[2] == 42))
    {
    // Skip the header -- little endian
    }
  else if ((size == 8) && (((char *) buf)[0] == 'M') &&
       (((char *) buf)[1] == 'M') &&y (((char *) buf)[2] == 42))
    {
    // Skip the header -- big endian
    }
  else
    {
    // Have we done anything yet?
    if (globalImageBuffer == NULL)
    if((globalImageBuffer = (char *) malloc (size * sizeof (char))) == NULL)
        {
          fprintf(stderr, "Memory allocation error\n");
          exit(42);
        }

    // Otherwise, we need to grow the memory buffer
    else
    {
      if ((globalImageBuffer = (char *) realloc (globalImageBuffer,
                              (size * sizeof (char)) +
                              globalImageBufferOffset)) == NULL)
        fprintf(stderr, "Could not grow the tiff conversion memory buffer\n");
            exit(42);
    }

    // Now move the image data into the buffer
    memcpy (globalImageBuffer + globalImageBufferOffset, buf, size);
    globalImageBufferOffset += size;
    }

  return (size);
}

static toff_t
libtiffDummySeekProc (thandle_t fd, toff_t off, int i)
{
  // This appears to return the location that it went to
  return off;
}

static int
libtiffDummyCloseProc (thandle_t fd)
{
  // Return a zero meaning all is well
  return 0;
}
```

Code: client.c

Storing more than one image inside a TIFF

A common request is to be able to store more than one image inside a single TIFF file. This is done with multiple directories, which is one of the reasons they're described at the start of this chapter.

Before embarking on a multiple image per TIFF solution, it is important to remember that the TIFF specification doesn't require TIFF image viewers to implement support for multiple images per TIFF, and many viewers in fact do not [5] .

Anyway, now that I've nagged you, putting multiple images into one TIFF file is really easy, just call the **TIFFWriteDirectory**(), which writes out the current image so you can move onto the next one. The following sample program is an example of this...

```
#include <stdio.h>
#include <tiffio.h>
#include <unistd.h>
#include <string.h>

void usage (char *, int);

int
main (int argc, char *argv[])
{
  TIFF *input, *output;
  uint32 width, height;
  tsize_t stripSize, stripNumber;
  unsigned long x, y;
  char *inputFilename = NULL, *outputFilename = NULL, *raster, *roff, optchar;
  int count = 4, i;

  ////////////////////////////////////////////////////////////////////////////
  // Parse the command line options
  while ((optchar = getopt (argc, argv, "i:o:c:")) != -1)
    {
      switch (optchar)
    {
    case 'i':
      inputFilename = (char *) strdup (optarg);
      break;

    case 'o':
      outputFilename = (char *) strdup (optarg);
      break;

    case 'c':
      count = atoi(optarg);
      break;

    default:
      usage(argv[0], 0);
      break;
    }
    }

  // Open the input TIFF image
  if ((inputFilename == NULL) ||
      (input = TIFFOpen (inputFilename, "r")) == NULL)
    {
      fprintf (stderr, "Could not open incoming input %s\n", inputFilename);
      usage (argv[0], 42);
    }

  // Open the output TIFF
  if ((outputFilename == NULL) ||
```

```
    (output = TIFFOpen (outputFilename, "w")) == NULL)
  {
    fprintf (stderr, "Could not open outgoing input %s\n", outputFilename);
    usage (argv[0], 42);
  }

// Find the width and height of the input
TIFFGetField (input, TIFFTAG_IMAGEWIDTH, &width);
TIFFGetField (input, TIFFTAG_IMAGELENGTH, &height);

////////////////////////////////////////////////////////////////////////
// Grab some memory
if ((raster = (char *) malloc (sizeof (char) * width * height * 3)) == NULL)
  {
    fprintf (stderr, "Could not allocate enough memory for input raster\n");
    exit (42);
  }

////////////////////////////////////////////////////////////////////////
// Read the input into the memory buffer
// todo: I couldn't use TIFFReadRGBAStrip here, because it gets confused
stripSize = TIFFStripSize (input);
roff = raster;
for (stripNumber = 0; stripNumber < TIFFNumberOfStrips (input);
     stripNumber++)
  {
    roff += TIFFReadEncodedStrip (input, stripNumber, roff, stripSize);
  }

////////////////////////////////////////////////////////////////////////
// Write the image buffer to the file, we do this c times
for(i = 0; i < count; i++){
  printf(".");
  fflush(stdout);

  // todo: We need to copy tags from the input image to the output image
  TIFFSetField (output, TIFFTAG_IMAGEWIDTH, width);
  TIFFSetField (output, TIFFTAG_IMAGELENGTH, height);
  TIFFSetField (output, TIFFTAG_COMPRESSION, COMPRESSION_DEFLATE);
  TIFFSetField (output, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);
  TIFFSetField (output, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_RGB);
  TIFFSetField (output, TIFFTAG_BITSPERSAMPLE, 8);
  TIFFSetField (output, TIFFTAG_SAMPLESPERPIXEL, 3);
  // todo: balance this off with having 8 kb per strip...
  TIFFSetField (output, TIFFTAG_ROWSPERSTRIP, 100000);


  if (TIFFWriteEncodedStrip (output, 0, raster,
                width * height * 3 * sizeof (char)) == 0)
    {
  fprintf (stderr, "Could not write the output image\n");
  exit (42);
    }

  // Flush this subfile and move onto the next one
  if(TIFFWriteDirectory(output) == 0){
    fprintf(stderr, "Error writing subfile %d\n", i);
    exit(44);
  }
}
printf("\n");

// Cleanup
TIFFClose (input);
TIFFClose (output);
free (raster);
```

```
}

void
usage (char *cmd, int exitamt)
{
  fprintf (stderr, "Bad command line arguements...\n\n");
  fprintf (stderr, "Usage: %s -i <inputfile> -o <outputfile> -c <count>\n",
       cmd);
  exit (exitamt);
}
```

Code: create.c

Figure 3-8. Four pictures of my son Andrew

Reading more than one image inside a TIFF

We should also probably know how to get to these images once we have more than one image inside a single TIFF file...

This example below demonstrates how to do this.

```
#include <stdio.h>
#include <tiffio.h>
#include <unistd.h>
#include <string.h>

void usage (char *, int);

int
```

```
main (int argc, char *argv[])
{
  TIFF *input, *output;
  uint32 width, height;
  tsize_t stripSize, stripNumber;
  unsigned long x, y;
  char *inputFilename = NULL, *outputFilename = NULL,
    outputFilenameActual[200], *raster, *roff, optchar;
  int count;

  ////////////////////////////////////////////////////////////////////////
  // Parse the command line options
  while ((optchar = getopt (argc, argv, "i:o:")) != -1)
    {
      switch (optchar)
    {
    case 'i':
      inputFilename = (char *) strdup (optarg);
      break;

    case 'o':
      outputFilename = (char *) strdup (optarg);
      break;

    default:
      usage(argv[0], 0);
      break;
    }
    }

  // Check the output parent name
  if(outputFilename == NULL){
    fprintf(stderr, \
        "You need to specify a name for the series of output files\n");
    usage(argv[0], 42);
  }

  // Open the input TIFF image
  if ((inputFilename == NULL) ||
      (input = TIFFOpen (inputFilename, "r")) == NULL)
    {
      fprintf (stderr, "Could not open incoming input %s\n", inputFilename);
      usage (argv[0], 42);
    }

  ////////////////////////////////////////////////////////////////////////
  // Grab a sub file from the input image and move it to a separate file. We do
  // this forever (until we break down below)...
  for(count = 0;; count++){
    // Find the width and height of the input
    TIFFGetField (input, TIFFTAG_IMAGEWIDTH, &width);
    TIFFGetField (input, TIFFTAG_IMAGELENGTH, &height);

    ////////////////////////////////////////////////////////////////////////
    // Grab some memory
    if ((raster = (char *) malloc (sizeof (char) * width * height * 3)) ==
    NULL)
      {
    fprintf (stderr,
         "Could not allocate enough memory for input raster\n");
    exit (42);
      }

    ////////////////////////////////////////////////////////////////////////
    // Read the input into the memory buffer
    // todo: I couldn't use TIFFReadRGBAStrip here, because it gets confused
```

```
    stripSize = TIFFStripSize (input);
    roff = raster;
    for (stripNumber = 0; stripNumber < TIFFNumberOfStrips (input);
     stripNumber++)
      {
    roff += TIFFReadEncodedStrip (input, stripNumber, roff, stripSize);
      }

    ////////////////////////////////////////////////////////////////////////
    // Open the output TIFF
    snprintf(outputFilenameActual, 200, "%s-%d.tif", outputFilename, count);
    if ((output = TIFFOpen (outputFilenameActual, "w")) == NULL)
      {
    fprintf (stderr, "Could not open outgoing input %s\n", outputFilename);
    usage (argv[0], 42);
      }

    printf(".");
    fflush(stdout);

    // todo: We need to copy tags from the input image to the output image
    TIFFSetField (output, TIFFTAG_IMAGEWIDTH, width);
    TIFFSetField (output, TIFFTAG_IMAGELENGTH, height);
    TIFFSetField (output, TIFFTAG_COMPRESSION, COMPRESSION_DEFLATE);
    TIFFSetField (output, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);
    TIFFSetField (output, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_RGB);
    TIFFSetField (output, TIFFTAG_BITSPERSAMPLE, 8);
    TIFFSetField (output, TIFFTAG_SAMPLESPERPIXEL, 3);
    // todo: balance this off with having 8 kb per strip...
    TIFFSetField (output, TIFFTAG_ROWSPERSTRIP, 100000);

    // Copy the subfile to a output location
    if (TIFFWriteEncodedStrip (output, 0, raster,
                   width * height * 3 * sizeof (char)) == 0)
      {
    fprintf (stderr, "Could not write the output image\n");
    exit (42);
      }

    ////////////////////////////////////////////////////////////////////////
    // Flush this subfile and move onto the next one
    if(TIFFReadDirectory(input) == 0){
      printf(" No more subfiles");
      break;
    }

    ////////////////////////////////////////////////////////////////////////
    // Doing correct cleanup with a loop like this is important...
    free(raster);
  }
  printf("\n");

  // Cleanup
  TIFFClose (input);
  TIFFClose (output);
  free (raster);
}

void
usage (char *cmd, int exitamt)
{
  fprintf (stderr, "Bad command line arguements...\n\n");
  fprintf (stderr, "Usage: %s -i <inputfile> -o <outputfile> -c <count>\n",
       cmd);
  exit (exitamt);
}
```

Code: read.c

Man pages

I have included the man pages to some of the more useful libtiff commands, so that you have them for reference when you need them.

tiff2bw

NAME

tiff2bw - convert a color *TIFF* image to greyscale

SYNOPSIS

tiff2bw [options] *input.tif output.tif*

DESCRIPTION

Tiff2bw converts an *RGB* or Palette color *TIFF* image to a greyscale image by combining percentages of the red, green, and blue channels. By default, output samples are created by taking 28% of the red channel, 59% of the green channel, and 11% of the blue channel. To alter these percentages, the and options may be used.

OPTIONS

- **-c** Specify a compression scheme to use when writing image data: **"-c none"** for no compression, **"-c packbits"** for the PackBits compression algorithm, **"-c zip** for the Deflate compression algorithm, **"-c g3** for the CCITT Group 3 compression algorithm, **"-c g4** for the CCITT Group 4 compression algorithm, and **"-c lzw"** for Lempel-Ziv & Welch (the default).
- **-r** Write data with a specified number of rows per strip; by default the number of rows/strip is selected so that each strip is approximately 8 kilobytes.
- **-R** Specify the percentage of the red channel to use (default 28).
- **-G** Specify the percentage of the green channel to use (default 59).
- **-B** Specify the percentage of the blue channel to use (default 11).

SEE ALSO

pal2rgb (1), tiffinfo (1), tiffcp (1), tiffmedian (1), libtiff (3)

tiff2ps

NAME

tiff2ps - convert a *TIFF* image to *(Ps\(tm

SYNOPSIS

tiff2ps [*options*] *"input.tif ..."*

DESCRIPTION

tiff2ps reads *TIFF* images and writes *(Ps or Encapsulated *(Ps (EPS) on the standard output. By default, *tiff2ps* writes Encapsulated *(Ps for the first image in the specified *TIFF* image file.

By default, *tiff2ps* will generate *(Ps that fills a printed area specified by the *TIFF* tags in the input file. If the file does not contain *XResolution* or *YResolution* tags, then the printed area is set according to the image dimensions. The **-w** and **-h** options (see below) can be used to set the dimensions of the printed area in inches; overriding any relevant *TIFF* tags.

The *(Ps generated for *RGB*, palette, and *CMYK* images uses the *colorimage* operator. The *(Ps generated for greyscale and bilevel images uses the *image* operator. When the *colorimage* operator is used, *(Ps code to emulate this operator on older *(Ps printers is also generated. Note that this emulation code can be very slow.

Color images with associated alpha data are composited over a white background.

OPTIONS

- **-1** Generate *(Ps Level I (the default).
- **-2** Generate *(Ps Level II.
- **-a** Generate output for all IFDs (pages) in the input file.
- **-d** Set the initial *TIFF* directory to the specified directory number. (NB: directories are numbered starting at zero.) This option is useful for selecting individual pages in a multi-page (e.g. facsimile) file.
- **-e** Force the generation of Encapsulated *(Ps.
- **-h** Specify the vertical size of the printed area (in inches).
- **-i** Enable/disable pixel interpolation. This option requires a single numeric value: zero to disable pixel interpolation and non-zero to enable. The default is enabled.
- **-m** Where possible render using the **imagemask** *(Ps operator instead of the image operator. When this option is specified *tiff2ps* will use **imagemask** for rendering 1 bit deep images. If this option is not specified or if the image depth is greater than 1 then the image operator is used.
- **-o** Set the initial *TIFF* directory to the *IFD* at the specified file offset. This option is useful for selecting thumbnail images and the like which are hidden using the SubIFD tag.
- **-p** Force the generation of (non-Encapsulated) *(Ps.
- **-s** Generate output for a single IFD (page) in the input file.
- **-w** Specify the horizontal size of the printed area (in inches).
- **-z** When generating *(Ps Level II, data is scaled so that it does not image into the *deadzone* on a page (the outer margin that the printing device is unable to mark). This option suppresses this behaviour. When *(Ps Level I is generated, data is imaged to the entire printed page and this option has no affect.

EXAMPLES

The following generates *(Ps Level II for all pages of a facsimile: tiff2ps -a2 fax.tif | lpr Note also that if you have version 2.6.1 or newer of Ghostscript then you can efficiently preview facsimile generated with the above command.

To generate Encapsulated *(Ps for a the image at directory 2 of an image use: tiff2ps -d 1 foo.tif (notice that directories are numbered starting at zero.)

BUGS

Because *(Ps does not support the notion of a colormap, 8-bit palette images produce 24-bit *(Ps images. This conversion results in output that is six times bigger than the original image and which takes a long time to send to a printer over a serial line. Matters are even worse for 4-, 2-, and 1-bit palette images.

BUGS

Does not handle tiled images when generating PS Level I output.

SEE ALSO

pal2rgb (1), tiffinfo (1), tiffcp (1), tiffgt (1), tiffmedian (1), tiff2bw (1), tiffsv (1), libtiff (3)

tiffcmp

NAME

tiffcmp - compare two *TIFF* files

SYNOPSIS

tiffcmp [*options*] *"file1.tif file2.tif"*

DESCRIPTION

Tiffcmp compares the tags and data in two files created according to the Tagged Image File Format, Revision 6.0. The schemes used for compressing data in each file are immaterial when data are compared-data are compared on a scanline-by-scanline basis after decompression. Most directory tags are checked; notable exceptions are: *GrayResponseCurve* , *ColorResponseCurve* , and *ColorMap* tags. Data will not be compared if any of the *BitsPerSample* , *SamplesPerPixel* , or *ImageWidth* values are not equal. By default, *tiffcmp* will terminate if it encounters any difference.

OPTIONS

- **-l** List each byte of image data that differs between the files.
- **-t** Ignore any differences in directory tags.

BUGS

Tags that are not recognized by the library are not compared; they may also generate spurious diagnostics.

The image data of tiled files is not compared, since the TIFFReadScanline() function is used. A error will be reported for tiled files.

The pixel and/or sample number reported in differences may be off in some exotic cases.

SEE ALSO

pal2rgb (1), tiffinfo (1), tiffcp (1), tiffmedian (1), libtiff (3)

Sample output

tiffcmp shows the differences in tags between images, for example, when I compare the input and output images from the pixel example in this chapter:

```
[mikal@localhost tiff-pixels]$ tiffcmp input.tif output.tif
ImageWidth: 256 520
[mikal@localhost tiff-pixels]$
```

tiffcp

NAME

tiffcp - copy (and possibly convert) a *TIFF* file

SYNOPSIS

tiffcp [*options*] *"src1.tif ... srcN.tif dst.tif"*

DESCRIPTION

tiffcp combines one or more files created according to the Tag Image File Format, Revision 6.0 into a single *TIFF* file. Because the output file may be compressed using a different algorithm than the input files, *tiffcp* is most often used to convert between different compression schemes.

By default, *tiffcp* will copy all the understood tags in a *TIFF* directory of an input file to the associated directory in the output file.

tiffcp can be used to reorganize the storage characteristics of data in a file, but it is explicitly intended to not alter or convert the image data content in any way.

OPTIONS

- **-b image** subtract the following monochrome image from all others processed. This can be used to remove a noise bias from a set of images. This bias image is typlically an image of noise the camera saw with its shutter closed.

- **-B** Force output to be written with Big-Endian byte order. This option only has an effect when the output file is created or overwritten and not when it is appended to.
- **-C** Suppress the use of "strip chopping" when reading images that have a single strip/tile of uncompressed data.
- **-c** Specify the compression to use for data written to the output file: **none** for no compression, **packbits** for PackBits compression, **lzw** for Lempel-Ziv & Welch compression, **jpeg** for baseline JPEG compression, **zip** for Deflate compression, **g3** for CCITT Group 3 (T.4) compression, and **g4** for CCITT Group 4 (T.6) compression. By default *tiffcp* will compress data according to the value of the *Compression* tag found in the source file. The *CCITT* Group 3 and Group 4 compression algorithms can only be used with bilevel data. Group 3 compression can be specified together with several T.4-specific options: **1d** for 1-dimensional encoding, **2d** for 2-dimensional encoding, and **fill** to force each encoded scanline to be zero-filled so that the terminating EOL code lies on a byte boundary. Group 3-specific options are specified by appending a ":"-separated list to the "g3" option; e.g. **"-c g3:2d:fill"** to get 2D-encoded data with byte-aligned EOL codes. *LZW* compression can be specified together with a *predictor* value. A predictor value of 2 causes each scanline of the output image to undergo horizontal differencing before it is encoded; a value of 1 forces each scanline to be encoded without differencing. LZW-specific options are specified by appending a ":"-separated list to the "lzw" option; e.g. **"-c lzw:2"** for *LZW* compression with horizontal differencing.
- **-f** Specify the bit fill order to use in writing output data. By default, *tiffcp* will create a new file with the same fill order as the original. Specifying **"-f lsb2msb"** will force data to be written with the FillOrder tag set to *LSB2MSB*, while **"-f msb2lsb"** will force data to be written with the FillOrder tag set to *MSB2LSB*.
- **-l** Specify the length of a tile (in pixels). *tiffcp* attempts to set the tile dimensions so that no more than 8 kilobytes of data appear in a tile.
- **-L** Force output to be written with Little-Endian byte order. This option only has an effect when the output file is created or overwritten and not when it is appended to.
- **-M** Suppress the use of memory-mapped files when reading images.
- **-p** Specify the planar configuration to use in writing image data that has one 8-bit sample per pixel. By default, *tiffcp* will create a new file with the same planar configuration as the original. Specifying **"-p contig"** will force data to be written with multi-sample data packed together, while **"-p separate"** will force samples to be written in separate planes.
- **-r** Specify the number of rows (scanlines) in each strip of data written to the output file. By default, *tiffcp* attempts to set the rows/strip that no more than 8 kilobytes of data appear in a strip.
- **-s** Force the output file to be written with data organized in strips (rather than tiles).
- **-t** Force the output file to be written wtih data organized in tiles (rather than strips). options can be used to force the resultant image to be written as strips or tiles of data, respectively.
- **-w** Specify the width of a tile (in pixels). *tiffcp* attempts to set the tile dimensions so that no more than 8 kilobytes of data appear in a tile. *tiffcp* attempts to set the tile dimensions so that no more than 8 kilobytes of data appear in a tile.
- **-,={character}** substitute {character} for ',' in parsing image directory indices in files. This is necessary if filenames contain commas. Note that ',=' with whitespace immediately following will disable the special meaning of the ',' entirely. See examples.

EXAMPLES

The following concatenates two files and writes the result using *LZW* encoding: tiffcp -c lzw a.tif b.tif result.tif

To convert a G3 1d-encoded *TIFF* to a single strip of G4-encoded data the following might be used: tiffcp -c g4 -r 10000 g3.tif g4.tif (1000 is just a number that is larger than the number of rows in the source file.)

To extract a selected set of images from a multi-image TIFF file, the file name may be immediately followed by a ',' separated list of image directory indices. The first image is always in directory 0. Thus, to copy the 1st and 3rd images of image file "album.tif" to "result.tif": tiffcp album.tif,0,2 result.tif

Given file "CCD.tif" whose first image is a noise bias followed by images which include that bias, subtract the noise from all those images following it (while decompressing) with the command: tiffcp -c none -b CCD.tif CCD.tif,1, result.tif

If the file above were named "CCD,X.tif", the "-,=" option would be required to correctly parse this filename with image numbers, as follows: tiffcp -c none -,=% -b CCD,X.tif CCD,X%1%.tif result.tif

SEE ALSO

pal2rgb (1), tiffinfo (1), tiffcmp (1), tiffmedian (1), tiffsplit (1), libtiff (3)

tiffdither

NAME

tiffdither - convert a greyscale image to bilevel using dithering

SYNOPSIS

tiffdither [*options*] *input.tif output.tif*

DESCRIPTION

tiffdither converts a single channel 8-bit greyscale image to a bilevel image using Floyd-Steinberg error propagation with threholding.

OPTIONS

- **-c** Specify the compression to use for data written to the output file: **none** for no compression, **packbits** for PackBits compression, **lzw** for Lempel-Ziv & Welch compression, **zip** for Deflate compression, **g3** for CCITT Group 3 (T.4) compression, and **g4** for CCITT Group 4 (T.6) compression. By default *tiffdither* will compress data according to the value of the *Compression* tag found in the source file. The *CCITT* Group 3 and Group 4 compression algorithms can only be used with bilevel data. Group 3 compression can be specified together with several T.4-specific options: **1d** for 1-dimensional encoding, **2d** for 2-dimensional encoding, and **fill** to force each encoded scanline to be zero-filled so that the terminating EOL code lies on a byte boundary. Group 3-specific options are specified by appending a ":"-separated list to the "g3" option; e.g. **"-c g3:2d:fill"** to get 2D-encoded data with

byte-aligned EOL codes. *LZW* compression can be specified together with a *predictor* value. A predictor value of 2 causes each scanline of the output image to undergo horizontal differencing before it is encoded; a value of 1 forces each scanline to be encoded without differencing. LZW-specific options are specified by appending a ":"-separated list to the "lzw" option; e.g. **"-c lzw:2"** for *LZW* compression with horizontal differencing.

- **-f** Specify the bit fill order to use in writing output data. By default, *tiffdither* will create a new file with the same fill order as the original. Specifying **"-f lsb2msb"** will force data to be written with the FillOrder tag set to *LSB2MSB* , while **"-f msb2lsb"** will force data to be written with the FillOrder tag set to *MSB2LSB* .
- **-t** Set the threshold value for dithering. By default the threshold value is 128.

NOTES

The dither algorithm is taken from the *tiffmedian (1)* program (written by Paul Heckbert).

SEE ALSO

pal2rgb (1), fax2tiff (1), tiffinfo (1), tiffcp (1), tiff2bw (1), libtiff (3)

Sample output

tiffdither takes gray scale images and dithers them into black and white images. For instance, the gray scale example from earlier in this tutorial produced the following image:

Figure 3-9. The gray scale input image

This image, after being pushed through **tiffdither** looks something like:

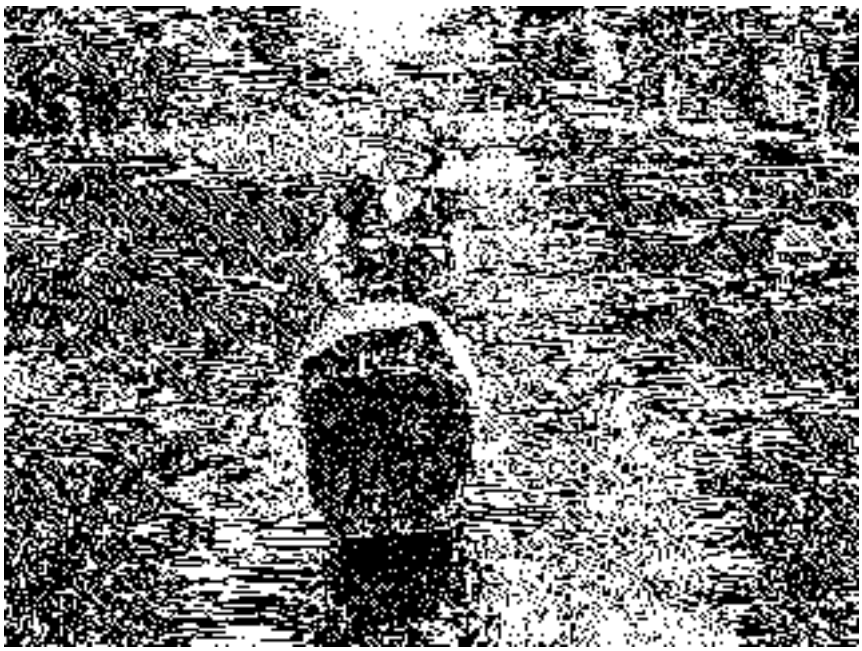

Figure 3-10. The dithered output image

tiffdump

NAME

tiffdump - print verbatim information about *TIFF* files

SYNOPSIS

tiffdump [*options*] *"name ..."*

DESCRIPTION

tiffdump displays directory information from files created according to the Tag Image File Format, Revision 6.0. The header of each *TIFF* file (magic number, version, and first directory offset) is displayed, followed by the tag contents of each directory in the file. For each tag, the name, datatype, count, and value(s) is displayed. When the symbolic name for a tag or datatype is known, the symbolic name is displayed followed by it's numeric (decimal) value. Tag values are displayed enclosed in "<>" characters immediately preceded by the value of the count field. For example, an *ImageWidth* tag might be displayed as "ImageWidth (256) SHORT (3) 1<800>".

tiffdump is particularly useful for investigating the contents of *TIFF* files that *libtiff* does not understand.

OPTIONS

- **-h** Force numeric data to be printed in hexadecimal rather than the default decimal.
- **-o** Dump the contents of the *IFD* at the a particular file offset. The file offset may be specified using the usual C-style syntax; i.e. a leading "0x" for hexadecimal and a leading "0" for octal.

SEE ALSO

tiffinfo (1), libtiff (3)

Sample output

tiffdump shows you heaps of useful things about a TIFF image, basically anything stored in a tag. Some sample output is:

```
[mikal@localhost tutorial-imaging]$ tiffdump tiff-figure10.tif
tiff-figure10.tif:
Magic: 0x4949 <little-endian> Version: 0x2a
Directory 0: offset 38228 (0x9554) next 0 (0)
ImageWidth (256) SHORT (3) 1<640>
ImageLength (257) SHORT (3) 1<479>
BitsPerSample (258) SHORT (3) 1<1>
Compression (259) SHORT (3) 1<32773>
Photometric (262) SHORT (3) 1<1>
ImageDescription (270) ASCII (2) 41<Dithered B&W version of  ...>
StripOffsets (273) LONG (4) 5<8 8141 16315 24528 32662>
SamplesPerPixel (277) SHORT (3) 1<1>
RowsPerStrip (278) SHORT (3) 1<102>
StripByteCounts (279) LONG (4) 5<8133 8174 8213 8134 5566>
PlanarConfig (284) SHORT (3) 1<1>
[mikal@localhost tutorial-imaging]$
```

tiffgt

NAME

tiffgt - display an image stored in a *TIFF* file (Silicon Graphics version)

SYNOPSIS

tiffgt [*options*] *"input.tif ..."*

DESCRIPTION

tiffgt displays one or more images stored using the Tag Image File Format, Revision 6.0. Each image is placed in a fixed size window that the user must position on the display (unless configured otherwise through X defaults). If the display has fewer than 24 bitplanes, or if the image does not warrant full color, then *RGB* color values are mapped to the closest values that exist in the colormap (this is done using the *rgbi* routine found in the graphics utility library

tiffgt correctly handles files with any of the following characteristics: BitsPerSample 1, 2, 4, 8, 16 SamplesPerPixel 1, 3, 4 (the 4th sample is ignored) PhotometricInterpretation 0 (min-is-white), 1 (min-is-black), 2 (RGB), 3 (palette), 6 (YCbCr) PlanarConfiguration 1 (contiguous), 2 (separate) Orientation 1 (top-left), 4 (bottom-left) Data may be organized as strips or tiles and may be compressed with any of the compression algorithms supported by the *libtiff (3)* library.

For palette images (\c *PhotomatricInterpretation =3), tiffgt* inspects the colormap values and assumes either 16-bit or 8-bit values according to the maximum value. That is, if no colormap entry greater than 255 is found, *tiffgt* assumes the colormap has only 8-bit values; otherwise it assumes 16-bit values. This inspection is done to handle old images written by previous (incorrect) versions of *libtiff* .

tiffgt can be used to display multiple images one-at-a-time. The left mouse button switches the display to the first image in the *next* file in the list of files specified on the command line. The right mouse button switches to the first image in the *previous* file in the list. The middle mouse button causes the first image in the first file specified on the command line to be displayed. In addition the following keyboard commands are recognized:

- **b** Use a *PhotometricIntepretation* of MinIsBlack in displaying the current image.
- **l** Use a *FillOrder* of lsb-to-msb in decoding the current image.
- **m** Use a *FillOrder* of msb-tolmsb in decoding the current image.
- **c** Use a colormap visual to display the current image.
- **r** Use a true color (24-bit RGB) visual to display the current image.
- **w** Use a *PhotometricIntepretation* of MinIsWhite in displaying the current image.
- **W** Toggle (enable/disable) display of warning messages from the *TIFF* library when decoding images.
- **E** Toggle (enable/disable) display of error messages from the *TIFF* library when decoding images.
- **z** Reset all parameters to their default settings (\c *FillOrder , PhotometricInterpretation ,* handling of warnings and errors).
- **PageUp** Display the previous image in the current file or the last image in the previous file.
- **PageDown** Display the next image in the current file or the first image in the next file.
- **Home** Display the first image in the current file.
- **End** Display the last image in the current file (unimplemented).

OPTIONS

- **-c** Force image display in a colormap window.
- **-d** Specify an image to display by directory number. By default the first image in the file is displayed. Directories are numbered starting at zero.
- **-e** Enable reporting of error messages from the *TIFF* library. By default *tiffgt* silently ignores images that cannot be read.
- **-f** Force *tiffgt* to run as a foreground process. By default *tiffgt* will place itself in the background once it has opened the requested image file.
- **-l** Force the presumed bit ordering to be *LSB* to *MSB.*
- **-m** Force the presumed bit ordering to be *MSB* to *LSB.*

- **-o** Specify an image to display by directory offset. By default the first image in the file is displayed. Directories offsets may be specified using C-style syntax; i.e. a leading "0x" for hexadecimal and a leading "0" for octal.
- **-p** Override the value of the *PhotometricInterpretation* tag; the parameter may be one of: *miniswhite , minisblack , rgb , palette , mask , separated , ycbcr ,* and *cielab .*
- **-r** Force image display in a full color window.
- **-s** Stop on the first read error. By default all errors in the input data are ignored and *tiffgt* does it's best to display as much of an image as possible.
- **-w** Enable reporting of warning messages from the *TIFF* library. By default *tiffgt* ignores warning messages generated when reading an image.
- **-v** Place information in the title bar describing what type of window (full color or colormap) is being used, the name of the input file, and the directory index of the image (if non-zero). By default, the window type is not shown in the title bar.

BUGS

Images wider and taller than the display are silently truncated to avoid crashing old versions of the window manager.

SEE ALSO

tiffdump (1), tiffinfo (1), tiffcp (1), libtiff (3)

tiffinfo

NAME

tiffinfo - print information about *TIFF* files

SYNOPSIS

tiffinfo [*options*] *"input.tif ..."*

DESCRIPTION

Tiffinfo displays information about files created according to the Tag Image File Format, Revision 6.0. By default, the contents of each *TIFF* directory in each file is displayed, with the value of each tag shown symbolically (where sensible).

OPTIONS

- **-c** Display the colormap and color/gray response curves, if present.
- **-D** In addition to displaying the directory tags, read and decompress all the data in each image (but not display it).
- **-d** In addition to displaying the directory tags, print each byte of decompressed data in hexadecimal.
- **-j** Display any \s-2JPEG\s0-related tags that are present.

- **-o** Set the initial *TIFF* directory according to the specified file offset. The file offset may be specified using the usual C-style syntax; i.e. a leading "0x" for hexadecimal and a leading "0" for octal.
- **-s** Display the offsets and byte counts for each data strip in a directory.
- **-z** Enable strip chopping when reading image data.
- **-#** Set the initial *TIFF* directory to # .

SEE ALSO

pal2rgb (1), tiffcp (1), tiffcmp (1), tiffmedian (1), libtiff (3)

Sample output

tiffinfo is probably the most useful command which comes with libtiff... In normal operation, it shows you nicely formatted information about the TIFF image, like so:

```
[mikal@localhost tutorial-imaging]$ tiffinfo tiff-figure9.tif
TIFF Directory at offset 0x40f7c
  Image Width: 640 Image Length: 480
  Bits/Sample: 8
  Compression Scheme: Deflate
  Photometric Interpretation: min-is-black
  Samples/Pixel: 1
  Rows/Strip: 100000
  Planar Configuration: single image plane
[mikal@localhost tutorial-imaging]$
```

When you ask nicely (with a -d), then you're shown random tiff data, which can be very handy:

```
[mikal@localhost tutorial-imaging]$ tiffinfo -d tiff-figure9.tif
TIFF Directory at offset 0x40f7c
  Image Width: 640 Image Length: 480
  Bits/Sample: 8
  Compression Scheme: Deflate
  Photometric Interpretation: min-is-black
  Samples/Pixel: 1
  Rows/Strip: 100000
  Planar Configuration: single image plane
Strip 0:
 59 5d 5c 55 4f 51 58 5d 5a 5a 57 51 4d 4d 4e 4d 58 54 55 5f 6c 76 7a 7c
 86 83 89 89 7f 7c 79 6e 61 57 70 7a 64 6f 7d 62 60 61 68 75 82 85 7e 76
 78 79 80 7a 66 5f 62 61 53 5b 62 6d 77 75 7a 8e a0 91 7a 66 5d 5d 61 62
 6f 69 65 64 63 61 63 68 76 79 74 76 86 8b 86 85 83 83 66 6a a0 b8 a7 a1
 94 a7 9d 72 5b 69 7b 7c 88 84 7f 7a 73 72 7e 8c 94 96 8a 75 6f 77 75 6a
 73 7d 85 85 84 89 90 94 8b 7d 74 7a 84 8a 8e 91 9c 99 87 7e 8d 93 85 79
 73 7e 88 87 80 7e 88 92 96 93 95 9d a1 9c 94 91 9c 8f 90 9e 9e 8a 7b 7b
 89 7a 68 5d 58 5b 66 71 81 82 7c 71 6c 6f 70 6e 5f 62 60 60 65 66 6f 84
 85 93 a4 a8 98 86 81 86 88 7e 83 7a 73 74 6f 7e 78 72 73 74 71 75 73 67
 72 73 77 85 90 8a 8f a7 ba cc cc c0 c1 c8 cf d7 dd e6 ec e5 e2 ef ed d4
 e4 e6 ea ee f1 f2 f1 ef f3 f3 f3 f2 f2 f2 f1 f1 f2 f1 f0 ef ee ed ec eb
 ... and so on
```

tiffmedian

NAME

tiffmedian - apply the median cut algorithm to data in a *TIFF* file

SYNOPSIS

tiffmedian [*options*] *input.tif output.tif*

DESCRIPTION

tiffmedian applys the median cut algorithm to an *RGB* image in *input.tif* to generate a palette image that is written to *output.tif* . The generated colormap has, by default, 256 entries. The image data is quantized by mapping each pixel to the closest color values in the colormap.

OPTIONS

- **-c** Specify the compression to use for data written to the output file: **none** for no compression, **packbits** for PackBits compression, **lzw** for Lempel-Ziv & Welch compression, and **zip** for Deflate compression. By default *tiffmedian* will compress data according to the value of the *Compression* tag found in the source file. *LZW* compression can be specified together with a *predictor* value. A predictor value of 2 causes each scanline of the output image to undergo horizontal differencing before it is encoded; a value of 1 forces each scanline to be encoded without differencing. LZW-specific options are specified by appending a ":"-separated list to the "lzw" option; e.g. **"-c lzw:2"** for *LZW* compression with horizontal differencing.
- **-C** Specify the number of entries to use in the generated colormap. By default all 256 entries/colors are used.
- **-f** Apply Floyd-Steinberg dithering before selecting a colormap entry.
- **-r** Specify the number of rows (scanlines) in each strip of data written to the output file. By default, *tiffmedian* attempts to set the rows/strip that no more than 8 kilobytes of data appear in a strip.

NOTES

This program is derived from Paul Heckbert's *median* program.

SEE ALSO

pal2rgb (1), tiffinfo (1), tiffcp (1), tiffcmp (1), libtiff (3)

"Color Image Quantization for Frame Buffer Display", Paul Heckbert, SIGGRAPH proceedings, 1982, pp. 297-307.

tiffsplit

NAME

tiffsplit - split a multi-image *TIFF* into single-image *TIFF* files

SYNOPSIS

tiffsplit *src.tif* [*prefix*]

DESCRIPTION

tiffsplit takes a multi-directory (page) *TIFF* file and creates one or more single-directory (page) *TIFF* files from it. The output files are given names created by concatenating a prefix, a lexically ordered suffix in the range [*aa*-*zz*], the suffix *.tif* (e.g. *xaa.tif , xab.tif , \... xzz.tif).* If a prefix is not specified on the command line, the default prefix of *x* is used.

OPTIONS

None.

BUGS

Only a select set of “known tags” is copied when spliting.

SEE ALSO

tiffcp (1), tiffinfo (1), libtiff (3)

tiffsv

NAME

tiffsv - save an image from the framebuffer in a *TIFF* file (Silicon Graphics version)

SYNOPSIS

tiffsv [*options*] *output.tif* [*”x1 x2 y1 y2”*]

DESCRIPTION

tiffsv saves all or part of the framebuffer in a file using the Tag Image File Format, Revision 6.0. By default, the image is saved with data samples packed (\c *PlanarConfiguration =1),* compressed with the Lempel-Ziv & Welch algorithm (\c *Compression =5),* and with each strip no more than 8 kilobytes. These characteristics can be overriden, or explicitly specified with the options described below.

OPTIONS

- **-b** Save the image as a greyscale image as if it were processed by *tiff2bw (1).* This option is included for compatibility with the standard *scrsave (6D)* program.
- **-c** Specify the compression to use for data written to the output file: **none** for no compression, **packbits** for PackBits compression, **jpeg** for baseline JPEG compression, **zip** for Deflate compression, and **lzw** for Lempel-Ziv & Welch compression (default). *LZW* compression can be specified together with a *predictor* value.

A predictor value of 2 causes each scanline of the output image to undergo horizontal differencing before it is encoded; a value of 1 forces each scanline to be encoded without differencing. LZW-specific options are specified by appending a ":"-separated list to the "lzw" option; e.g. **"-c lzw:2"** for *LZW* compression with horizontal differencing.

- **-p** Specify the planar configuration to use in writing image data. By default, *tiffsv* will create a new file with the data samples packed contiguously. Specifying **"-p contig"** will force data to be written with multi-sample data packed together, while **"-p separate"** will force samples to be written in separate planes.
- **-r** Specify the number of rows (scanlines) in each strip of data written to the output file. By default, *tiffsv* attempts to set the rows/strip that no more than 8 kilobytes of data appear in a strip.

NOTE

Except for the use of *TIFF,* this program is equivalent to the standard *scrsave* program. This means, for example, that you can use it in conjunction with the standard *icut* program simply by creating a link called *scrsave ,* or by creating a shell script called *scrsave* that invokes *tiffgt* with the appropriate options.

BUGS

If data are saved compressed and in separate planes, then the rows in each strip is silently set to one to avoid limitations in the *libtiff (3)* library.

SEE ALSO

scrsave (6D) pal2rgb (1), tiffdump (1), tiffgt (1), tiffinfo (1), tiffcp (1), tiffmedian (1), libtiff (3)

tifftopnm

NAME

tifftopnm - convert a TIFF file into a portable anymap

SYNOPSIS

tifftopnm [={alpha-filename,-}] [**--headerdump**] *tiff-filename*

DESCRIPTION

Reads a TIFF file as input. Produces a portable anymap as output. The type of the output file depends on the input file - if it's black & white, generates a *pbm* file; if it's grayscale, generates a *pgm* file; otherwise, a *ppm* file. The program tells you which type it is writing.

This program cannot read every possible TIFF file -- there are myriad variations of the TIFF format. However, it does understand monochrome and gray scale, RGB, RGBA (red/green/blue with alpha channel), CMYK (Cyan-Magenta-Yellow-Black ink color separation), and color palette TIFF files. An RGB file can have either single

plane (interleaved) color or multiple plane format. The program reads 1-8 and 16 bit-per-sample input, the latter in either bigendian or littlendian encoding. Tiff directory information may also be either bigendian or littendian.

One reason this program isn't as general as TIFF programs often are is that it does not use the TIFFRGBAImageGet() function of the TIFF library to read TIFF files. Rather, it uses the more primitive TIFFReadScanLine() function and decodes it itself.

There is no fundamental reason that this program could not read other kinds of TIFF files; the existing limitations are mainly because no one has asked for more.

The PNM output has the same maxval as the Tiff input, except that if the Tiff input is colormapped (which implies a maxval of 65535) the PNM output has a maxval of 255. Though this may result in lost information, such input images hardly ever actually have more color resolution than a maxval of 255 provides and people often cannot deal with PNM files that have maxval > 255. By contrast, a non-colormapped Tiff image that doesn't need a maxval > 255 doesn't *have* a maxval > 255, so when we see a non-colormapped maxval > 255, we take it seriously and produce a matching output maxval.

The *tiff-filename* argument names the regular file that contains the Tiff image. You cannot use Standard Input or any other special file because the Tiff library must be able to perform seeks on it.

OPTIONS

- **tifftopnm** creates a PGM (portable graymap) file containing the alpha channel values in the input image. If the input image doesn't contain an alpha channel, the *alpha-filename* file contains all zero (transparent) alpha values. If you don't specify **tifftopnm** does not generate an alpha file, and if the input image has an alpha channel, **tifftopnm** simply discards it.

  If you specify **-** as the filename, **tifftopnm** writes the alpha output to Standard Output and discards the image.

  See for one way to use the alpha output file.

- **--headerdump** Dump TIFF file information to stderr. This information may be useful in debugging TIFF file conversion problems.

  All options can be abbreviated to their shortest unique prefix.

SEE ALSO

AUTHOR

Derived by Jef Poskanzer from tif2ras.c, which is Copyright (c) 1990 by Sun Microsystems, Inc. Author: Patrick J. Naughton (naughton@wind.sun.com).

Example: Pixel enlargement

In the introduction to this tutorial I promised that I would include the code to draw the demonstration that raster images are made up of pixels. Well, this example is that code.

```
#include <stdio.h>
#include <tiffio.h>
#include <unistd.h>
#include <string.h>

void usage (char *, int);

#define EOFFSET 150

int
main (int argc, char *argv[])
{
  TIFF *input, *output;
  uint32 width, height, offset, offset2, xs = 0, ys = 0, xe = -1, ye = -
1;
  tsize_t stripSize, stripNumber;
  unsigned long x, y;
  char *inputFilename = NULL, *outputFilename = NULL, *raster, *roff,
    *enlarged, *rout, optchar;
  int xrep, yrep;
  float m;

  ////////////////////////////////////////////////////////////////////////
  // Parse the command line options
  while ((optchar = getopt (argc, argv, "i:o:x:y:w:l:")) != -1)
    {
      switch (optchar)
    {
    case 'i':
      inputFilename = (char *) strdup (optarg);
      break;

    case 'o':
      outputFilename = (char *) strdup (optarg);
      break;

    case 'x':
      xs = atoi (optarg);
      break;

    case 'y':
      ys = atoi (optarg);
      break;

    case 'w':
      xe = xs + atoi (optarg);
      break;

    case 'l':
      ye = ys + atoi (optarg);
      break;

    default:
      usage (argv[0], 0);
      break;
    }
    }

  // Make sure we have reasonable defaults
  if (xe == -1)
```

```
    xe = xs + 10;

  if (ye == -1)
    ye = ys + 10;

  // Open the input TIFF image
  if ((inputFilename == NULL) ||
      (input = TIFFOpen (inputFilename, "r")) == NULL)
    {
      fprintf (stderr, "Could not open incoming input %s\n", inputFilename);
      usage (argv[0], 42);
    }

  // Open the output TIFF
  if ((outputFilename == NULL) ||
      (output = TIFFOpen (outputFilename, "w")) == NULL)
    {
      fprintf (stderr, "Could not open outgoing input %s\n", outputFilename);
      usage (argv[0], 42);
    }

  // Find the width and height of the input
  TIFFGetField (input, TIFFTAG_IMAGEWIDTH, &width);
  TIFFGetField (input, TIFFTAG_IMAGELENGTH, &height);

  // Sanity check some of our arguements
  if (xe > width)
    {
      fprintf (stderr,
           "You choice of starting x position, or width, results in the en-
largement falling off the edge of the input image\n");
      usage (argv[0], 43);
    }

  if (ye > height)
    {
      fprintf (stderr,
           "You choice of starting y position, or length, results in the en-
largement falling off the end of the input image\n");
      usage (argv[0], 43);
    }

  printf ("Enlarging a %d by %d portion of the image\n", xe - xs, ye -
ys);

  ////////////////////////////////////////////////////////////////////////
  // Grab some memory
  if ((raster = (char *) malloc (sizeof (char) * width * height * 3)) == NULL)
    {
      fprintf (stderr, "Could not allocate enough memory for input raster\n");
      exit (42);
    }

  // todo: crap assumption about the data being 8 bps, 3 spp
  if ((enlarged = (char *) malloc (sizeof (char) * (xe - xs) * (ye - ys) * 3 *
                   121)) == NULL)
    {
      fprintf (stderr,
           "Could not allocate enough memory for enlarged raster\n");
      exit (42);
    }

  if ((rout = (char *) malloc (sizeof (char) * (width + EOFFSET) * height * 3))
      == NULL)
    {
      fprintf (stderr,
```

```
          "Could not allocate enough memory for output raster\n");
    exit (42);
  }

// todo: We need to copy tags from the input image to the output image
TIFFSetField (output, TIFFTAG_IMAGEWIDTH, width + EOFFSET);
TIFFSetField (output, TIFFTAG_IMAGELENGTH, height);
TIFFSetField (output, TIFFTAG_COMPRESSION, COMPRESSION_DEFLATE);
TIFFSetField (output, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);
TIFFSetField (output, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_RGB);
TIFFSetField (output, TIFFTAG_BITSPERSAMPLE, 8);
TIFFSetField (output, TIFFTAG_SAMPLESPERPIXEL, 3);
// todo: balance this off with having 8 kb per strip...
TIFFSetField (output, TIFFTAG_ROWSPERSTRIP, 100000);

//////////////////////////////////////////////////////////////////////////
// Read the input into the memory buffer
// todo: I couldn't use TIFFReadRGBAStrip here, because it gets confused
stripSize = TIFFStripSize (input);
roff = raster;
for (stripNumber = 0; stripNumber < TIFFNumberOfStrips (input);
     stripNumber++)
  {
    roff += TIFFReadEncodedStrip (input, stripNumber, roff, stripSize);
  }

//////////////////////////////////////////////////////////////////////////
// Actually do the enlargement of the portion of the image specified
offset = offset2 = 0;
for (y = ys; y < ye; y++)
  {
    for (yrep = 0; yrep < 10; yrep++)
  {
    for (x = xs; x < xe; x++)
      {
        offset = (x + (y * width)) * 3;
        for (xrep = 0; xrep < 10; xrep++)
      {
        memcpy (enlarged + offset2, raster + offset, 3);
        offset2 += 3;
      }

        // The white border to the left of the pixel
        memset (enlarged + offset2, 255, 3);
        offset2 += 3;
      }
  }

    // The white line at the bottom of these pixels
    memset (enlarged + offset2, 255, (ye - ys) * 3 * 11);
    offset2 += (ye - ys) * 3 * 11;
  }

//////////////////////////////////////////////////////////////////////////
// Now we assemble the two parts of the image together into a big output
// raster
memset (rout, 255, sizeof (char) * (width + EOFFSET) * height * 3);

// The original image
offset = 0;
offset2 = 0;
for (y = 0; y < height; y++)
  {
    memcpy (rout + offset2, raster + offset, width * 3);
    offset += width * 3;
    offset2 += (width + EOFFSET) * 3;
```

```
    }

  // Box the bit that was enlarged in the original image, can't use mem-
set here
  // Top line
  offset = (((width + EOFFSET) * (ys - 1)) + xs - 1) * 3;
  for (x = 0; x < xe - xs + 2; x++)
    {
      rout[offset++] = 255;
      rout[offset++] = 0;
      rout[offset++] = 0;
    }

  // Bottom line
  offset = (((width + EOFFSET) * ye) + xs - 1) * 3;
  for (x = 0; x < xe - xs + 2; x++)
    {
      rout[offset++] = 255;
      rout[offset++] = 0;
      rout[offset++] = 0;
    }

  // Vertical lines
  offset = (((width + EOFFSET) * ys) + xs - 1) * 3;
  for (y = 0; y < ye - ys + 1; y++)
    {
      rout[offset] = 255;
      rout[offset + 1] = 0;
      rout[offset + 2] = 0;

      rout[offset + ((ye - ys + 1) * 3)] = 255;
      rout[offset + ((ye - ys + 1) * 3) + 1] = 0;
      rout[offset + ((ye - ys + 1) * 3) + 2] = 0;

      offset += (width + EOFFSET) * 3;
    }

  ////////////////////////////////////////////////////////////////////////
  // Box the enlarged portion of the image

  ////////////////////////////////////////////////////////////////////////
  // Draw the two diagonal lines between the original and the enlarged
  //    this bit is based on the premis that y = mx + b
  //    and m = (y2 - y1) / (x2 - x1) and that geometry hasn't significantly
  //    changed since my high school days
  //
  //    if we assume that the first point is the origin, then the maths is
  //    even easier
  //
  //    which I think is probably a fairly safe set of assumptions at this
  //    stage...
  // todo
  //  printf("ye = %d, xe = %d\n", ye, xe);
  //  m = (ye - ys) / (xe - xs);
  //  for(x = 0; x < (xe - xs); x++){
  //    y = m * x;
  //
  //    printf("%d, %d (%f = %f)\n", x, y, m, ye / xe);
  //  }

  ////////////////////////////////////////////////////////////////////////
  // Copy the enlarged portion across

  offset = 0;
  offset2 = (((((height / 2) - ((ye - ys) * 11 / 2)) *
           (width + EOFFSET)) + width + 20) * 3);
```

```
  for (y = 0; y < (ye - ys) * 11; y++)
    {
      memcpy (rout + offset2, enlarged + offset, (ye - ys) * 11 * 3);
      offset += (ye - ys) * 11 * 3;
      offset2 += (width + EOFFSET) * 3;
    }

  ////////////////////////////////////////////////////////////////////////
  // Write the image buffer to the file
  if (TIFFWriteEncodedStrip (output, 0, rout,
                (width + EOFFSET) * height * 3 * sizeof (char)) == 0)
    {
      fprintf (stderr, "Could not write the output image\n");
      exit (42);
    }

  // Cleanup
  TIFFClose (input);
  TIFFClose (output);
  free (raster);
  free (enlarged);
}

void
usage (char *cmd, int exitamt)
{
  fprintf (stderr, "Bad command line arguements...\n\n");
  fprintf (stderr, "Usage: %s -i <inputfile> -o <outputfile> -x <start x> -
y <start y> -w <width> -l <length>\n", cmd);
  exit (exitamt);
}
```

Code: pixel.c

Example: Converting a color image to gray scale

In one of the previous chapters I also promised to provide the source code for the gray scale conversion examples. Here it is, there are two versions -- a broken algorithm (averaging color values), and a non-broken algorithm (using the NTSC recommended color coefficients).

A broken algorithm

```
#include <stdio.h>
#include <tiffio.h>
#include <unistd.h>
#include <string.h>

void usage (char *, int);

int
main (int argc, char *argv[])
{
  TIFF *input, *output;
  uint32 width, height, offset, offset2;
  tsize_t stripSize, stripNumber;
  unsigned long x, y;
  char *inputFilename = NULL, *outputFilename = NULL, *raster, *roff,
    optchar;
  unsigned char *rout;

  ////////////////////////////////////////////////////////////////////////
  // Parse the command line options
```

```
while ((optchar = getopt (argc, argv, "i:o:x:y:w:l:")) != -1)
  {
    switch (optchar)
  {
  case 'i':
    inputFilename = (char *) strdup (optarg);
    break;

  case 'o':
    outputFilename = (char *) strdup (optarg);
    break;

  default:
    usage (argv[0], 0);
    break;
  }
  }

// Open the input TIFF image
if ((inputFilename == NULL) ||
    (input = TIFFOpen (inputFilename, "r")) == NULL)
  {
    fprintf (stderr, "Could not open incoming input %s\n", inputFilename);
    usage (argv[0], 42);
  }

// Open the output TIFF
if ((outputFilename == NULL) ||
    (output = TIFFOpen (outputFilename, "w")) == NULL)
  {
    fprintf (stderr, "Could not open outgoing input %s\n", outputFilename);
    usage (argv[0], 42);
  }

// Find the width and height of the input
TIFFGetField (input, TIFFTAG_IMAGEWIDTH, &width);
TIFFGetField (input, TIFFTAG_IMAGELENGTH, &height);

////////////////////////////////////////////////////////////////////////////
// Grab some memory
if ((raster = (char *) malloc (sizeof (char) * width * height * 3)) == NULL)
  {
    fprintf (stderr, "Could not allocate enough memory for input raster\n");
    exit (42);
  }

if ((rout = (unsigned char *) malloc (sizeof (char) * width * height))
    == NULL)
  {
    fprintf (stderr,
         "Could not allocate enough memory for enlarged raster\n");
    exit (42);
  }

// todo: We need to copy tags from the input image to the output image
TIFFSetField (output, TIFFTAG_IMAGEWIDTH, width);
TIFFSetField (output, TIFFTAG_IMAGELENGTH, height);
TIFFSetField (output, TIFFTAG_COMPRESSION, COMPRESSION_DEFLATE);
TIFFSetField (output, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);
TIFFSetField (output, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_MINISBLACK);
TIFFSetField (output, TIFFTAG_BITSPERSAMPLE, 8);
TIFFSetField (output, TIFFTAG_SAMPLESPERPIXEL, 1);
// todo: balance this off with having 8 kb per strip...
TIFFSetField (output, TIFFTAG_ROWSPERSTRIP, 100000);

////////////////////////////////////////////////////////////////////////////
```

```
  // Read the input into the memory buffer
  // todo: I couldn't use TIFFReadRGBAStrip here, because it gets confused
  stripSize = TIFFStripSize (input);
  roff = raster;
  for (stripNumber = 0; stripNumber < TIFFNumberOfStrips (input);
       stripNumber++)
    {
      roff += TIFFReadEncodedStrip (input, stripNumber, roff, stripSize);
    }

  ////////////////////////////////////////////////////////////////////////
  // Convert to grayscale
  offset2 = 0;
  for(offset = 0; offset < width * height * 3; offset += 3){
    rout[offset2++] = ((unsigned char) (raster[offset]) * 0.333 +
               (unsigned char) (raster[offset + 1]) * 0.333 +
               (unsigned char) (raster[offset + 2]) * 0.333);
  }

  ////////////////////////////////////////////////////////////////////////
  // Write the image buffer to the file
  if (TIFFWriteEncodedStrip (output, 0, rout,
                 width * height * sizeof (char)) == 0)
    {
      fprintf (stderr, "Could not write the output image\n");
      exit (42);
    }

  // Cleanup
  TIFFClose (input);
  TIFFClose (output);
  free (raster);
  free (rout);
}

void
usage (char *cmd, int exitamt)
{
  fprintf (stderr, "Bad command line arguements...\n\n");
  fprintf (stderr, "Usage: %s -i <inputfile> -o <outputfile>\n", cmd);
  exit (exitamt);
}
```

Code: gray-wrong.c

Which produces:

Figure 3-11. An average of the color values for each pixel

A sensible algorithm

```
#include <stdio.h>
#include <tiffio.h>
#include <unistd.h>
#include <string.h>

void usage (char *, int);

int
main (int argc, char *argv[])
{
  TIFF *input, *output;
  uint32 width, height, offset, offset2;
  tsize_t stripSize, stripNumber;
  unsigned long x, y;
  char *inputFilename = NULL, *outputFilename = NULL, *raster, *roff,
    optchar;
  unsigned char *rout;

  ////////////////////////////////////////////////////////////////////////
  // Parse the command line options
  while ((optchar = getopt (argc, argv, "i:o:x:y:w:l:")) != -1)
    {
      switch (optchar)
    {
    case 'i':
      inputFilename = (char *) strdup (optarg);
      break;

    case 'o':
      outputFilename = (char *) strdup (optarg);
      break;
```

```
  default:
    usage (argv[0], 0);
    break;
  }
  }

// Open the input TIFF image
if ((inputFilename == NULL) ||
    (input = TIFFOpen (inputFilename, "r")) == NULL)
  {
    fprintf (stderr, "Could not open incoming input %s\n", inputFilename);
    usage (argv[0], 42);
  }

// Open the output TIFF
if ((outputFilename == NULL) ||
    (output = TIFFOpen (outputFilename, "w")) == NULL)
  {
    fprintf (stderr, "Could not open outgoing input %s\n", outputFilename);
    usage (argv[0], 42);
  }

// Find the width and height of the input
TIFFGetField (input, TIFFTAG_IMAGEWIDTH, &width);
TIFFGetField (input, TIFFTAG_IMAGELENGTH, &height);

////////////////////////////////////////////////////////////////////////
// Grab some memory
if ((raster = (char *) malloc (sizeof (char) * width * height * 3)) == NULL)
  {
    fprintf (stderr, "Could not allocate enough memory for input raster\n");
    exit (42);
  }

if ((rout = (unsigned char *) malloc (sizeof (char) * width * height))
    == NULL)
  {
    fprintf (stderr,
         "Could not allocate enough memory for enlarged raster\n");
    exit (42);
  }

// todo: We need to copy tags from the input image to the output image
TIFFSetField (output, TIFFTAG_IMAGEWIDTH, width);
TIFFSetField (output, TIFFTAG_IMAGELENGTH, height);
TIFFSetField (output, TIFFTAG_COMPRESSION, COMPRESSION_DEFLATE);
TIFFSetField (output, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);
TIFFSetField (output, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_MINISBLACK);
TIFFSetField (output, TIFFTAG_BITSPERSAMPLE, 8);
TIFFSetField (output, TIFFTAG_SAMPLESPERPIXEL, 1);
// todo: balance this off with having 8 kb per strip...
TIFFSetField (output, TIFFTAG_ROWSPERSTRIP, 100000);

////////////////////////////////////////////////////////////////////////
// Read the input into the memory buffer
// todo: I couldn't use TIFFReadRGBAStrip here, because it gets confused
stripSize = TIFFStripSize (input);
roff = raster;
for (stripNumber = 0; stripNumber < TIFFNumberOfStrips (input);
     stripNumber++)
  {
    roff += TIFFReadEncodedStrip (input, stripNumber, roff, stripSize);
  }

////////////////////////////////////////////////////////////////////////
// Convert to grayscale
```

```
  offset2 = 0;
  for(offset = 0; offset < width * height * 3; offset += 3){
    rout[offset2++] = ((unsigned char) (raster[offset]) * 0.299 +
               (unsigned char) (raster[offset + 1]) * 0.587 +
               (unsigned char) (raster[offset + 2]) * 0.114);
  }

  ////////////////////////////////////////////////////////////////////////
  // Write the image buffer to the file
  if (TIFFWriteEncodedStrip (output, 0, rout,
                width * height * sizeof (char)) == 0)
    {
      fprintf (stderr, "Could not write the output image\n");
      exit (42);
    }

  // Cleanup
  TIFFClose (input);
  TIFFClose (output);
  free (raster);
  free (rout);
}

void
usage (char *cmd, int exitamt)
{
  fprintf (stderr, "Bad command line arguements...\n\n");
  fprintf (stderr, "Usage: %s -i <inputfile> -o <outputfile>\n", cmd);
  exit (exitamt);
}
```

Code: gray-good.c

Which gives us:

Figure 3-12. A correct conversion to gray scale

Now, just to justify that the second algorithm is better, have a look at the shadows on the hill and the trees in the background. The contrast and darkness of the second picture is might better than the averaging algorithm...

Example: Repeated compression

Earlier in this chapter I promised that I would include the code for the repeated compression loss example, so here it is. The example is made up of a simple (and quite ugly) shell script, and a c program to actually compress the images...

```
#!/bin/bash

count=1
cp $1 $1-0.tif

while [ $count -lt 200 ]
do
  ./read $1-$(( $count -1 )).tif $1-$count.tif
  count=$(( $count + 1 ))
done
```

Code: recompress.sh

```
#include <stdio.h>
#include <tiffio.h>

int main(int argc, char *argv[]){
  TIFF *image, *output;
  uint16 photo, bps, spp, fillorder;
  uint32 width, height, *raster;
  tsize_t stripSize;
  unsigned long imagesize, c, d, e;
  char *raster2;

  // Open the TIFF image
  if((image = TIFFOpen(argv[1], "r")) == NULL){
    fprintf(stderr, "Could not open incoming image\n");
    exit(42);
  }

  // Open the output image
  if((output = TIFFOpen(argv[2], "w")) == NULL){
    fprintf(stderr, "Could not open outgoing image\n");
    exit(42);
  }

  // Find the width and height of the image
  TIFFGetField(image, TIFFTAG_IMAGEWIDTH, &width);
  TIFFGetField(image, TIFFTAG_IMAGELENGTH, &height);
  imagesize = height * width + 1;

  if((raster = (uint32 *) malloc(sizeof(uint32) * imagesize)) == NULL){
    fprintf(stderr, "Could not allocate enough memory\n");
    exit(42);
  }

  if((raster2 = (char *) malloc(sizeof(char) * imagesize * 3)) == NULL){
    fprintf(stderr, "Could not allocate enough memory\n");
    exit(42);
  }

  // Read the image into the memory buffer
  if(TIFFReadRGBAStrip(image, 0, raster) == 0){
```

```
    fprintf(stderr, "Could not read image\n");
    exit(42);
  }

  d = 0;
  for(e = height - 1; e != -1; e--){
    for(c = 0; c < width; c++){
      raster2[d++] = TIFFGetR(raster[e * width + c]);
      raster2[d++] = TIFFGetG(raster[e * width + c]);
      raster2[d++] = TIFFGetB(raster[e * width + c]);
    }
  }

  // Recompress it straight away -- set the tags we require
  TIFFSetField(output, TIFFTAG_IMAGEWIDTH, width);
  TIFFSetField(output, TIFFTAG_IMAGELENGTH, height);
  TIFFSetField(output, TIFFTAG_COMPRESSION, COMPRESSION_JPEG);
  TIFFSetField(output, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);
  TIFFSetField(output, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_RGB);
  TIFFSetField(output, TIFFTAG_BITSPERSAMPLE, 8);
  TIFFSetField(output, TIFFTAG_SAMPLESPERPIXEL, 3);
  TIFFSetField(output, TIFFTAG_JPEGQUALITY, 25);

  // Actually write the image
  if(TIFFWriteEncodedStrip(output, 0, raster2, imagesize * 3) == 0){
    fprintf(stderr, "Could not read image\n");
    exit(42);
  }

  TIFFClose(output);
  TIFFClose(image);
}
```

Code: recompress.c

Conclusion

In this chapter we've discussed the TIFF file format, how to program with libtiff for black and white, gray scale and color images. We haven't been through an exhaustive discussion of the possible settings, or all the calls available in libtiff, but you should be ready to start exploring further.

Further reading

- http://www.libtiff.org: The libtiff website is a good place to download the libtiff source. It is also quite likely there is a binary package for your chosen operating system.
- http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf: If all else fails, then the Adobe TIFF Specification can be useful.
- http://gopher.std.com/homepages/jimf/xloadimage.html: The xloadimage web page might be of interest.
- http://www.ee.cooper.edu/courses/course_pages/past_courses/EE458/TIFF/index.html: The Cooper Union for the Advancement of Science and Art has some notes from a previous course dealing with libtiff online.
- http://partners.adobe.com/asn/developer/graphics/graphics.html: The Adobe Graphics technical notes page is quite useful.

- http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf: The current version of the TIFF specification can be found here.
- http://home.earthlink.net/~ritter/tiff/: The unofficial TIFF site contains some useful resources.
- http://www.inforamp.net/~poynton/notes/colour_and_gamma/ColorFAQ.html: a FAQ document discussion of converting to gray scale

Portions first published by IBM DeveloperWorks

The basis of this chapter was two articles, first published by IBM DeveloperWorks at http://www.ibm.com/developerworks/ in April and May 2002. This material has been expanded in these chapters from the original 5,000 words.

Notes

1. Please note that there are binary packages for libtiff for most operating systems, so you might be best off installing one of those.
2. From page 13 onwards...
3. You need not worry about this, because libtiff worries about all of this for you.
4. http://www.stillhq.com/cgi-bin/getpage?area=panda<page=index.htm to be exact.
5. Refer to page 16 of the TIFF specification, version 6.

Chapter 4. GIF

"Between 1987 and 1994, GIF (Graphics Interchange Format) peacefully became the most popular file format for archiving and exchanging computer images. At the end of December 1994, CompuServe Inc. and Unisys Corporation announced to the public that developers would have to pay a license fee in order to continue to use technology patented by Unisys in certain categories of software supporting the GIF format. These first statements caused immediate reactions and some confusion. As a longer term consequence, it appears likely that GIF will be replaced and extended by new file formats, but not so before the expiration of the patent which caused so much debate." -- http://www.cloanto.com/users/mcb/19950127giflzw.html (The GIF Controversy: A Software Developer's Perspective)

GIF was once the image format of choice for the Internet, in web browsers at least. This is not nearly as much the case today. The main reason for this is because Unisys has only relatively recently started enforcing the Patent they hold on LZW compression.

The disappearance of GIF from the scene is not a major loss, especially as the format was not particularly extensible.

The GIF on disc format

The GIF file format is much simpler than that used for TIFF files. Pay attention to this description though, because it will help explain PNG images when we get to them in a later chapter.

Data streams?

It should be noted that the GIF specification doesn't speak of files as such. Instead it uses the term "Data Stream", which is a concept which embraces files, as well as in memory buffers, and other interesting forms of data storage. It doesn't really make any difference to the discussion of the format here though.

The data stream

The outer body of a GIF image is called a data stream, for reasons described in the last few paragraphs. This data stream can be thought of as a file for the purposes of this discussion. The data stream is composed of header, a section called the logical screen, the image data, and a trailer. These elements are described individually below.

The header

Table 4-1. GIF on disc: the data stream header

Bytes	Description
0 - 2	A magic number identifying that this is a GIF image. This should be the ASCII characters "GIF".
3 - 5	The version number of the GIF specification this image uses. See below for a list of possible version numbers.

There are two main version numbers which are in common use for GIF images. These are:

- 87a (released in May 1987)
- 89a (released in July 1989)

Version numbers

The GIF specification makes it quite clear that the image generator should use the lowest version number which matches the functionality required to correctly decode the image. This makes the generation of images a little more complex, as the generator needs to know the specification version number of each feature, but it ensures the maximum possible level of backwards compatability, without sacraficing accuracy of decoding of the image. To quote the GIF specification:

"The version number in the Header of a Data Stream is intended to identify the minimum set of capabilities required of a decoder in order to fully process the Data Stream. An encoder should use the earliest possible version number that includes all the blocks used in the Data Stream. ... The encoder should make every attempt to use the earliest version number covering all the blocks in the Data Stream; the unnecessary use of later version numbers will hinder processing by some decoders."

Logical Screen

The logical screen is the area on which the image will be painted. This block defines the characteristics of this virtual screen:

Table 4-2. GIF on disc: the logical screen block

Bytes	Description
0 - 1	Width
2 - 3	Height
4	Packed fields. See below for a description.
5	Background color index. If a global color table exists, then this indicates the index into that table to use for pixels which are transparent. For instance, the author of the GIF image might have specified a certain shade of blue as representing a transparent pixel. If the global color table is not supported by this image, this value should be zero.
6	Pixel aspect ratio -- of the original image. The default value is zero, but if desired, this value can be used to calculate the original image aspect ratio using the formula *aspect ratio = (pixel aspect ratio + 15) / 64*

The packed fields in byte 4 of the logical screen block are as follows:

Table 4-3. GIF on disc: fields packed into byte 4 of the logical screen block

Bits	Description

Bits	Description
0	Global color table flag -- this indicates if there is a global color table in the image file (0 for no, 1 for yes).
1 - 3	The color resolution (bits per primary color minus one), of the original image before it was converted to GIF. Therefore, for a 24 bit color image which is converted to GIF, this value would be seven.
4	Sort flag -- indicates if the global color table is sorted (0 for no, 1 for yes).
5 - 7	Size of the global color table. Please note that this is not a simple number representing the size of the table, but must be converted using a formula described below.

We have now mentioned the global color table, but haven't defined what it actually means. Don't panic, we'll get there in just a second.

Formula for global color table size

To determine the size of the of the global color table, simply apply this formula: *size of global color table = 2 ^ (size in table + 1)*. The maximum size of the global color table is therefore *2 ^ (7 + 1)*, or 255 items.

There will actually be three times that number of color bytes in the global color table, because each color is 24 bit per pixel.

Global color table

If the logical screen block of the image specified that a global color table is present in the image, then this is the next part of the GIF data stream. The global color table is a palette -- in other words, it maps the values actually stored in the image data to the 24 bit values which are the colors.

The format of the global color table is very simple, it is simply the red, green and blue values stored without any header or footer.

Table 4-4. GIF on disc: the global color table

Bytes	Description
0	Colour 1 red entry
1	Colour 1 green entry
2	Colour 1 blue entry
...	
	Colour x red entry
	Colour x green entry
	Colour x blue entry

Data

Next comes the images that are stored in the GIF data stream. GIF data streams are valid, even if there are no images stored within the data stream.

Image descriptors

"Each image in the Data Stream is composed of an Image Descriptor, an optional Local Color Table, and the image data. Each image must fit within the boundaries of the Logical Screen, as defined in the Logical Screen Descriptor." -- GIF 89a specification

The image descriptor is stored at the start of each image within the data stream. The image description stores information which is specific to that single image, unlike the logical screen section, which has a global scope over all images in the data stream. The image descriptor has the following layout:

Table 4-5. GIF on disc: the image descriptor

Bytes	Description
0	0x2C -- marks the start of this image
1 - 2	Image left position (on the logical screen)
3 - 4	Image top postion (on the logical screen)
5 - 6	Image width
7 - 8	Image height
9	Packed fields. See below for a description.

The packed field byte stores the following information:

Table 4-6. GIF on disc: fields packed into byte 9 of the image descriptor

Bits	Description
0	Logical color table flag (0 for no, 1 for yes). If present, this local color table with override the global color table. If there is no global color table in this data stream, then a local color table *must* be present.
1	Interlace flag (0 for no, 1 for yes). See the interlaced images sections below for more information.
2	Sort flag (0 for no, 1 for yes). This indicates if the local color table is sorted. "Indicates whether the Local Color Table is sorted. If the flag is set, the Local Color Table is sorted, in order of decreasing importance. Typically, the order would be decreasing frequency, with most frequent color first. This assists a decoder, with fewer available colors, in choosing the best subset of colors; the decoder may use an initial segment of the table to render the graphic." -- GIF 89a specification
3 - 4	*Reserved for future use*

Bits	Description
5 - 8	Size of the local color table. This value obeys the same rules as the size field for the global color table.

Local color table

The format rules for the local color table are identical to those for the global color table.

Image data

The raster information for the image is stored in a series of image data blocks, each block having a maximum size of 255 bytes, which works well on machines with very little memory. Each byte is an index into whatever color table entry is active for this image (remembering that the local color table overrides the global color table). Each of these image data blocks is LZW compressed as part of one big buffer. The format of the image data blocks is:

Table 4-7. GIF on disc: the image data block

Bytes	Description
0	LZW minimum code size (outside the scope of this tutorial)
...	Image data subblocks

Image data sub blocks

The format of these image data subblocks is:

Table 4-8. GIF on disc: the image data subblocks

Bytes	Description
0	Block size, in bytes, not including this byte. The maxiumum size if 255 bytes.
1 - blocksize	Image data (indices into the relevant color table)

Terminating subblock

The last block in the chain of image data subblocks will be a terminating subblock. It will have the format:

Table 4-9. GIF on disc: the terminating image subblock

Bytes	Description
0	0x0 (i.e. a length of zero)

Special purpose blocks

The other thing which can be stored in the data area of the data stream of a GIF file is special purpose blocks. These are outside the scope of this tutorial, and wont be discussed here.

Trailer

This portion of the GIF image format specifies the end of the data stream. It is really quite simple and doesn't justify a table. It is simply a byte with the value 0x3B.

Interlaced images

GIF supports the interlacing of images, which is where are the image is loaded, progressively better representations of the image are displayed. Discussing the inner workings of this within the file format is out of the scope of this tutorial however.

Conclusion

In this chapter we have learnt how the GIF format is laid out internally. This will be useful when it is time to dicsuss the PNG format...

Chapter 5. PNG

"A Turbo-Studly Image Format with Lossless Compression (Not Related to Papua New Guinea, the Pawnee National Grassland, the Professional Numismatists Guild or the "Pack 'N' Go" format)" -- libpng.org

PNG [1] is my favorite image format after TIFF. It is well conceived, well implemented, and very powerful. It also has some interesting features, mainly aimed at Internet use, which TIFF lacks. It is also usable in modern web browsers [2], unlike TIFF. PNG is also much simpler in many respects than TIFF.

This chapter will focus on the libpng library. This isn't really a limitation, as the people behind libpng are also the people who write the PNG specification, so you're pretty safe in assuming that if it's useful and PNG does it, then libpng implements it.

Introduction

PNG happened because of the patent problems with the GIF format described in an earlier chapter of this tutorial. In the words of the official history of PNG:

"... Unisys in 1993 began aggressively pursuing commercial vendors of software-only LZW implementations. CompuServe seems to have been its primary target at first, culminating in an agreement--quietly announced on 28 December 1994, right in the middle of the Christmas holidays--to begin collecting royalties from authors of GIF-supporting software. The spit hit the fan on the Internet the following week; what was then the comp.graphics newsgroup went nuts, to use a technical term. As is the way of Usenet, much ire was directed at CompuServe for making the announcement, and then at Unisys once the details became a little clearer; but mixed in with the noise was the genesis of an informal Internet working group led by Thomas Boutell [2]. Its purpose was not only to design a replacement for the GIF format, but a successor to it: better, smaller, more extensible, and FREE." -- http://www.libpng.org/pub/png/pnghist.html

The PNG has gone from being a simple specification with limitations such as only supporting 8 bit images first announced on comp.graphics, comp.compression and comp.infosystems.www.providers, into a mature and extremely extensible image format.

Various versions of the PNG specification have been published as RFC-2083 and as a W3C Recommendation, which means that supporting the format is a lot easier because it is well defined.

A bunch of the examples in this chapter are based on the pngtools code I wrote a while ago, which I have updated as part of writing this chapter.

> **Refer to the TIFF chapter**
>
> Much of the discussion in this chapter will make a whole bunch more sense if you've read the TIFF chapter. Off you go, I'll wait for you to come back...

Installation

You can get the latest libpng code from http://www.libpng.org.

Unix

libpng doesn't have a configure script. You'll need to follow the steps below to get it to compile:

- cp scripts/makefile.linux Makefile *(You'll need to copy the Makefile that matches your architecture and operating system)*
- make
- make install

win32

There are Makefiles for the win32 platform as well. I haven't verified how well they work however.

The PNG on disc format

It is useful to introduce the PNG file format as one of the first parts of this chapter, as it helps to see the advantages and disadvantages of the PNG format compared with the other formats that we will discuss during this tutorial.

Byte order

If you recall the TIFF format, multi byte values can either be little endian or big endian. This means that the libtiff library must perform byte swapping every time you read a value which takes more than one byte (which is most of the time). libpng doesn't have these problems, as all multi byte values are in network byte order (big endian).

File header

All PNG files start with a magic number (as have all the other formats which we have discussed in this tutorial). It looks something like:

Table 5-1. PNG on disc: the file header

Bytes	Description
0	Decimal 137 *No ASCII equivalent*
1	Decimal 80 *P*
2	Decimal 78 *N*
3	Decimal 71 *G*
4	Decimal 13 *Carriage return (\r)*
5	Decimal 10 *Newline (\n)*
6	Decimal 26 *No ASCII equivalent*
7	Decimal 10 *Newline (\n)*

Chunk format

Chunks map as a concept to the blocks that the GIF format used. The generic format for a PNG chunk is:

Table 5-2. PNG on disc: chunk format

Bytes	Description

Bytes	Description
0 - 3	Length of the data portion of this chunk (does not include the length field, the type field, or the CRC). The value is unsigned, and zero is a valid length. The maximum value is 2^32 - 1
4 - 7	Chunk type. Chunk types normally fall into the upper and lower case ASCII ranges. Software should treat this field as an unsigned value however. Chunk type naming conventions are discussed later in this chapter.
...	Chunk data.
(4 bytes at end of chunk)	Cyclic Redundancy Check (CRC). The CRC covers the type and data fields, but not the length field. The CRC is always present, even if we have no data.

Chunk naming conventions

The case used in chunk names has special significance. The table below describes the chunk name characters.

Table 5-3. PNG on disc: chunk naming conventions

Bytes	Description
0	Uppercase: critical. Lowercase: optional (known as ancillary). If a decoder encounters a critical chunk it does not understand, it will warn the user about possible incorrect decoding of the image.
1	Uppercase: public. Lowercase: private. A public chunk is either part of the PNG specification, or is published in the list of PNG public chunk types. Private chunks are developer specific, and should be ignored if not understood.
2	Uppercase: PNG 1.2 compliant. Lowercase: reserved for future use. For the time being, you should never see a third character which is lower case.

Bytes	Description
3	Uppercase: unsafe to copy. Lowercase: safe to copy. If the decoder doesn't understand the chunk, then should it be copied into new images? A chunk would be unsafe to copy if it relied on the image data which the decoder might have changed (including some critical chunks). An example of a safe to copy chunk might be a chunk which includes a MP3 sound recording from the time in which the image was encoded. An example of an unsafe to copy chunk might be an MD5 hash on the image data.

CRC algorithm

The exact workings of the CRC algorithm are outside the scope of this tutorial...

The IHDR chunk

This is the image header chunk. This chunk *must be first* in the PNG file. This chunk contains the following fields:

Table 5-4. PNG on disc: IHDR chunk fields

Bytes	Description
0 - 3	Width
4 - 7	Height
8	Bit depth (not of bits per sample or palette index)
9	Color type
10	Compression method (method zero is the only method currently defined)
11	Filter method (method zero is the only method currently defined)
12	Interlace method. The current options are: zero (no interlace) and one (adam7)

There are various values which are valid for the color type field. Each of these types has only certain byte depths allowed.

Table 5-5. PNG on disc: IHDR chunk color types

Color type: allowed bit depths	Comments
0: 1, 2, 4, 8, 16	Each pixel is a gray scale sample
1: not applicable	Each pixel value is an index into a palette table
2: 8, 16	Each pixel has an red, green, and blue value

Color type: allowed bit depths	Comments
3: 1, 2, 4, 8	Each pixel is a palette index, and a PLTE chunk must appear in the PNG file
4: 8, 16	Each pixel is a gray scale sample, followed by an alpha channel
6: 8, 16	Each pixel is an red, green, blue, alpha set

For all of these types, the sample depth is the same as the bit depth, except for type three, where the sample depth is always eight bits.

Table 5-6. PNG on disc: IHDR chunk

Required?	Critical
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

Filtering algorithm

The PNG specification states on page 16 of version 1.2: "Filter method is a single-byte integer that indicates the preprocessing method applied to the image data before compression. At present, only filter method 0 (adaptive filtering with five basic filter types) is defined. As with the compression method field, decoders must check this byte and report an error if it holds an unrec-ognized code. See Filter Algorithms (Chapter 6) for details."

The PLTE chunk

This chunks stores the palette information for the image. A palette contains 1 through 256 entries, and each entry consists of a red byte, green byte, and a blue byte. You don't need to have 256 entries in the palette -- the size of the palette is determined by dividing the chunk length by three. A chunk length which is not divisible by three is an error condition.

Table 5-7. PNG on disc: PLTE chunk

Required?	Critical
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The IDAT chunk

The IDAT chunk contains the actual image data. This data will be the output of the selected compression method, and will need to be uncompressed before it is used.

Table 5-8. PNG on disc: IDAT chunk

Required?	Critical

Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The IEND chunk

The IEND chunk must be the last chunk in the PNG file. It marks the end of the PNG file. The length of the data inside this chunk is zero.

Table 5-9. PNG on disc: IEND chunk

Required?	Critical
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

Ancillary chunks

Each ancillary chunk is only given a brief description below. Refer to the PNG specification for more information...

The tRNS chunk

Transparency information.

Table 5-10. PNG on disc: tRNS chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The gAMA chunk

Image gamma (pixel intensity) information.

Table 5-11. PNG on disc: gAMA chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The cHRM chunk

Chromacity information.

Table 5-12. PNG on disc: cHRM chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The sRGB chunk

The image data conforms to the ICC RGB color space.

Table 5-13. PNG on disc: sRGB chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The iCCP chunk

Contains an embedded ICC profile.

Table 5-14. PNG on disc: iCCP chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The iTXt chunk

International textual data.

Table 5-15. PNG on disc: iTXt chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Safe to copy

The tEXt chunk

Uncompressed text.

Table 5-16. PNG on disc: tEXt chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Safe to copy

The zTXt chunk

Flate compressed text.

Table 5-17. PNG on disc: zTXt chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Safe to copy

The bKGD chunk

Background color information.

Table 5-18. PNG on disc: bKGD chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The pHYs chunk

Pixel size information.

Table 5-19. PNG on disc: pHYs chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Safe to copy

The sBIT chunk

Stores the original number of significant bits.

Table 5-20. PNG on disc: sBIT chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The sPLT chunk

Suggested palette.

Table 5-21. PNG on disc: sPLT chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The hIST chunk

Palette histogram.

Table 5-22. PNG on disc: hIST chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

The tIME chunk

Time of last image modification.

Table 5-23. PNG on disc: tIME chunk

Required?	Ancillary
Scope?	Public
In reserved name space?	PNG 1.2 compliant
Safe to copy?	Unsafe to copy

PNG should be easier than TIFF

Unlike the TIFF format, which allows the creator of the image files to specify many different image options, PNG imposes rules such as that the images are big endian. Whilst this is less flexible for the creators of PNG files compared with TIFF, it does make it much easier to reliably decode a PNG image.

Opening a PNG file

The first step to learning how to use libpng is probably to understand how to open a PNG file and get some data out of it. Below is a minimal example of how to open a PNG file, not including actually reading the image data...

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <png.h>
#include <unistd.h>

int main(int argc, char *argv[]){
  FILE *image;
  unsigned long width, height;
  int bitdepth, colourtype;
  png_uint_32 i, j;
  png_structp png;
  png_infop info;
  unsigned char sig[8];

  // Open the file
  if ((image = fopen (argv[1], "rb")) == NULL){
    fprintf(stderr, "Could not open the specified PNG file.");
    exit(0);
  }

  // Check that it really is a PNG file
  fread(sig, 1, 8, image);
  if(!png_check_sig(sig, 8)){
    printf("This file is not a valid PNG file\n");
    fclose(image);
    return;
  }

  // Start decompressing
  if((png = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
                    NULL, NULL)) == NULL){
    fprintf(stderr, "Could not create a PNG read structure (out of memory?)");
    exit(0);
  }

  if((info = png_create_info_struct(png)) == NULL){
    fprintf(stderr, "Could not create PNG info structure (out of memory?)");
    exit(0);
  }

  // If pnginfo_error did not exit, we would have to call
  // png_destroy_read_struct
  if(setjmp(png_jmpbuf(png))){
    fprintf(stderr, "Could not set PNG jump value");
    exit(0);
  }

  // Get ready for IO and tell the API we have already read the image signature
  png_init_io(png, image);
  png_set_sig_bytes(png, 8);
```

```
  png_read_info(png, info);
  png_get_IHDR(png, info, &width, &height, &bitdepth, &colourtype, NULL,
          NULL, NULL);

  // This cleans things up for us in the PNG library
  fclose(image);
  png_destroy_read_struct(&png, &info, NULL);
}
```

Code: read-infrastructure.c

In this example, we do everything we need to open a PNG image, without actually reading the image data. The steps to getting to the image data are:

- Open the file ready for reading. For this we just use the c standard library's **FILE ***, unlike the libtiff examples, in which we used a **TIFF *** [3]
- Another thing we need to do which libtiff does for us and libpng doesn't is check that the file really is a PNG file. We do this with the **png_check_sig**() call. This expects the first 8 bytes of the file to be handed to it, which is what the **fread**() function call gets for us.
- We then start decompressing the image
- The info struct gives us access to important information about the image
- Then we read the IHDR chunk (see above for a description of this chunk)

Example: pnginfo

The **pnginfo** command implemented here is modeled on the **tiffinfo**, which was discussed in the TIFF chapter earlier in this tutorial. In fact, the output text is written to be as close to the **tiffinfo** command as possible.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <png.h>
#include <unistd.h>

void pnginfo_displayfile (char *, int, int, int);
void pnginfo_error (char *);
void *pnginfo_xmalloc (size_t);
void usage (void);

#define pnginfo_true 1
#define pnginfo_false 0

int
main (int argc, char *argv[])
{
  int i, optchar, extractBitmap = pnginfo_false, displayBitmap =
    pnginfo_false, tiffnames = pnginfo_false;

  // Initialise the argument that filenames start at
  i = 1;

  // Use getopt to determine what we have been asked to do
  while ((optchar = getopt (argc, argv, "tDd")) != -1)
    {
      switch (optchar)
    {
    case 't':
      tiffnames = pnginfo_true;
      i++;
```

```
      break;

    case 'd':
      displayBitmap = pnginfo_true;
      extractBitmap = pnginfo_true;
      i++;
      break;

    case 'D':
      extractBitmap = pnginfo_true;
      i++;
      break;

    case '?':
    default:
      usage ();
      break;
    }
    }

  // Determine if we were given a filename on the command line
  if (argc < 2)
    usage ();

  // For each filename that we have:
  for (; i < argc; i++)
    pnginfo_displayfile (argv[i], extractBitmap, displayBitmap, tiffnames);
}

void
pnginfo_displayfile (char *filename, int extractBitmap, int displayBitmap, int tiffname
{
  FILE *image;
  unsigned long imageBufSize, width, height, runlen;
  unsigned char signature;
  int bitdepth, colourtype;
  png_uint_32 i, j, rowbytes;
  png_structp png;
  png_infop info;
  unsigned char sig[8];
  png_bytepp row_pointers = NULL;
  char *bitmap;

  printf ("%s%s...\n", filename, \
      tiffnames == pnginfo_true? " (tiffinfo compatible labels)" : "");

  // Open the file
  if ((image = fopen (filename, "rb")) == NULL)
    pnginfo_error ("Could not open the specified PNG file.");

  // Check that it really is a PNG file
  fread (sig, 1, 8, image);
  if (!png_check_sig (sig, 8))
    {
      printf ("  This file is not a valid PNG file\n");
      fclose (image);
      return;
    }

  // Start decompressing
  if ((png = png_create_read_struct (PNG_LIBPNG_VER_STRING, NULL,
                    NULL, NULL)) == NULL)
    pnginfo_error ("Could not create a PNG read structure (out of memory?)");

  if ((info = png_create_info_struct (png)) == NULL)
    pnginfo_error ("Could not create PNG info structure (out of memory?)");
```

```
// If pnginfo_error did not exit, we would have to call
// png_destroy_read_struct

if (setjmp (png_jmpbuf (png)))
  pnginfo_error ("Could not set PNG jump value");

// Get ready for IO and tell the API we have already read the image signature
png_init_io (png, image);
png_set_sig_bytes (png, 8);
png_read_info (png, info);
png_get_IHDR (png, info, &width, &height, &bitdepth, &colourtype, NULL,
      NULL, NULL);

////////////////////////////////////////////////////////////////////////
// Start displaying information
////////////////////////////////////////////////////////////////////////

printf ("  Image Width: %d Image Length: %d\n", width, height);
if(tiffnames == pnginfo_true){
  printf ("  Bits/Sample: %d\n", bitdepth);
  printf ("  Samples/Pixel: %d\n", info->channels);
  printf ("  Pixel Depth: %d\n", info->pixel_depth);    // Does this add value?
}
else{
  printf ("  Bitdepth (Bits/Sample): %d\n", bitdepth);
  printf ("  Channels (Samples/Pixel): %d\n", info->channels);
  printf ("  Pixel depth (Pixel Depth): %d\n", info->pixel_depth);    // Does this ad
}

// Photometric interp packs a lot of information
printf ("  Colour Type (Photometric Interpretation): ");

switch (colourtype)
  {
  case PNG_COLOR_TYPE_GRAY:
    printf ("GRAYSCALE ");
    break;

  case PNG_COLOR_TYPE_PALETTE:
    printf ("PALETTED COLOUR ");
    if (info->num_trans > 0)
  printf ("with alpha ");
    printf ("(%d colours, %d transparent) ",
        info->num_palette, info->num_trans);
    break;

  case PNG_COLOR_TYPE_RGB:
    printf ("RGB ");
    break;

  case PNG_COLOR_TYPE_RGB_ALPHA:
    printf ("RGB with alpha channel ");
    break;

  case PNG_COLOR_TYPE_GRAY_ALPHA:
    printf ("GRAYSCALE with alpha channel ");
    break;

  default:
    printf ("Unknown photometric interpretation!");
    break;
  }
printf ("\n");

printf ("  Image filter: ");
```

```
switch (info->filter_type)
  {
  case PNG_FILTER_TYPE_BASE:
    printf ("Single row per byte filter ");
    break;

  case PNG_INTRAPIXEL_DIFFERENCING:
    printf ("Intrapixel differencing (MNG only) ");
    break;

  default:
    printf ("Unknown filter! ");
    break;
  }
printf ("\n");

printf ("  Interlacing: ");
switch (info->interlace_type)
  {
  case PNG_INTERLACE_NONE:
    printf ("No interlacing ");
    break;

  case PNG_INTERLACE_ADAM7:
    printf ("Adam7 interlacing ");
    break;

  default:
    printf ("Unknown interlacing ");
    break;
  }
printf ("\n");

printf ("  Compression Scheme: ");
switch (info->compression_type)
  {
  case PNG_COMPRESSION_TYPE_BASE:
    printf ("Deflate method 8, 32k window");
    break;

  default:
    printf ("Unknown compression scheme!");
    break;
  }
printf ("\n");

printf ("  Resolution: %d, %d ",
    info->x_pixels_per_unit, info->y_pixels_per_unit);
switch (info->phys_unit_type)
  {
  case PNG_RESOLUTION_UNKNOWN:
    printf ("(unit unknown)");
    break;

  case PNG_RESOLUTION_METER:
    printf ("(pixels per meter)");
    break;

  default:
    printf ("(Unknown value for unit stored)");
    break;
  }
printf ("\n");

// FillOrder is always msb-to-lsb, big endian
printf ("  FillOrder: msb-to-lsb\n  Byte Order: Network (Big Endian)\n");
```

```
  // Text comments
  printf ("  Number of text strings: %d of %d\n",
      info->num_text, info->max_text);

  for (i = 0; i < info->num_text; i++)
    {
      printf ("    %s ", info->text[i].key);

      switch (info->text[1].compression)
    {
    case -1:
      printf ("(tEXt uncompressed)");
      break;

    case 0:
      printf ("(xTXt deflate compressed)");
      break;

    case 1:
      printf ("(iTXt uncompressed)");
      break;

    case 2:
      printf ("(iTXt deflate compressed)");
      break;

    default:
      printf ("(unknown compression)");
      break;
    }

      printf (": ");
      j = 0;
      while (info->text[i].text[j] != '\0')
    {
      if (info->text[i].text[j] == '\n')
        printf ("\\n");
      else
        fputc (info->text[i].text[j], stdout);

      j++;
    }

      printf ("\n");
    }

  // Print a blank line
  printf ("\n");

  // Do we want to extract the image data? We are meant to tell the user if
  // there are errors, but we don't currently trap any errors here --
I need
  // to look into this
  if (extractBitmap == pnginfo_true)
    {
      // This will force the samples to be packed to the byte boundary
      if(bitdepth < 8)
    png_set_packing(png);

      if (colourtype == PNG_COLOR_TYPE_PALETTE)
    png_set_expand (png);

      // png_set_strip_alpha (png);
      png_read_update_info (png, info);
```

```
      rowbytes = png_get_rowbytes (png, info);
      bitmap = (unsigned char *) pnginfo_xmalloc ((rowbytes * height) + 1);
      row_pointers = pnginfo_xmalloc (height * sizeof (png_bytep));

      // Get the image bitmap
      for (i = 0; i < height; ++i)
    row_pointers[i] = bitmap + (i * rowbytes);
      png_read_image (png, row_pointers);
      free (row_pointers);
      png_read_end (png, NULL);

      // Do we want to display this bitmap?
      if (displayBitmap == pnginfo_true)
    {
      int bytespersample;

      bytespersample = bitdepth / 8;
      if(bitdepth % 8 != 0)
        bytespersample++;

      printf ("Dumping the bitmap for this image:\n");
      printf ("(Expanded samples result in %d bytes per pixel, %d chan-
nels with %d bytes per channel)\n\n",
          info->channels * bytespersample, info->channels, bytespersample);

      // runlen is used to stop us displaying repeated byte patterns over and over -
-
      // I display them once, and then tell you how many times it oc-
cured in the file.
      // This currently only applies to runs on zeros -- I should one day add an
      // option to extend this to runs of other values as well
      runlen = 0;
      for (i = 0; i < rowbytes * height / info->channels; i += info->channels * bytesp
        {
          int scount, bcount, pixel;

          if ((runlen != 0) && (bitmap[i] == 0) && (bitmap[i] == 0)
          && (bitmap[i] == 0))
        runlen++;
          else if (runlen != 0)
        {
          if (runlen > 1)
            printf ("* %d ", runlen);
          runlen = 0;
        }

          // Determine if this is a pixel whose entire value is zero
          pixel = 0;
          for(scount = 0; scount < info->channels; scount++)
        for(bcount = 0; bcount < bytespersample; bcount++)
          pixel += bitmap[i + scount * bytespersample + bcount];

          if ((runlen == 0) && !pixel)
        {
          printf ("[");
          for(scount = 0; scount < info->channels; scount++){
            for(bcount = 0; bcount < bytespersample; bcount++) printf("00");
            if(scount != info->channels - 1) printf(" ");
          }
          printf ("] ");
          runlen++;
        }

          if (runlen == 0){
        printf ("[");
        for(scount = 0; scount < info->channels; scount++){
```

```
          for(bcount = 0; bcount < bytespersample; bcount++)
            printf("%02x", (unsigned char) bitmap[i + scount * bytes-
persample + bcount]);
          if(scount != info->channels - 1) printf(" ");
        }
        printf("] ");
          }

          // Perhaps one day a new row should imply a line break here?
        }

      printf("\n");
    }
    }

  // This cleans things up for us in the PNG library
  fclose (image);
  png_destroy_read_struct (&png, &info, NULL);
}

// You can bang or head or you can drown in a hole
//                                                    -- Vanessa Amarosi, Shine
void
pnginfo_error (char *message)
{
  fprintf (stderr, "%s\n", message);
  exit (42);
}

// Allocate some memory
void *
pnginfo_xmalloc (size_t size)
{
  void *buffer;

  if ((buffer = malloc (size)) == NULL)
    {
      pnginfo_error ("pnginfo_xmalloc failed to allocate memory");
    }

  return buffer;
}

void
usage ()
{
  pnginfo_error ("Usage: pnginfo [-d] [-D] <filenames>");
}
```

Code: pnginfo.c

Reading a PNG image

This example shows you how to read in the raster data that is embedded into the image file. libpng will also expand paletted raster information for you if you ask nicely...

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <png.h>
#include <unistd.h>

int main(int argc, char *argv[]){
```

```
 FILE *image;
 unsigned long width, height;
 int bitdepth, colourtype;
 png_uint_32 i, j, rowbytes;
 png_structp png;
 png_infop info;
 png_bytepp row_pointers = NULL;
 unsigned char sig[8];
 char *raster;

 // Open the file
 if ((image = fopen (argv[1], "rb")) == NULL){
   fprintf(stderr, "Could not open the specified PNG file.");
   exit(0);
 }

 // Check that it really is a PNG file
 fread(sig, 1, 8, image);
 if(!png_check_sig(sig, 8)){
   printf("This file is not a valid PNG file\n");
   fclose(image);
   return;
 }

 // Start decompressing
 if((png = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
                  NULL, NULL)) == NULL){
   fprintf(stderr, "Could not create a PNG read structure (out of memory?)");
   exit(0);
 }

 if((info = png_create_info_struct(png)) == NULL){
   fprintf(stderr, "Could not create PNG info structure (out of memory?)");
   exit(0);
 }

 // If pnginfo_error did not exit, we would have to call
 // png_destroy_read_struct
 if(setjmp(png_jmpbuf(png))){
   fprintf(stderr, "Could not set PNG jump value");
   exit(0);
 }

 // Get ready for IO and tell the API we have already read the image signature
 png_init_io(png, image);
 png_set_sig_bytes(png, 8);
 png_read_info(png, info);
 png_get_IHDR(png, info, &width, &height, &bitdepth, &colourtype, NULL,
          NULL, NULL);

if (colourtype == PNG_COLOR_TYPE_PALETTE)
   png_set_expand (png);
 //  if(colourtype & PNG_COLOR_MASK_ALPHA)
 png_set_strip_alpha (png);
 png_read_update_info (png, info);

 rowbytes = png_get_rowbytes (png, info);
 if((row_pointers = malloc (height * sizeof (png_bytep))) == NULL){
   fprintf(stderr, "Could not allocate memory\n");
   exit(42);
 }

 // Space for the bitmap
 if((raster = (unsigned char *) malloc ((rowbytes * height) + 1)) == NULL){
   fprintf(stderr, "Could not allocate memory\n");
   exit(42);
```

```
  }

  // Get the image bitmap
  for (i = 0; i < height; ++i)
    row_pointers[i] = raster + (i * rowbytes);
  png_read_image (png, row_pointers);
  free(row_pointers);
  png_read_end (png, NULL);
  fclose (image);

  // We should dump the bitmap at this point
  for(i = 0; i < width * height; i++){
    printf("%08x ", raster[i]);
  }

  // This cleans things up for us in the PNG library
  png_destroy_read_struct (&png, &info, NULL);
}
```

Code: read.c

Writing a PNG image

Writing a PNG file is *very* to reading the PNG file. The example below extends the code from above to open a PNG file, and then copy it to a new PNG file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <png.h>
#include <unistd.h>

int main(int argc, char *argv[]){
  FILE *image;
  unsigned long width, height;
  int bitdepth, colourtype;
  png_uint_32 i, j, rowbytes;
  png_structp png;
  png_infop info;
  png_bytepp row_pointers = NULL;
  unsigned char sig[8];
  char *raster;

  // Open the file
  if ((image = fopen (argv[1], "rb")) == NULL){
    fprintf(stderr, "Could not open the specified PNG file.");
    exit(0);
  }

  // Check that it really is a PNG file
  fread(sig, 1, 8, image);
  if(!png_check_sig(sig, 8)){
    printf("This file is not a valid PNG file\n");
    fclose(image);
    return;
  }

  // Start decompressing
  if((png = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
                    NULL, NULL)) == NULL){
    fprintf(stderr, "Could not create a PNG read structure (out of memory?)");
    exit(0);
  }
```

```
 if((info = png_create_info_struct(png)) == NULL){
   fprintf(stderr, "Could not create PNG info structure (out of memory?)");
   exit(0);
 }

 // If pnginfo_error did not exit, we would have to call
 // png_destroy_read_struct
 if(setjmp(png_jmpbuf(png))){
   fprintf(stderr, "Could not set PNG jump value");
   exit(0);
 }

 // Get ready for IO and tell the API we have already read the image signature
 png_init_io(png, image);
 png_set_sig_bytes(png, 8);
 png_read_info(png, info);
 png_get_IHDR(png, info, &width, &height, &bitdepth, &colourtype, NULL,
          NULL, NULL);

if (colourtype == PNG_COLOR_TYPE_PALETTE)
   png_set_expand (png);
 //  if(colourtype & PNG_COLOR_MASK_ALPHA)
 png_set_strip_alpha (png);
 png_read_update_info (png, info);

 rowbytes = png_get_rowbytes (png, info);
 if((row_pointers = malloc (height * sizeof (png_bytep))) == NULL){
   fprintf(stderr, "Could not allocate memory\n");
   exit(42);
 }

 // Space for the bitmap
 if((raster = (unsigned char *) malloc ((rowbytes * height) + 1)) == NULL){
   fprintf(stderr, "Could not allocate memory\n");
   exit(42);
 }

 // Get the image bitmap
 for (i = 0; i < height; ++i)
   row_pointers[i] = raster + (i * rowbytes);
 png_read_image (png, row_pointers);
 free(row_pointers);
 png_read_end (png, NULL);
 fclose (image);


 // This cleans things up for us in the PNG library
 png_destroy_read_struct (&png, &info, NULL);

 ////////////////////////////////////////////////////////////////////////////
 // Now write the image out again
 ////////////////////////////////////////////////////////////////////////////
 if((image = fopen("output.png", "wb")) == NULL){
   fprintf(stderr, "Could not open the output image\n");
   exit(42);
 }

 // Get ready for writing
 if ((png =
      png_create_write_struct (PNG_LIBPNG_VER_STRING, NULL, NULL,
                               NULL)) == NULL){
   fprintf(stderr, "Could not create write structure for PNG (out of memory?)");
   exit(42);
 }

 // Get ready to specify important stuff about the image
```

```
  if ((info = png_create_info_struct (png)) == NULL){
    fprintf(stderr,
        "Could not create PNG info structure for writing (out of memory?)");
    exit(42);
  }

  if (setjmp (png_jmpbuf (png))){
    fprintf(stderr, "Could not set the PNG jump value for writing");
    exit(42);
  }

  // This is needed before IO will work (unless you define callbacks)
  png_init_io(png, image);

  // Define important stuff about the image
  png_set_IHDR (png, info, width, height, bitdepth, PNG_COLOR_TYPE_RGB,
                PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_DEFAULT,
                PNG_FILTER_TYPE_DEFAULT);
  png_write_info (png, info);

  // Write the image out
  png_write_image (png, row_pointers);

  // Cleanup
  png_write_end (png, info);
  png_destroy_write_struct (&png, &info);
  fclose(image);
}
```

Code: write.c

Storing PNG data in places other than files

This example below, much like the TIFFClient examples in the TIFF chapter shows how to use the libpng call backs to get to image data in places other than files. Note that it wont compile, but it does give you an example of the structure you need...

```
// This code wont compile, but it provides an example of how to use the libpng
// callbacks to get to image data in other places than files...
#include <png.h>
#include <pthread.h>

#define panda_true 1
#define panda_false 0

// This mutex keeps us thread safe (for the globals)
pthread_mutex_t convMutex = PTHREAD_MUTEX_INITIALIZER;

// Function prototypes
void libpngDummyWriteProc (png_structp png, png_bytep data, png_uint_32 len);
void libpngDummyFlushProc (png_structp png);

// We need thse globals because of the callbacks
char *globalImageBuffer;
unsigned long globalImageBufferOffset;
char globalIsIDAT;

////////////////////////////////////////////////////////////////////////////
// Setup libpng
if ((png =
     png_create_write_struct (PNG_LIBPNG_VER_STRING, NULL, NULL,
                  NULL)) == NULL){
  fprintf(stderr,
      "Could not create write structure for PNG (out of memory?)");
```

```
  exit(42);
}

if ((info = png_create_info_struct (png)) == NULL){
  fprintf(stderr,
      "Could not create PNG info structure for writing (out of memory?)");
  exit(42);
}

if (setjmp (png_jmpbuf (png))){
  fprintf(stderr, "Could not set the PNG jump value for writing");
  exit(42);
}

// If this call is done before png_create_write_struct, then everything
// seg faults...
pthread_mutex_lock (&amp;convMutex);

png_set_write_fn (png, NULL, (png_rw_ptr) libpngDummyWriteProc,
          (png_flush_ptr) libpngDummyFlushProc);
globalIsIDAT = panda_false;
globalImageBuffer = NULL;
globalImageBufferOffset = 0;

png_set_IHDR (png, info, width, height, bitdepth, outColourType,
          PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_DEFAULT,
          PNG_FILTER_TYPE_DEFAULT);
png_write_info (png, info);

png_write_image (png, row_pointers);
png_write_end (png, info);
png_destroy_write_struct (&amp;png, &amp;info);

////////////////////////////////////////////////////////////////////////
// When we get back to here we have the image data

  pthread_mutex_unlock (&amp;convMutex);

////////////////////////////////////////////////////////////////////////
// Callbacks
void
libpngDummyWriteProc (png_structp png, png_bytep data, png_uint_32 len)
{
  char tempString[5];

  // Copy the first 4 bytes into a string
  tempString[0] = data[0];
  tempString[1] = data[1];
  tempString[2] = data[2];
  tempString[3] = data[3];
  tempString[4] = '\0';

  // If we know this is an IDAT, then copy the compressed image information
  if (globalIsIDAT == panda_true)
    {
      // Have we done anything yet?
      if (globalImageBuffer == NULL)
    globalImageBuffer = (char *) panda_xmalloc (len * sizeof (char));

      // Otherwise, we need to grow the memory buffer
      else
    {
      if ((globalImageBuffer = (char *) realloc (globalImageBuffer,
                                 (len * sizeof (char)) +
                                 globalImageBufferOffset))
          == NULL){
```

```
        fprintf(stderr,
             "Could not grow the png conversion memory buffer.");
        exit(42);
      }
    }

      // Now move the image data into the buffer
      memcpy (globalImageBuffer + globalImageBufferOffset, data, len);
      globalImageBufferOffset += len;

      globalIsIDAT = panda_false;
    }
  else if ((len == 4) &amp;amp;&amp;amp; (strcmp (tempString, "IDAT") == 0))
    globalIsIDAT = panda_true;
  else
    globalIsIDAT = panda_false;
}

void
libpngDummyFlushProc (png_structp png)
{
}
```

Code: client.c

Man pages

To be honest, the libtiff man pages are much better than the documentation which comes with libpng. libpng has one main man page, which is included for ease of reference below.

libpng

NAME

libpng - Portable Network Graphics (PNG) Reference Library 1.0.9

SYNOPSIS

#include <png.h>

png_uint_32 png_access_version_number (void);

int png_check_sig (png_bytep *sig*, **int** *num*);

void png_chunk_error (png_structp *png_ptr*, **png_const_charp** *error*);

void png_chunk_warning (png_structp *png_ptr*, **png_const_charp** *message*);

void png_convert_from_struct_tm (png_timep *ptime*, **struct tm FAR *** *ttime*);

void png_convert_from_time_t (png_timep *ptime*, **time_t** *ttime*);

png_charp png_convert_to_rfc1123 (png_structp *png_ptr*, **png_timep** *ptime*);

png_infop png_create_info_struct (png_structp *png_ptr*);

png_structp png_create_read_struct (png_const_charp *user_png_ver*, **png_voidp** *error_ptr*, **png_error_ptr** *error_fn*, **png_error_ptr** *warn_fn*);

png_structp png_create_read_struct_2(png_const_charp *user_png_ver*, **png_voidp** *error_ptr*, **png_error_ptr** *error_fn*, **png_error_ptr** *warn_fn*, **png_voidp** *mem_ptr*, **png_malloc_ptr** *malloc_fn*, **png_free_ptr** *free_fn*);

png_structp png_create_write_struct (png_const_charp *user_png_ver*, **png_voidp** *error_ptr*, **png_error_ptr** *error_fn*, **png_error_ptr** *warn_fn*);

png_structp png_create_write_struct_2(png_const_charp *user_png_ver*, **png_voidp** *error_ptr*, **png_error_ptr** *error_fn*, **png_error_ptr** *warn_fn*, **png_voidp** *mem_ptr*, **png_malloc_ptr** *malloc_fn*, **png_free_ptr** *free_fn*);

int png_debug(int *level*, **png_const_charp** *message*);

int png_debug1(int *level*, **png_const_charp** *message*, *p1*);

int png_debug2(int *level*, **png_const_charp** *message*, *p1*, *p2*);

void png_destroy_info_struct (png_structp *png_ptr*, **png_infopp** *info_ptr_ptr*);

void png_destroy_read_struct (png_structpp *png_ptr_ptr*, **png_infopp** *info_ptr_ptr*, **png_infopp** *end_info_ptr_ptr*);

void png_destroy_write_struct (png_structpp *png_ptr_ptr*, **png_infopp** *info_ptr_ptr*);

void png_error (png_structp *png_ptr*, **png_const_charp** *error*);

void png_free (png_structp *png_ptr*, **png_voidp** *ptr*);

void png_free_chunk_list (png_structp *png_ptr*);

void png_free_default(png_structp *png_ptr*, **png_voidp** *ptr*);

void png_free_data (png_structp *png_ptr*, **png_infop** *info_ptr*, **int** *num*);

png_byte png_get_bit_depth (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_bKGD (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_color_16p** **background*);

png_byte png_get_channels (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_cHRM (png_structp *png_ptr*, **png_infop** *info_ptr*, **double** **white_x*, **double** **white_y*, **double** **red_x*, **double** **red_y*, **double** **green_x*, **double** **green_y*, **double** **blue_x*, **double** **blue_y*);

png_uint_32 png_get_cHRM_fixed (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** **white_x*, **png_uint_32** **white_y*, **png_uint_32** **red_x*, **png_uint_32** **red_y*, **png_uint_32** **green_x*, **png_uint_32** **green_y*, **png_uint_32** **blue_x*, **png_uint_32** **blue_y*);

png_byte png_get_color_type (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_byte png_get_compression_type (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_byte png_get_copyright (png_structp *png_ptr*);

png_voidp png_get_error_ptr (png_structp *png_ptr*);

png_byte png_get_filter_type (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_gAMA (png_structp *png_ptr*, **png_infop** *info_ptr*, **double** **file_gamma*);

png_uint_32 png_get_gAMA_fixed (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** **int_file_gamma*);

png_byte png_get_header_ver (png_structp *png_ptr*);

png_byte png_get_header_version (png_structp *png_ptr*);

png_uint_32 png_get_hIST (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_16p** **hist*);

png_uint_32 png_get_iCCP (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_charpp** *name*, **int** **compression_type*, **png_charpp** *profile*, **png_uint_32** **proflen*);

png_uint_32 png_get_IHDR (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** **width*, **png_uint_32** **height*, **int** **bit_depth*, **int** **color_type*, **int** **interlace_type*, **int** **compression_type*, **int** **filter_type*);

png_uint_32 png_get_image_height (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_image_width (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_byte png_get_interlace_type (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_voidp png_get_io_ptr (png_structp *png_ptr*);

png_byte png_get_libpng_ver (png_structp *png_ptr*);

png_voidp png_get_mem_ptr(png_structp *png_ptr*);

png_uint_32 png_get_oFFs (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** **offset_x*, **png_uint_32** **offset_y*, **int** **unit_type*);

png_uint_32 png_get_pCAL (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_charp** **purpose*, **png_int_32** **X0*, **png_int_32** **X1*, **int** **type*, **int** **nparams*, **png_charp** **units*, **png_charpp** **params*);

png_uint_32 png_get_pHYs (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** **res_x*, **png_uint_32** **res_y*, **int** **unit_type*);

float png_get_pixel_aspect_ratio (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_pixels_per_meter (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_voidp png_get_progressive_ptr (png_structp *png_ptr*);

png_uint_32 png_get_PLTE (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_colorp** **palette*, **int** **num_palette*);

png_byte png_get_rgb_to_gray_status (png_structp *png_ptr)*

png_uint_32 png_get_rowbytes (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_bytepp png_get_rows (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_sBIT (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_color_8p** **sig_bit*);

png_bytep png_get_signature (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_sPLT (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_spalette_p** **splt_ptr*);

png_uint_32 png_get_sRGB (png_structp *png_ptr*, **png_infop** *info_ptr*, **int** **intent*);

png_uint_32 png_get_text (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_textp** **text_ptr*, **int** **num_text*);

png_uint_32 png_get_tIME (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_timep** **mod_time*);

png_uint_32 png_get_tRNS (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_bytep** **trans*, **int** **num_trans*, **png_color_16p** **trans_values*);

png_uint_32 png_get_unknown_chunks (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_unknown_chunkpp** *unknowns*);

png_voidp png_get_user_chunk_ptr (png_structp *png_ptr*);

png_voidp png_get_user_transform_ptr (png_structp *png_ptr*);

png_uint_32 png_get_valid (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** *flag*);

png_int_32 png_get_x_offset_microns (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_int_32 png_get_x_offset_pixels (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_x_pixels_per_meter (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_int_32 png_get_y_offset_microns (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_int_32 png_get_y_offset_pixels (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_y_pixels_per_meter (png_structp *png_ptr*, **png_infop** *info_ptr*);

png_uint_32 png_get_compression_buffer_size (png_structp *png_ptr*);

void png_info_init (png_infop *info_ptr*);

void png_init_io (png_structp *png_ptr*, **FILE** **fp*);

png_voidp png_malloc (png_structp *png_ptr*, **png_uint_32** *size*);

png_voidp png_malloc_default(png_structp *png_ptr*, **png_uint_32** *size*);

voidp png_memcpy (png_voidp *s1*, **png_voidp** *s2*, **png_size_t** *size*);

png_voidp png_memcpy_check (png_structp *png_ptr*, **png_voidp** *s1*, **png_voidp** *s2*, **png_uint_32** *size*);

voidp png_memset (png_voidp *s1*, **int** *value*, **png_size_t** *size*);

png_voidp png_memset_check (png_structp *png_ptr*, **png_voidp** *s1*, **int** *value*, **png_uint_32** *size*);

void png_permit_empty_plte (png_structp *png_ptr*, **int** *empty_plte_permitted*);

void png_process_data (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_bytep** *buffer*, **png_size_t** *buffer_size*);

void png_progressive_combine_row (png_structp *png_ptr*, **png_bytep** *old_row*, **png_bytep** *new_row*);

void png_read_destroy (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_infop** *end_info_ptr*);

void png_read_end (png_structp *png_ptr*, **png_infop** *info_ptr*);

void png_read_image (png_structp *png_ptr*, **png_bytepp** *image*);

DEPRECATED: void png_read_init (png_structp *png_ptr*);

DEPRECATED: void png_read_init_2 (png_structp *png_ptr*, **png_const_charp** *user_png_ver*, **png_size_t** *png_struct_size*, **png_size_t** *png_info_size*);

void png_read_info (png_structp *png_ptr*, **png_infop** *info_ptr*);

void png_read_png (png_structp *png_ptr*, **png_infop** *info_ptr*, **int** *transforms*, **png_voidp** *params*);

void png_read_row (png_structp *png_ptr*, **png_bytep** *row*, **png_bytep** *display_row*);

void png_read_rows (png_structp *png_ptr*, **png_bytepp** *row*, **png_bytepp** *display_row*, **png_uint_32** *num_rows*);

void png_read_update_info (png_structp *png_ptr*, **png_infop** *info_ptr*);

void png_set_background (png_structp *png_ptr*, **png_color_16p** *background_color*, **int** *background_gamma_code*, **int** *need_expand*, **double** *background_gamma*);

void png_set_bgr (png_structp *png_ptr*);

void png_set_bKGD (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_color_16p** *background*);

void png_set_cHRM (png_structp *png_ptr*, **png_infop** *info_ptr*, **double** *white_x*, **double** *white_y*, **double** *red_x*, **double** *red_y*, **double** *green_x*, **double** *green_y*, **double** *blue_x*, **double** *blue_y*);

void png_set_cHRM_fixed (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** *white_x*, **png_uint_32** *white_y*, **png_uint_32** *red_x*, **png_uint_32** *red_y*, **png_uint_32** *green_x*, **png_uint_32** *green_y*, **png_uint_32** *blue_x*, **png_uint_32** *blue_y*);

void png_set_compression_level (png_structp *png_ptr*, **int** *level*);

void png_set_compression_mem_level (png_structp *png_ptr*, **int** *mem_level*);

void png_set_compression_method (png_structp *png_ptr*, **int** *method*);

void png_set_compression_strategy (png_structp *png_ptr*, **int** *strategy*);

void png_set_compression_window_bits (png_structp *png_ptr*, **int** *window_bits*);

void png_set_crc_action (png_structp *png_ptr*, **int** *crit_action*, **int** *ancil_action*);

void png_set_dither (png_structp *png_ptr*, **png_colorp** *palette*, **int** *num_palette*, **int** *maximum_colors*, **png_uint_16p** *histogram*, **int** *full_dither*);

void png_set_error_fn (png_structp *png_ptr*, **png_voidp** *error_ptr*, **png_error_ptr** *error_fn*, **png_error_ptr** *warning_fn*);

void png_set_expand (png_structp *png_ptr*);

void png_set_filler (png_structp *png_ptr*, **png_uint_32** *filler*, **int** *flags*);

void png_set_filter (png_structp *png_ptr*, **int** *method*, **int** *filters*);

void png_set_filter_heuristics (png_structp *png_ptr*, **int** *heuristic_method*, **int** *num_weights*, **png_doublep** *filter_weights*, **png_doublep** *filter_costs*);

void png_set_flush (png_structp *png_ptr*, **int** *nrows*);

void png_set_gamma (png_structp *png_ptr*, **double** *screen_gamma*, **double** *default_file_gamma*);

void png_set_gAMA (png_structp *png_ptr*, **png_infop** *info_ptr*, **double** *file_gamma*);

void png_set_gAMA_fixed (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** *file_gamma*);

void png_set_gray_1_2_4_to_8(png_structp *png_ptr*);

void png_set_gray_to_rgb (png_structp *png_ptr*);

void png_set_hIST (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_16p** *hist*);

void png_set_iCCP (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_charp** *name*, **int** *compression_type*, **png_charp** *profile*, **png_uint_32** *proflen*);

int png_set_interlace_handling (png_structp *png_ptr*);

void png_set_invalid (png_structp *png_ptr*, **png_infop** *info_ptr*, **int** *mask*);

void png_set_invert_alpha (png_structp *png_ptr*);

void png_set_invert_mono (png_structp *png_ptr*);

void png_set_IHDR (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** *width*, **png_uint_32** *height*, **int** *bit_depth*, **int** *color_type*, **int** *interlace_type*, **int** *compression_type*, **int** *filter_type*);

void png_set_keep_unknown_chunks (png_structp *png_ptr*, **int** *keep*, **png_bytep** *chunk_list*, **int** *num_chunks*);

void png_set_mem_fn(png_structp *png_ptr*, **png_voidp** *mem_ptr*, **png_malloc_ptr** *malloc_fn*, **png_free_ptr** *free_fn*);

void png_set_oFFs (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** *offset_x*, **png_uint_32** *offset_y*, **int** *unit_type*);

void png_set_packing (png_structp *png_ptr*);

void png_set_packswap (png_structp *png_ptr*);

void png_set_palette_to_rgb(png_structp *png_ptr*);

void png_set_pCAL (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_charp** *purpose*, **png_int_32** *X0*, **png_int_32** *X1*, **int** *type*, **int** *nparams*, **png_charp** *units*, **png_charpp** *params*);

void png_set_pHYs (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_uint_32** *res_x*, **png_uint_32** *res_y*, **int** *unit_type*);

void png_set_progressive_read_fn (png_structp *png_ptr*, **png_voidp** *progressive_ptr*, **png_progressive_info_ptr** *info_fn*, **png_progressive_row_ptr** *row_fn*, **png_progressive_end_ptr** *end_fn*);

void png_set_PLTE (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_colorp** *palette*, **int** *num_palette*);

void png_set_read_fn (png_structp *png_ptr*, **png_voidp** *io_ptr*, **png_rw_ptr** *read_data_fn*);

void png_set_read_status_fn (png_structp *png_ptr*, **png_read_status_ptr** *read_row_fn*);

void png_set_read_user_transform_fn (png_structp *png_ptr*, **png_user_transform_ptr** *read_user_transform_fn*);

void png_set_rgb_to_gray (png_structp *png_ptr*, **int** *error_action*, **double** *red*, **double** *green*);

void png_set_rgb_to_gray_fixed (png_structp *png_ptr*, **int error_action png_fixed_point** *red*, **png_fixed_point** *green*);

void png_set_rows (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_bytepp** *row_pointers*);

void png_set_sBIT (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_color_8p** *sig_bit*);

void png_set_sCAL (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_charp** *unit*, **double** *width*, **double** *height*);

void png_set_shift (png_structp *png_ptr*, **png_color_8p** *true_bits*);

void png_set_sig_bytes (png_structp *png_ptr*, **int** *num_bytes*);

void png_set_sPLT (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_spalette_p** *splt_ptr*, **int** *num_spalettes*);

void png_set_sRGB (png_structp *png_ptr*, **png_infop** *info_ptr*, **int** *intent*);

void png_set_sRGB_gAMA_and_cHRM (png_structp *png_ptr*, **png_infop** *info_ptr*, **int** *intent*);

void png_set_strip_16 (png_structp *png_ptr*);

void png_set_strip_alpha (png_structp *png_ptr*);

void png_set_swap (png_structp *png_ptr*);

void png_set_swap_alpha (png_structp *png_ptr*);

void png_set_text (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_textp** *text_ptr*, **int** *num_text*);

void png_set_tIME (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_timep** *mod_time*);

void png_set_tRNS (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_bytep** *trans*, **int** *num_trans*, **png_color_16p** *trans_values*);

void png_set_tRNS_to_alpha(png_structp *png_ptr*);

png_uint_32 png_set_unknown_chunks (png_structp *png_ptr*, **png_infop** *info_ptr*, **png_unknown_chunkp** *unknowns*, **int** *num*, **int** *location*);

void png_set_unknown_chunk_location(png_structp *png_ptr*, **png_infop** *info_ptr*, **int** *chunk*, **int** *location*);

void png_set_read_user_chunk_fn (png_structp *png_ptr*, **png_voidp** *user_chunk_ptr*, **png_user_chunk_ptr** *read_user_chunk_fn*);

void png_set_user_transform_info (png_structp *png_ptr*, **png_voidp** *user_transform_ptr*, **int** *user_transform_depth*, **int** *user_transform_channels*);

void png_set_write_fn (png_structp *png_ptr*, **png_voidp** *io_ptr*, **png_rw_ptr** *write_data_fn*, **png_flush_ptr** *output_flush_fn*);

void png_set_write_status_fn (png_structp *png_ptr*, **png_write_status_ptr** *write_row_fn*);

void png_set_write_user_transform_fn (png_structp *png_ptr*, **png_user_transform_ptr** *write_user_transform_fn*);

void png_set_compression_buffer_size(png_structp *png_ptr*, **png_uint_32** *size*);

int png_sig_cmp (png_bytep *sig*, **png_size_t** *start*, **png_size_t** *num_to_check*);

void png_start_read_image (png_structp *png_ptr*);

void png_warning (png_structp *png_ptr*, **png_const_charp** *message*);

void png_write_chunk (png_structp *png_ptr*, **png_bytep** *chunk_name*, **png_bytep** *data*, **png_size_t** *length*);

void png_write_chunk_data (png_structp *png_ptr*, **png_bytep** *data*, **png_size_t** *length*);

void png_write_chunk_end (png_structp *png_ptr*);

void png_write_chunk_start (png_structp *png_ptr*, **png_bytep** *chunk_name*, **png_uint_32** *length*);

void png_write_destroy (png_structp *png_ptr*);

void png_write_destroy_info (png_infop *info_ptr*);

void png_write_end (png_structp *png_ptr*, **png_infop** *info_ptr*);

void png_write_flush (png_structp *png_ptr*);

void png_write_image (png_structp *png_ptr*, **png_bytepp** *image*);

DEPRECATED: void png_write_init (png_structp *png_ptr*);

DEPRECATED: void png_write_init_2 (png_structp *png_ptr*, **png_const_charp** *user_png_ver*, **png_size_t** *png_struct_size*, **png_size_t** *png_info_size*);

void png_write_info (png_structp *png_ptr*, **png_infop** *info_ptr*);

void png_write_info_before_PLTE (png_structp *png_ptr*, **png_infop** *info_ptr*);

void png_write_png (png_structp *png_ptr*, **png_infop** *info_ptr*, **int** *transforms*, **png_voidp** *params*);

void png_write_row (png_structp *png_ptr*, **png_bytep** *row*);

void png_write_rows (png_structp *png_ptr*, **png_bytepp** *row*, **png_uint_32** *num_rows*);

DESCRIPTION

The *libpng* library supports encoding, decoding, and various manipulations of the Portable Network Graphics (PNG) format image files. It uses the *zlib(3)* compression library. Following is a copy of the libpng.txt file that accompanies libpng.

LIBPNG.TXT

libpng.txt - A description on how to use and modify libpng

libpng version 1.0.9 - January 31, 2001 Updated and distributed by Glenn Randers-Pehrson <randeg@alum.rpi.edu> Copyright (c) 1998, 1999, 2000 Glenn Randers-Pehrson For conditions of distribution and use, see copyright notice in png.h.

based on:

libpng 1.0 beta 6 version 0.96 May 28, 1997 Updated and distributed by Andreas Dilger Copyright (c) 1996, 1997 Andreas Dilger

libpng 1.0 beta 2 - version 0.88 January 26, 1996 For conditions of distribution and use, see copyright notice in png.h. Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.

Updated/rewritten per request in the libpng FAQ Copyright (c) 1995, 1996 Frank J. T. Wojcik December 18, 1995 & January 20, 1996

I. Introduction

This file describes how to use and modify the PNG reference library (known as libpng) for your own use. There are five sections to this file: introduction, structures, reading, writing, and modification and configuration notes for various special platforms. In addition to this file, example.c is a good starting point for using the library, as it is heavily commented and should include everything most people will need. We assume that libpng is already installed; see the INSTALL file for instructions on how to install libpng.

Libpng was written as a companion to the PNG specification, as a way of reducing the amount of time and effort it takes to support the PNG file format in application programs.

The PNG-1.2 specification is available at <http://www.libpng.org/pub/png> and at <ftp://ftp.uu.net/graphics/png/documents/>.

The PNG-1.0 specification is available as RFC 2083 <ftp://ftp.uu.net/graphics/png/documents/> and as a W3C Recommendation <http://www.w3.org/TR/REC.png.html>. Some additional chunks are described in the special-purpose public chunks documents at <ftp://ftp.uu.net/graphics/png/documents/>.

Other information about PNG, and the latest version of libpng, can be found at the PNG home page, <http://www.libpng.org/pub/png/> and at <ftp://ftp.uu.net/graphics/png/>.

Most users will not have to modify the library significantly; advanced users may want to modify it more. All attempts were made to make it as complete as possible, while keeping the code easy to understand. Currently, this library only supports C. Support for other languages is being considered.

Libpng has been designed to handle multiple sessions at one time, to be easily modifiable, to be portable to the vast majority of machines (ANSI, K&R, 16-, 32-, and 64-bit) available, and to be easy to use. The ultimate goal of libpng is to promote the acceptance of the PNG file format in whatever way possible. While there is still work to be done (see the TODO file), libpng should cover the majority of the needs of its users.

Libpng uses zlib for its compression and decompression of PNG files. Further information about zlib, and the latest version of zlib, can be found at the zlib home page, <http://www.info-zip.org/pub/infozip/zlib/>. The zlib compression utility is a general purpose utility that is useful for more than PNG files, and can be used without libpng. See the documentation delivered with zlib for more details. You can usually find the source files for the zlib utility wherever you find the libpng source files.

Libpng is thread safe, provided the threads are using different instances of the structures. Each thread should have its own png_struct and png_info instances, and thus its own image. Libpng does not protect itself against two threads using the same instance of a structure.

II. Structures

There are two main structures that are important to libpng, png_struct and png_info. The first, png_struct, is an internal structure that will not, for the most part, be used by a user except as the first variable passed to every libpng function call.

The png_info structure is designed to provide information about the PNG file. At one time, the fields of png_info were intended to be directly accessible to the user. However, this tended to cause problems with applications using dynamically loaded libraries, and as a result a set of interface functions for png_info (the png_get_*() and png_set_*() functions) was developed. The fields of png_info are still available for older applications, but it is suggested that applications use the new interfaces if at all possible.

Applications that do make direct access to the members of png_struct (except for png_ptr->jmpbuf) must be recompiled whenever the library is updated, and applications that make direct access to the members of png_info must be recompiled if they were compiled or loaded with libpng version 1.0.6, in which the members were in a different order. In version 1.0.7, the members of the png_info structure reverted to the old order, as they were in versions 0.97c through 1.0.5. Starting with version 2.0.0, both structures are going to be hidden, and the contents of the structures will only be accessible through the png_get/png_set functions.

The png.h header file is an invaluable reference for programming with libpng. And while I'm on the topic, make sure you include the libpng header file:

#include <png.h>

III. Reading

We'll now walk you through the possible functions to call when reading in a PNG file sequentially, briefly explaining the syntax and purpose of each one. See example.c and png.h for more detail. While progressive reading is covered in the next section, you will still need some of the functions discussed in this section to read a PNG file.

You will want to do the I/O initialization(*) before you get into libpng, so if it doesn't work, you don't have much to undo. Of course, you will also want to insure that you are, in fact, dealing with a PNG file. Libpng provides a simple check to see if a file is a PNG file. To use it, pass in the first 1 to 8 bytes of the file to the function

png_sig_cmp(), and it will return 0 if the bytes match the corresponding bytes of the PNG signature, or nonzero otherwise. Of course, the more bytes you pass in, the greater the accuracy of the prediction.

If you are intending to keep the file pointer open for use in libpng, you must ensure you don't read more than 8 bytes from the beginning of the file, and you also have to make a call to png_set_sig_bytes_read() with the number of bytes you read from the beginning. Libpng will then only check the bytes (if any) that your program didn't read.

(*): If you are not using the standard I/O functions, you will need to replace them with custom functions. See the discussion under Customizing libpng.

FILE *fp = fopen(file_name, "rb"); if (!fp) { return (ERROR); } fread(header, 1, number, fp); is_png = !png_sig_cmp(header, 0, number); if (!is_png) { return (NOT_PNG); }

Next, png_struct and png_info need to be allocated and initialized. In order to ensure that the size of these structures is correct even with a dynamically linked libpng, there are functions to initialize and allocate the structures. We also pass the library version, optional pointers to error handling functions, and a pointer to a data struct for use by the error functions, if necessary (the pointer and functions can be NULL if the default error handlers are to be used). See the section on Changes to Libpng below regarding the old initialization functions. The structure allocation functions quietly return NULL if they fail to create the structure, so your application should check for that.

png_structp png_ptr = png_create_read_struct (PNG_LIBPNG_VER_STRING, (png_voidp)user_error_ptr, user_error_fn, user_warning_fn); if (!png_ptr) return (ERROR);

png_infop info_ptr = png_create_info_struct(png_ptr); if (!info_ptr) { png_destroy_read_struct(&png_ptr, (png_infopp)NULL, (png_infopp)NULL); return (ERROR); }

png_infop end_info = png_create_info_struct(png_ptr); if (!end_info) { png_destroy_read_struct(&png_ptr, &info_ptr, (png_infopp)NULL); return (ERROR); }

If you want to use your own memory allocation routines, define PNG_USER_MEM_SUPPORTED and use png_create_read_struct_2() instead of png_create_read_struct():

png_structp png_ptr = png_create_read_struct_2 (PNG_LIBPNG_VER_STRING, (png_voidp)user_error_ptr, user_error_fn, user_warning_fn, (png_voidp) user_mem_ptr, user_malloc_fn, user_free_fn);

The error handling routines passed to png_create_read_struct() and the memory alloc/free routines passed to png_create_struct_2() are only necessary if you are not using the libpng supplied error handling and memory alloc/free functions.

When libpng encounters an error, it expects to longjmp back to your routine. Therefore, you will need to call setjmp and pass your png_jmpbuf(png_ptr). If you read the file from different routines, you will need to update the jmpbuf field every time you enter a new routine that will call a png_*() function.

See your documentation of setjmp/longjmp for your compiler for more information on setjmp/longjmp. See the discussion on libpng error handling in the Customizing Libpng section below for more information on the libpng error handling. If an error occurs, and libpng longjmp's back to your setjmp, you will want to call png_destroy_read_struct() to free any memory.

if (setjmp(png_jmpbuf(png_ptr))) { png_destroy_read_struct(&png_ptr, &info_ptr, &end_info); fclose(fp); return (ERROR); }

If you would rather avoid the complexity of setjmp/longjmp issues, you can compile libpng with PNG_SETJMP_NOT_SUPPORTED, in which case errors will result in a

call to PNG_ABORT() which defaults to abort().

Now you need to set up the input code. The default for libpng is to use the C function fread(). If you use this, you will need to pass a valid FILE * in the function png_init_io(). Be sure that the file is opened in binary mode. If you wish to handle reading data in another way, you need not call the png_init_io() function, but you must then implement the libpng I/O methods discussed in the Customizing Libpng section below.

png_init_io(png_ptr, fp);

If you had previously opened the file and read any of the signature from the beginning in order to see if this was a PNG file, you need to let libpng know that there are some bytes missing from the start of the file.

png_set_sig_bytes(png_ptr, number);

You can set up a callback function to handle any unknown chunks in the input stream. You must supply the function

read_chunk_callback(png_ptr ptr, png_unknown_chunkp chunk); { /* The unknown chunk structure contains your chunk data: */ png_byte name[5]; png_byte *data; png_size_t size; /* Note that libpng has already taken care of the CRC handling */

/* put your code here. Return one of the following: */

return (-n); /* chunk had an error */ return (0); /* did not recognize */ return (n); /* success */ }

(You can give your function another name that you like instead of "read_chunk_callback")

To inform libpng about your function, use

png_set_read_user_chunk_fn(png_ptr, user_chunk_ptr, read_chunk_callback);

This names not only the callback function, but also a user pointer that you can retrieve with

png_get_user_chunk_ptr(png_ptr);

At this point, you can set up a callback function that will be called after each row has been read, which you can use to control a progress meter or the like. It's demonstrated in pngtest.c. You must supply a function

void read_row_callback(png_ptr ptr, png_uint_32 row, int pass); { /* put your code here */ }

(You can give it another name that you like instead of "read_row_callback")

To inform libpng about your function, use

png_set_read_status_fn(png_ptr, read_row_callback);

Now you get to set the way the library processes unknown chunks in the input PNG stream. Both known and unknown chunks will be read. Normal behavior is that known chunks will be parsed into information in various info_ptr members; unknown chunks will be discarded. To change this, you can call:

png_set_keep_unknown_chunks(png_ptr, info_ptr, keep, chunk_list, num_chunks); keep - 0: do not keep 1: keep only if safe-to-copy 2: keep even if unsafe-to-copy chunk_list - list of chunks affected (a byte string, five bytes per chunk, NULL or '\0' if num_chunks is 0) num_chunks - number of chunks affected; if 0, all unknown chunks are affected

Unknown chunks declared in this way will be saved as raw data onto a list of png_unknown_chunk structures. If a chunk that is normally known to libpng is named in the list, it will be handled as unknown, according to the "keep" directive. If a chunk is named in successive instances of png_set_keep_unknown_chunks(), the final instance will take precedence.

At this point there are two ways to proceed; through the high-level read interface, or through a sequence of low-level read operations. You can use the high-level interface if (a) you are willing to read the entire image into memory, and (b) the input transformations you want to do are limited to the following set:

PNG_TRANSFORM_IDENTITY No transformation PNG_TRANSFORM_STRIP_16 Strip 16-bit samples to 8 bits PNG_TRANSFORM_STRIP_ALPHA Discard the alpha channel PNG_TRANSFORM_PACKING Expand 1, 2 and 4-bit samples to bytes PNG_TRANSFORM_PACKSWAP Change order of packed pixels to LSB first PNG_TRANSFORM_EXPAND Perform set_expand() PNG_TRANSFORM_INVERT_MONO Invert monochrome images PNG_TRANSFORM_SHIFT Normalize pixels to the sBIT depth PNG_TRANSFORM_BGR Flip RGB to BGR, RGBA to BGRA PNG_TRANSFORM_SWAP_ALPHA Flip RGBA to ARGB or GA to AG PNG_TRANSFORM_INVERT_ALPHA Change alpha from opacity to transparency PNG_TRANSFORM_SWAP_ENDIAN Byte-swap 16-bit samples

(This excludes setting a background color, doing gamma transformation, dithering, and setting filler.) If this is the case, simply do this:

png_read_png(png_ptr, info_ptr, png_transforms, NULL)

where png_transforms is an integer containing the logical OR of some set of transformation flags. This call is equivalent to png_read_info(), followed the set of transformations indicated by the transform mask, then png_read_image(), and finally png_read_end().

(The final parameter of this call is not yet used. Someday it might point to transformation parameters required by some future input transform.)

After you have called png_read_png(), you can retrieve the image data with

row_pointers = png_get_rows(png_ptr, info_ptr);

where row_pointers is an array of pointers to the pixel data for each row:

png_bytep row_pointers[height];

If you know your image size and pixel size ahead of time, you can allocate row_pointers prior to calling png_read_png() with

row_pointers = png_malloc(png_ptr, height*sizeof(png_bytep)); for (int i=0; i<height, i++) row_pointers[i]=png_malloc(png_ptr, width*pixel_size); png_set_rows(png_ptr, info_ptr, &row_pointers);

Alternatively you could allocate your image in one big block and define row_pointers[i] to point into the proper places in your block.

If you use png_set_rows(), the application is responsible for freeing row_pointers (and row_pointers[i], if they were separately allocated).

If you don't allocate row_pointers ahead of time, png_read_png() will do it, and it'll be free'ed when you call png_destroy_*().

If you are going the low-level route, you are now ready to read all the file information up to the actual image data. You do this with a call to png_read_info().

png_read_info(png_ptr, info_ptr);

This will process all chunks up to but not including the image data.

Functions are used to get the information from the info_ptr once it has been read. Note that these fields may not be completely filled in until png_read_end() has read the chunk data following the image.

png_get_IHDR(png_ptr, info_ptr, &width, &height, &bit_depth, &color_type, &interlace_type, &compression_type, &filter_method);

width - holds the width of the image in pixels (up to 2^31). height - holds the height of the image in pixels (up to 2^31). bit_depth - holds the bit depth of

one of the image channels. (valid values are 1, 2, 4, 8, 16 and depend also on the color_type. See also significant bits (sBIT) below). color_type - describes which color/alpha channels are present. PNG_COLOR_TYPE_GRAY (bit depths 1, 2, 4, 8, 16) PNG_COLOR_TYPE_GRAY_ALPHA (bit depths 8, 16) PNG_COLOR_TYPE_PALETTE (bit depths 1, 2, 4, 8) PNG_COLOR_TYPE_RGB (bit_depths 8, 16) PNG_COLOR_TYPE_RGB_ALPHA (bit_depths 8, 16)

PNG_COLOR_MASK_PALETTE PNG_COLOR_MASK_COLOR PNG_COLOR_MASK_ALPHA

filter_method - (must be PNG_FILTER_TYPE_BASE for PNG 1.0, and can also be PNG_INTRAPIXEL_DIFFERENCING if the PNG datastream is embedded in a MNG-1.0 datastream) compression_type - (must be PNG_COMPRESSION_TYPE_BASE for PNG 1.0) interlace_type - (PNG_INTERLACE_NONE or PNG_INTERLACE_ADAM7) Any or all of interlace_type, compression_type, of filter_method can be NULL if you are not interested in their values.

channels = png_get_channels(png_ptr, info_ptr); channels - number of channels of info for the color type (valid values are 1 (GRAY, PALETTE), 2 (GRAY_ALPHA), 3 (RGB), 4 (RGB_ALPHA or RGB + filler byte)) rowbytes = png_get_rowbytes(png_ptr, info_ptr); rowbytes - number of bytes needed to hold a row

signature = png_get_signature(png_ptr, info_ptr); signature - holds the signature read from the file (if any). The data is kept in the same offset it would be if the whole signature were read (i.e. if an application had already read in 4 bytes of signature before starting libpng, the remaining 4 bytes would be in signature[4] through signature[7] (see png_set_sig_bytes())).

width = png_get_image_width(png_ptr, info_ptr); height = png_get_image_height(png_ptr, info_ptr); bit_depth = png_get_bit_depth(png_ptr, info_ptr); color_type = png_get_color_type(png_ptr, info_ptr); filter_method = png_get_filter_type(png_ptr, info_ptr); compression_type = png_get_compression_type(png_ptr, info_ptr); interlace_type = png_get_interlace_type(png_ptr, info_ptr);

These are also important, but their validity depends on whether the chunk has been read. The png_get_valid(png_ptr, info_ptr, PNG_INFO_<chunk>) and png_get_<chunk>(png_ptr, info_ptr, ...) functions return non-zero if the data has been read, or zero if it is missing. The parameters to the png_get_<chunk> are set directly if they are simple data types, or a pointer into the info_ptr is returned for any complex types.

png_get_PLTE(png_ptr, info_ptr, &palette, &num_palette); palette - the palette for the file (array of png_color) num_palette - number of entries in the palette

png_get_gAMA(png_ptr, info_ptr, &gamma); gamma - the gamma the file is written at (PNG_INFO_gAMA)

png_get_sRGB(png_ptr, info_ptr, &srgb_intent); srgb_intent - the rendering intent (PNG_INFO_sRGB) The presence of the sRGB chunk means that the pixel data is in the sRGB color space. This chunk also implies specific values of gAMA and cHRM.

png_get_iCCP(png_ptr, info_ptr, &name, &compression_type, &profile, &proflen); name - The profile name. compression - The compression type; always PNG_COMPRESSION_TYPE_BASE for PNG 1.0. You may give NULL to this argument to ignore it. profile - International Color Consortium color profile data. May contain NULs. proflen - length of profile data in bytes.

png_get_sBIT(png_ptr, info_ptr, &sig_bit); sig_bit - the number of significant bits for (PNG_INFO_sBIT) each of the gray, red, green, and blue channels, whichever are appropriate for the given color type (png_color_16)

png_get_tRNS(png_ptr, info_ptr, &trans, &num_trans, &trans_values); trans - array of transparent entries for palette (PNG_INFO_tRNS) trans_values - graylevel

or color sample values of the single transparent color for non-paletted images (PNG_INFO_tRNS) num_trans - number of transparent entries (PNG_INFO_tRNS)

png_get_hIST(png_ptr, info_ptr, &hist); (PNG_INFO_hIST) hist - histogram of palette (array of png_uint_16)

png_get_tIME(png_ptr, info_ptr, &mod_time); mod_time - time image was last modified (PNG_VALID_tIME)

png_get_bKGD(png_ptr, info_ptr, &background); background - background color (PNG_VALID_bKGD) valid 16-bit red, green and blue values, regardless of color_type

num_comments = png_get_text(png_ptr, info_ptr, &text_ptr, &num_text); num_comments - number of comments text_ptr - array of png_text holding image comments text_ptr[i].compression - type of compression used on "text" PNG_TEXT_COMPRESSION_NONE PNG_TEXT_COMPRESSION_zTXt PNG_ITXT_COMPRESSION_NONE PNG_ITXT_COMPRESSION_zTXt text_ptr[i].key - keyword for comment. Must contain 1-79 characters. text_ptr[i].text - text comments for current keyword. Can be empty. text_ptr[i].text_length - length of text string, after decompression, 0 for iTXt text_ptr[i].itxt_length - length of itxt string, after decompression, 0 for tEXt/zTXt text_ptr[i].lang - language of comment (empty string for unknown). text_ptr[i].translated_keyword - keyword in UTF-8 (empty string for unknown). num_text - number of comments (same as num_comments; you can put NULL here to avoid the duplication) Note while png_set_text() will accept text, language, and translated keywords that can be NULL pointers, the structure returned by png_get_text will always contain regular zero-terminated C strings. They might be empty strings but they will never be NULL pointers.

num_spalettes = png_get_sPLT(png_ptr, info_ptr, &palette_ptr); palette_ptr - array of palette structures holding contents of one or more sPLT chunks read. num_spalettes - number of sPLT chunks read.

png_get_oFFs(png_ptr, info_ptr, &offset_x, &offset_y, &unit_type); offset_x - positive offset from the left edge of the screen offset_y - positive offset from the top edge of the screen unit_type - PNG_OFFSET_PIXEL, PNG_OFFSET_MICROMETER

png_get_pHYs(png_ptr, info_ptr, &res_x, &res_y, &unit_type); res_x - pixels/unit physical resolution in x direction res_y - pixels/unit physical resolution in x direction unit_type - PNG_RESOLUTION_UNKNOWN, PNG_RESOLUTION_METER

png_get_sCAL(png_ptr, info_ptr, &unit, &width, &height) unit - physical scale units (an integer) width - width of a pixel in physical scale units height - height of a pixel in physical scale units (width and height are doubles)

png_get_sCAL_s(png_ptr, info_ptr, &unit, &width, &height) unit - physical scale units (an integer) width - width of a pixel in physical scale units height - height of a pixel in physical scale units (width and height are strings like "2.54")

num_unknown_chunks = png_get_unknown_chunks(png_ptr, info_ptr, &unknowns) unknowns - array of png_unknown_chunk structures holding unknown chunks unknowns[i].name - name of unknown chunk unknowns[i].data - data of unknown chunk unknowns[i].size - size of unknown chunk's data unknowns[i].location - position of chunk in file

The value of "i" corresponds to the order in which the chunks were read from the PNG file or inserted with the png_set_unknown_chunks() function.

The data from the pHYs chunk can be retrieved in several convenient forms:

res_x = png_get_x_pixels_per_meter(png_ptr, info_ptr) res_y = png_get_y_pixels_per_meter(png_ptr, info_ptr) res_x_and_y = png_get_pixels_per_meter(png_ptr, info_ptr) res_x = png_get_x_pixels_per_inch(png_ptr, info_ptr) res_y = png_get_y_pixels_per_inch(png_ptr, info_ptr) res_x_and_y

= png_get_pixels_per_inch(png_ptr, info_ptr) aspect_ratio = png_get_pixel_aspect_ratio(png_ptr, info_ptr)

(Each of these returns 0 [signifying "unknown"] if the data is not present or if res_x is 0; res_x_and_y is 0 if res_x != res_y)

The data from the oFFs chunk can be retrieved in several convenient forms:

x_offset = png_get_x_offset_microns(png_ptr, info_ptr); y_offset = png_get_y_offset_microns(png_ptr, info_ptr); x_offset = png_get_x_offset_inches(png_ptr, info_ptr); y_offset = png_get_y_offset_inches(png_ptr, info_ptr);

(Each of these returns 0 [signifying "unknown" if both x and y are 0] if the data is not present or if the chunk is present but the unit is the pixel)

For more information, see the png_info definition in png.h and the PNG specification for chunk contents. Be careful with trusting rowbytes, as some of the transformations could increase the space needed to hold a row (expand, filler, gray_to_rgb, etc.). See png_read_update_info(), below.

A quick word about text_ptr and num_text. PNG stores comments in keyword/text pairs, one pair per chunk, with no limit on the number of text chunks, and a 2^31 byte limit on their size. While there are suggested keywords, there is no requirement to restrict the use to these strings. It is strongly suggested that keywords and text be sensible to humans (that's the point), so don't use abbreviations. Non-printing symbols are not allowed. See the PNG specification for more details. There is also no requirement to have text after the keyword.

Keywords should be limited to 79 Latin-1 characters without leading or trailing spaces, but non-consecutive spaces are allowed within the keyword. It is possible to have the same keyword any number of times. The text_ptr is an array of png_text structures, each holding a pointer to a language string, a pointer to a keyword and a pointer to a text string. The text string, language code, and translated keyword may be empty or NULL pointers. The keyword/text pairs are put into the array in the order that they are received. However, some or all of the text chunks may be after the image, so, to make sure you have read all the text chunks, don't mess with these until after you read the stuff after the image. This will be mentioned again below in the discussion that goes with png_read_end().

After you've read the header information, you can set up the library to handle any special transformations of the image data. The various ways to transform the data will be described in the order that they should occur. This is important, as some of these change the color type and/or bit depth of the data, and some others only work on certain color types and bit depths. Even though each transformation checks to see if it has data that it can do something with, you should make sure to only enable a transformation if it will be valid for the data. For example, don't swap red and blue on grayscale data.

The colors used for the background and transparency values should be supplied in the same format/depth as the current image data. They are stored in the same format/depth as the image data in a bKGD or tRNS chunk, so this is what libpng expects for this data. The colors are transformed to keep in sync with the image data when an application calls the png_read_update_info() routine (see below).

Data will be decoded into the supplied row buffers packed into bytes unless the library has been told to transform it into another format. For example, 4 bit/pixel paletted or grayscale data will be returned 2 pixels/byte with the leftmost pixel in the high-order bits of the byte, unless png_set_packing() is called. 8-bit RGB data will be stored in RGB RGB RGB format unless png_set_filler() is called to insert filler bytes, either before or after each RGB triplet. 16-bit RGB data will be returned RRGGBB RRGGBB, with the most significant byte of the color value first, unless png_set_strip_16() is called to transform it to regular RGB RGB triplets, or png_set_filler() is called to insert filler bytes, either before or after each RRGGBB

triplet. Similarly, 8-bit or 16-bit grayscale data can be modified with png_set_filler() or png_set_strip_16().

The following code transforms grayscale images of less than 8 to 8 bits, changes paletted images to RGB, and adds a full alpha channel if there is transparency information in a tRNS chunk. This is most useful on grayscale images with bit depths of 2 or 4 or if there is a multiple-image viewing application that wishes to treat all images in the same way.

```
if (color_type == PNG_COLOR_TYPE_PALETTE) png_set_palette_to_rgb(png_ptr);

if (color_type == PNG_COLOR_TYPE_GRAY && bit_depth < 8) png_set_gray_1_2_4_to_8(png_ptr);

if (png_get_valid(png_ptr, info_ptr, PNG_INFO_tRNS)) png_set_tRNS_to_alpha(png_ptr);
```

These three functions are actually aliases for png_set_expand(), added in libpng version 1.0.4, with the function names expanded to improve code readability. In some future version they may actually do different things.

PNG can have files with 16 bits per channel. If you only can handle 8 bits per channel, this will strip the pixels down to 8 bit.

```
if (bit_depth == 16) png_set_strip_16(png_ptr);
```

If, for some reason, you don't need the alpha channel on an image, and you want to remove it rather than combining it with the background (but the image author certainly had in mind that you *would* combine it with the background, so that's what you should probably do):

```
if (color_type & PNG_COLOR_MASK_ALPHA) png_set_strip_alpha(png_ptr);
```

In PNG files, the alpha channel in an image is the level of opacity. If you need the alpha channel in an image to be the level of transparency instead of opacity, you can invert the alpha channel (or the tRNS chunk data) after it's read, so that 0 is fully opaque and 255 (in 8-bit or paletted images) or 65535 (in 16-bit images) is fully transparent, with

```
png_set_invert_alpha(png_ptr);
```

PNG files pack pixels of bit depths 1, 2, and 4 into bytes as small as they can, resulting in, for example, 8 pixels per byte for 1 bit files. This code expands to 1 pixel per byte without changing the values of the pixels:

```
if (bit_depth < 8) png_set_packing(png_ptr);
```

PNG files have possible bit depths of 1, 2, 4, 8, and 16. All pixels stored in a PNG image have been "scaled" or "shifted" up to the next higher possible bit depth (e.g. from 5 bits/sample in the range [0,31] to 8 bits/sample in the range [0, 255]). However, it is also possible to convert the PNG pixel data back to the original bit depth of the image. This call reduces the pixels back down to the original bit depth:

```
png_color_16p sig_bit;

if (png_get_sBIT(png_ptr, info_ptr, &sig_bit)) png_set_shift(png_ptr, sig_bit);
```

PNG files store 3-color pixels in red, green, blue order. This code changes the storage of the pixels to blue, green, red:

```
if (color_type == PNG_COLOR_TYPE_RGB || color_type == PNG_COLOR_TYPE_RGB_ALPHA) png_set_bgr(png_ptr);
```

PNG files store RGB pixels packed into 3 bytes. This code expands them into 4 bytes for windowing systems that need them in this format:

```
if (bit_depth == 8 && color_type == PNG_COLOR_TYPE_RGB) png_set_filler(png_ptr, filler, PNG_FILLER_BEFORE);
```

where "filler" is the 8 or 16-bit number to fill with, and the location is either PNG_FILLER_BEFORE or PNG_FILLER_AFTER, depending upon whether you

want the filler before the RGB or after. This transformation does not affect images that already have full alpha channels.

If you are reading an image with an alpha channel, and you need the data as ARGB instead of the normal PNG format RGBA:

if (color_type == PNG_COLOR_TYPE_RGB_ALPHA) png_set_swap_alpha(png_ptr);

For some uses, you may want a grayscale image to be represented as RGB. This code will do that conversion:

if (color_type == PNG_COLOR_TYPE_GRAY || color_type == PNG_COLOR_TYPE_GRAY_ALPHA) png_set_gray_to_rgb(png_ptr);

Conversely, you can convert an RGB or RGBA image to grayscale or grayscale with alpha.

if (color_type == PNG_COLOR_TYPE_RGB || color_type == PNG_COLOR_TYPE_RGB_ALPHA) png_set_rgb_to_gray_fixed(png_ptr, error_action, int red_weight, int green_weight);

error_action = 1: silently do the conversion error_action = 2: issue a warning if the original image has any pixel where red != green or red != blue error_action = 3: issue an error and abort the conversion if the original image has any pixel where red != green or red != blue

red_weight: weight of red component times 100000 green_weight: weight of green component times 100000 If either weight is negative, default weights (21268, 71514) are used.

If you have set error_action = 1 or 2, you can later check whether the image really was gray, after processing the image rows, with the png_get_rgb_to_gray_status(png_ptr) function. It will return a png_byte that is zero if the image was gray or 1 if there were any non-gray pixels. bKGD and sBIT data will be silently converted to grayscale, using the green channel data, regardless of the error_action setting.

With red_weight+green_weight<=100000, the normalized graylevel is computed:

int rw = red_weight * 65536; int gw = green_weight * 65536; int bw = 65536 - (rw + gw); gray = (rw*red + gw*green + bw*blue)/65536;

The default values approximate those recommended in the Charles Poynton's Color FAQ, <http://www.inforamp.net/~poynton/> Copyright (c) 1998-01-04 Charles Poynton poynton@inforamp.net

Y = 0.212671 * R + 0.715160 * G + 0.072169 * B

Libpng approximates this with

Y = 0.21268 * R + 0.7151 * G + 0.07217 * B

which can be expressed with integers as

Y = (6969 * R + 23434 * G + 2365 * B)/32768

The calculation is done in a linear colorspace, if the image gamma is known.

If you have a grayscale and you are using png_set_expand_depth() or png_set_expand() to change to a higher bit-depth, you must either supply the background color as a gray value at the original file bit-depth (need_expand = 1) or else supply the background color as an RGB triplet at the final, expanded bit depth (need_expand = 0). Similarly, if you are reading a paletted image, you must either supply the background color as a palette index (need_expand = 1) or as an RGB triplet that may or may not be in the palette (need_expand = 0).

png_color_16 my_background; png_color_16p image_background;

if (png_get_bKGD(png_ptr, info_ptr, &image_background)) png_set_background(png_ptr, image_background,

PNG_BACKGROUND_GAMMA_FILE, 1, 1.0); else png_set_background(png_ptr, &my_background, PNG_BACKGROUND_GAMMA_SCREEN, 0, 1.0);

The png_set_background() function tells libpng to composite images with alpha or simple transparency against the supplied background color. If the PNG file contains a bKGD chunk (PNG_INFO_bKGD valid), you may use this color, or supply another color more suitable for the current display (e.g., the background color from a web page). You need to tell libpng whether the color is in the gamma space of the display (PNG_BACKGROUND_GAMMA_SCREEN for colors you supply), the file (PNG_BACKGROUND_GAMMA_FILE for colors from the bKGD chunk), or one that is neither of these gammas (PNG_BACKGROUND_GAMMA_UNIQUE - I don't know why anyone would use this, but it's here).

To properly display PNG images on any kind of system, the application needs to know what the display gamma is. Ideally, the user will know this, and the application will allow them to set it. One method of allowing the user to set the display gamma separately for each system is to check for a SCREEN_GAMMA or DISPLAY_GAMMA environment variable, which will hopefully be correctly set.

Note that display_gamma is the overall gamma correction required to produce pleasing results, which depends on the lighting conditions in the surrounding environment. In a dim or brightly lit room, no compensation other than the physical gamma exponent of the monitor is needed, while in a dark room a slightly smaller exponent is better.

double gamma, screen_gamma;

if (/* We have a user-defined screen gamma value */) { screen_gamma = user_defined_screen_gamma; } /* One way that applications can share the same screen gamma value */ else if ((gamma_str = getenv("SCREEN_GAMMA")) != NULL) { screen_gamma = (double)atof(gamma_str); } /* If we don't have another value */ else { screen_gamma = 2.2; /* A good guess for a PC monitor in a bright office or a dim room */ screen_gamma = 2.0; /* A good guess for a PC monitor in a dark room */ screen_gamma = 1.7 or 1.0; /* A good guess for Mac systems */ }

The png_set_gamma() function handles gamma transformations of the data. Pass both the file gamma and the current screen_gamma. If the file does not have a gamma value, you can pass one anyway if you have an idea what it is (usually 0.45455 is a good guess for GIF images on PCs). Note that file gammas are inverted from screen gammas. See the discussions on gamma in the PNG specification for an excellent description of what gamma is, and why all applications should support it. It is strongly recommended that PNG viewers support gamma correction.

if (png_get_gAMA(png_ptr, info_ptr, &gamma)) png_set_gamma(png_ptr, screen_gamma, gamma); else png_set_gamma(png_ptr, screen_gamma, 0.45455);

If you need to reduce an RGB file to a paletted file, or if a paletted file has more entries then will fit on your screen, png_set_dither() will do that. Note that this is a simple match dither that merely finds the closest color available. This should work fairly well with optimized palettes, and fairly badly with linear color cubes. If you pass a palette that is larger then maximum_colors, the file will reduce the number of colors in the palette so it will fit into maximum_colors. If there is a histogram, it will use it to make more intelligent choices when reducing the palette. If there is no histogram, it may not do as good a job.

if (color_type & PNG_COLOR_MASK_COLOR) { if (png_get_valid(png_ptr, info_ptr, PNG_INFO_PLTE)) { png_uint_16p histogram;

png_get_hIST(png_ptr, info_ptr, &histogram); png_set_dither(png_ptr, palette, num_palette, max_screen_colors, histogram, 1); } else { png_color std_color_cube[MAX_SCREEN_COLORS] = { ... colors ... };

png_set_dither(png_ptr, std_color_cube, MAX_SCREEN_COLORS, MAX_SCREEN_COLORS, NULL,0); } }

PNG files describe monochrome as black being zero and white being one. The following code will reverse this (make black be one and white be zero):

if (bit_depth == 1 && color_type == PNG_COLOR_GRAY) png_set_invert_mono(png_ptr);

PNG files store 16 bit pixels in network byte order (big-endian, ie. most significant bits first). This code changes the storage to the other way (little-endian, i.e. least significant bits first, the way PCs store them):

if (bit_depth == 16) png_set_swap(png_ptr);

If you are using packed-pixel images (1, 2, or 4 bits/pixel), and you need to change the order the pixels are packed into bytes, you can use:

if (bit_depth < 8) png_set_packswap(png_ptr);

Finally, you can write your own transformation function if none of the existing ones meets your needs. This is done by setting a callback with

png_set_read_user_transform_fn(png_ptr, read_transform_fn);

You must supply the function

void read_transform_fn(png_ptr ptr, row_info_ptr row_info, png_bytep data)

See pngtest.c for a working example. Your function will be called after all of the other transformations have been processed.

You can also set up a pointer to a user structure for use by your callback function, and you can inform libpng that your transform function will change the number of channels or bit depth with the function

png_set_user_transform_info(png_ptr, user_ptr, user_depth, user_channels);

The user's application, not libpng, is responsible for allocating and freeing any memory required for the user structure.

You can retrieve the pointer via the function png_get_user_transform_ptr(). For example:

voidp read_user_transform_ptr = png_get_user_transform_ptr(png_ptr);

The last thing to handle is interlacing; this is covered in detail below, but you must call the function here if you want libpng to handle expansion of the interlaced image.

number_of_passes = png_set_interlace_handling(png_ptr);

After setting the transformations, libpng can update your png_info structure to reflect any transformations you've requested with this call. This is most useful to update the info structure's rowbytes field so you can use it to allocate your image memory. This function will also update your palette with the correct screen_gamma and background if these have been given with the calls above.

png_read_update_info(png_ptr, info_ptr);

After you call png_read_update_info(), you can allocate any memory you need to hold the image. The row data is simply raw byte data for all forms of images. As the actual allocation varies among applications, no example will be given. If you are allocating one large chunk, you will need to build an array of pointers to each row, as it will be needed for some of the functions below.

After you've allocated memory, you can read the image data. The simplest way to do this is in one function call. If you are allocating enough memory to hold the whole image, you can just call png_read_image() and libpng will read in all the image data and put it in the memory area supplied. You will need to pass in an array of pointers to each row.

This function automatically handles interlacing, so you don't need to call png_set_interlace_handling() or call this function multiple times, or any of that other stuff necessary with png_read_rows().

png_read_image(png_ptr, row_pointers);

where row_pointers is:

png_bytep row_pointers[height];

You can point to void or char or whatever you use for pixels.

If you don't want to read in the whole image at once, you can use png_read_rows() instead. If there is no interlacing (check interlace_type == PNG_INTERLACE_NONE), this is simple:

png_read_rows(png_ptr, row_pointers, NULL, number_of_rows);

where row_pointers is the same as in the png_read_image() call.

If you are doing this just one row at a time, you can do this with a single row_pointer instead of an array of row_pointers:

png_bytep row_pointer = row; png_read_row(png_ptr, row_pointer, NULL);

If the file is interlaced (interlace_type != 0 in the IHDR chunk), things get somewhat harder. The only current (PNG Specification version 1.2) interlacing type for PNG is (interlace_type == PNG_INTERLACE_ADAM7) is a somewhat complicated 2D interlace scheme, known as Adam7, that breaks down an image into seven smaller images of varying size, based on an 8x8 grid.

libpng can fill out those images or it can give them to you "as is". If you want them filled out, there are two ways to do that. The one mentioned in the PNG specification is to expand each pixel to cover those pixels that have not been read yet (the "rectangle" method). This results in a blocky image for the first pass, which gradually smooths out as more pixels are read. The other method is the "sparkle" method, where pixels are drawn only in their final locations, with the rest of the image remaining whatever colors they were initialized to before the start of the read. The first method usually looks better, but tends to be slower, as there are more pixels to put in the rows.

If you don't want libpng to handle the interlacing details, just call png_read_rows() seven times to read in all seven images. Each of the images is a valid image by itself, or they can all be combined on an 8x8 grid to form a single image (although if you intend to combine them you would be far better off using the libpng interlace handling).

The first pass will return an image 1/8 as wide as the entire image (every 8th column starting in column 0) and 1/8 as high as the original (every 8th row starting in row 0), the second will be 1/8 as wide (starting in column 4) and 1/8 as high (also starting in row 0). The third pass will be 1/4 as wide (every 4th pixel starting in column 0) and 1/8 as high (every 8th row starting in row 4), and the fourth pass will be 1/4 as wide and 1/4 as high (every 4th column starting in column 2, and every 4th row starting in row 0). The fifth pass will return an image 1/2 as wide, and 1/4 as high (starting at column 0 and row 2), while the sixth pass will be 1/2 as wide and 1/2 as high as the original (starting in column 1 and row 0). The seventh and final pass will be as wide as the original, and 1/2 as high, containing all of the odd numbered scanlines. Phew!

If you want libpng to expand the images, call this before calling png_start_read_image() or png_read_update_info():

if (interlace_type == PNG_INTERLACE_ADAM7) number_of_passes = png_set_interlace_handling(png_ptr);

This will return the number of passes needed. Currently, this is seven, but may change if another interlace type is added. This function can be called even if the file is not interlaced, where it will return one pass.

If you are not going to display the image after each pass, but are going to wait until the entire image is read in, use the sparkle effect. This effect is faster and the end result of either method is exactly the same. If you are planning on displaying the

image after each pass, the "rectangle" effect is generally considered the better looking one.

If you only want the "sparkle" effect, just call png_read_rows() as normal, with the third parameter NULL. Make sure you make pass over the image number_of_passes times, and you don't change the data in the rows between calls. You can change the locations of the data, just not the data. Each pass only writes the pixels appropriate for that pass, and assumes the data from previous passes is still valid.

png_read_rows(png_ptr, row_pointers, NULL, number_of_rows);

If you only want the first effect (the rectangles), do the same as before except pass the row buffer in the third parameter, and leave the second parameter NULL.

png_read_rows(png_ptr, NULL, row_pointers, number_of_rows);

After you are finished reading the image through either the high- or low-level interfaces, you can finish reading the file. If you are interested in comments or time, which may be stored either before or after the image data, you should pass the separate png_info struct if you want to keep the comments from before and after the image separate. If you are not interested, you can pass NULL.

png_read_end(png_ptr, end_info);

When you are done, you can free all memory allocated by libpng like this:

png_destroy_read_struct(&png_ptr, &info_ptr, &end_info);

It is also possible to individually free the info_ptr members that point to libpng-allocated storage with the following function:

png_free_data(png_ptr, info_ptr, mask, n) mask - identifies data to be freed, a mask containing the logical OR of one or more of PNG_FREE_PLTE, PNG_FREE_TRNS, PNG_FREE_HIST, PNG_FREE_ICCP, PNG_FREE_PCAL, PNG_FREE_ROWS, PNG_FREE_SCAL, PNG_FREE_SPLT, PNG_FREE_TEXT, PNG_FREE_UNKN, or simply PNG_FREE_ALL n - sequence number of item to be freed (-1 for all items)

This function may be safely called when the relevant storage has already been freed, or has not yet been allocated, or was allocated by the user and not by libpng, and will in those cases do nothing. The "n" parameter is ignored if only one item of the selected data type, such as PLTE, is allowed. If "n" is not -1, and multiple items are allowed for the data type identified in the mask, such as text or sPLT, only the n'th item is freed.

The default behavior is only to free data that was allocated internally by libpng. This can be changed, so that libpng will not free the data, or so that it will free data that was allocated by the user with png_malloc() or png_zalloc() and passed in via a png_set_*() function, with

png_data_freer(png_ptr, info_ptr, freer, mask) mask - which data elements are affected same choices as in png_free_data() freer - one of PNG_DESTROY_WILL_FREE_DATA PNG_SET_WILL_FREE_DATA PNG_USER_WILL_FREE_DATA

This function only affects data that has already been allocated. You can call this function after reading the PNG data but before calling any png_set_*() functions, to control whether the user or the png_set_*() function is responsible for freeing any existing data that might be present, and again after the png_set_*() functions to control whether the user or png_destroy_*() is supposed to free the data. When the user assumes responsibility for libpng-allocated data, the application must use png_free() to free it, and when the user transfers responsibility to libpng for data that the user has allocated, the user must have used png_malloc() or png_zalloc() to allocate it (the png_zalloc() function is the same as png_malloc() except that it also zeroes the newly-allocated memory).

If you allocated your row_pointers in a single block, as suggested above in the description of the high level read interface, you must not transfer responsibility for free-

ing it to the png_set_rows or png_read_destroy function, because they would also try to free the individual row_pointers[i].

If you allocated text_ptr.text, text_ptr.lang, and text_ptr.translated_keyword separately, do not transfer responsibility for freeing text_ptr to libpng, because when libpng fills a png_text structure it combines these members with the key member, and png_free_data() will free only text_ptr.key. Similarly, if you transfer responsibility for free'ing text_ptr from libpng to your application, your application must not separately free those members.

The png_free_data() function will turn off the "valid" flag for anything it frees. If you need to turn the flag off for a chunk that was freed by your application instead of by libpng, you can use

png_set_invalid(png_ptr, info_ptr, mask); mask - identifies the chunks to be made invalid, containing the logical OR of one or more of PNG_INFO_gAMA, PNG_INFO_sBIT, PNG_INFO_cHRM, PNG_INFO_PLTE, PNG_INFO_tRNS, PNG_INFO_bKGD, PNG_INFO_hIST, PNG_INFO_pHYs, PNG_INFO_oFFs, PNG_INFO_tIME, PNG_INFO_pCAL, PNG_INFO_sRGB, PNG_INFO_iCCP, PNG_INFO_sPLT, PNG_INFO_sCAL, PNG_INFO_IDAT

For a more compact example of reading a PNG image, see the file example.c.

The progressive reader is slightly different then the non-progressive reader. Instead of calling png_read_info(), png_read_rows(), and png_read_end(), you make one call to png_process_data(), which calls callbacks when it has the info, a row, or the end of the image. You set up these callbacks with png_set_progressive_read_fn(). You don't have to worry about the input/output functions of libpng, as you are giving the library the data directly in png_process_data(). I will assume that you have read the section on reading PNG files above, so I will only highlight the differences (although I will show all of the code).

png_structp png_ptr; png_infop info_ptr;

/* An example code fragment of how you would initialize the progressive reader in your application. */ int initialize_png_reader() { png_ptr = png_create_read_struct (PNG_LIBPNG_VER_STRING, (png_voidp)user_error_ptr, user_error_fn, user_warning_fn); if (!png_ptr) return (ERROR); info_ptr = png_create_info_struct(png_ptr); if (!info_ptr) { png_destroy_read_struct(&png_ptr, (png_infopp)NULL, (png_infopp)NULL); return (ERROR); }

if (setjmp(png_jmpbuf(png_ptr))) { png_destroy_read_struct(&png_ptr, &info_ptr, (png_infopp)NULL); return (ERROR); }

/* This one's new. You can provide functions to be called when the header info is valid, when each row is completed, and when the image is finished. If you aren't using all functions, you can specify NULL parameters. Even when all three functions are NULL, you need to call png_set_progressive_read_fn(). You can use any struct as the user_ptr (cast to a void pointer for the function call), and retrieve the pointer from inside the callbacks using the function

png_get_progressive_ptr(png_ptr);

which will return a void pointer, which you have to cast appropriately. */ png_set_progressive_read_fn(png_ptr, (void *)user_ptr, info_callback, row_callback, end_callback);

return 0; }

/* A code fragment that you call as you receive blocks of data */ int process_data(png_bytep buffer, png_uint_32 length) { if (setjmp(png_jmpbuf(png_ptr))) { png_destroy_read_struct(&png_ptr, &info_ptr, (png_infopp)NULL); return (ERROR); }

/* This one's new also. Simply give it a chunk of data from the file stream (in order, of course). On machines with segmented memory models machines, don't give it any more than 64K. The library seems to run fine with sizes of 4K. Although you can

give it much less if necessary (I assume you can give it chunks of 1 byte, I haven't tried less then 256 bytes yet). When this function returns, you may want to display any rows that were generated in the row callback if you don't already do so there. */ png_process_data(png_ptr, info_ptr, buffer, length); return 0; }

/* This function is called (as set by png_set_progressive_read_fn() above) when enough data has been supplied so all of the header has been read. */ void info_callback(png_structp png_ptr, png_infop info) { /* Do any setup here, including setting any of the transformations mentioned in the Reading PNG files section. For now, you _must_ call either png_start_read_image() or png_read_update_info() after all the transformations are set (even if you don't set any). You may start getting rows before png_process_data() returns, so this is your last chance to prepare for that. */ }

/* This function is called when each row of image data is complete */ void row_callback(png_structp png_ptr, png_bytep new_row, png_uint_32 row_num, int pass) { /* If the image is interlaced, and you turned on the interlace handler, this function will be called for every row in every pass. Some of these rows will not be changed from the previous pass. When the row is not changed, the new_row variable will be NULL. The rows and passes are called in order, so you don't really need the row_num and pass, but I'm supplying them because it may make your life easier.

For the non-NULL rows of interlaced images, you must call png_progressive_combine_row() passing in the row and the old row. You can call this function for NULL rows (it will just return) and for non-interlaced images (it just does the memcpy for you) if it will make the code easier. Thus, you can just do this for all cases: */

png_progressive_combine_row(png_ptr, old_row, new_row);

/* where old_row is what was displayed for previously for the row. Note that the first pass (pass == 0, really) will completely cover the old row, so the rows do not have to be initialized. After the first pass (and only for interlaced images), you will have to pass the current row, and the function will combine the old row and the new row. */ }

void end_callback(png_structp png_ptr, png_infop info) { /* This function is called after the whole image has been read, including any chunks after the image (up to and including the IEND). You will usually have the same info chunk as you had in the header, although some data may have been added to the comments and time fields.

Most people won't do much here, perhaps setting a flag that marks the image as finished. */ }

IV. Writing

Much of this is very similar to reading. However, everything of importance is repeated here, so you won't have to constantly look back up in the reading section to understand writing.

You will want to do the I/O initialization before you get into libpng, so if it doesn't work, you don't have anything to undo. If you are not using the standard I/O functions, you will need to replace them with custom writing functions. See the discussion under Customizing libpng.

FILE *fp = fopen(file_name, "wb"); if (!fp) { return (ERROR); }

Next, png_struct and png_info need to be allocated and initialized. As these can be both relatively large, you may not want to store these on the stack, unless you have stack space to spare. Of course, you will want to check if they return NULL. If you are also reading, you won't want to name your read structure and your write struc-

ture both "png_ptr"; you can call them anything you like, such as "read_ptr" and "write_ptr". Look at pngtest.c, for example.

png_structp png_ptr = png_create_write_struct (PNG_LIBPNG_VER_STRING, (png_voidp)user_error_ptr, user_error_fn, user_warning_fn); if (!png_ptr) return (ERROR);

png_infop info_ptr = png_create_info_struct(png_ptr); if (!info_ptr) { png_destroy_write_struct(&png_ptr, (png_infopp)NULL); return (ERROR); }

If you want to use your own memory allocation routines, define PNG_USER_MEM_SUPPORTED and use png_create_write_struct_2() instead of png_create_write_struct():

png_structp png_ptr = png_create_write_struct_2 (PNG_LIBPNG_VER_STRING, (png_voidp)user_error_ptr, user_error_fn, user_warning_fn, (png_voidp) user_mem_ptr, user_malloc_fn, user_free_fn);

After you have these structures, you will need to set up the error handling. When libpng encounters an error, it expects to longjmp() back to your routine. Therefore, you will need to call setjmp() and pass the png_jmpbuf(png_ptr). If you write the file from different routines, you will need to update the png_jmpbuf(png_ptr) every time you enter a new routine that will call a png_*() function. See your documentation of setjmp/longjmp for your compiler for more information on setjmp/longjmp. See the discussion on libpng error handling in the Customizing Libpng section below for more information on the libpng error handling.

if (setjmp(png_jmpbuf(png_ptr))) { png_destroy_write_struct(&png_ptr, &info_ptr); fclose(fp); return (ERROR); } ... return;

If you would rather avoid the complexity of setjmp/longjmp issues, you can compile libpng with PNG_SETJMP_NOT_SUPPORTED, in which case errors will result in a call to PNG_ABORT() which defaults to abort().

Now you need to set up the output code. The default for libpng is to use the C function fwrite(). If you use this, you will need to pass a valid FILE * in the function png_init_io(). Be sure that the file is opened in binary mode. Again, if you wish to handle writing data in another way, see the discussion on libpng I/O handling in the Customizing Libpng section below.

png_init_io(png_ptr, fp);

At this point, you can set up a callback function that will be called after each row has been written, which you can use to control a progress meter or the like. It's demonstrated in pngtest.c. You must supply a function

void write_row_callback(png_ptr, png_uint_32 row, int pass); { /* put your code here */ }

(You can give it another name that you like instead of "write_row_callback")

To inform libpng about your function, use

png_set_write_status_fn(png_ptr, write_row_callback);

You now have the option of modifying how the compression library will run. The following functions are mainly for testing, but may be useful in some cases, like if you need to write PNG files extremely fast and are willing to give up some compression, or if you want to get the maximum possible compression at the expense of slower writing. If you have no special needs in this area, let the library do what it wants by not calling this function at all, as it has been tuned to deliver a good speed/compression ratio. The second parameter to png_set_filter() is the filter method, for which the only valid values are 0 (as of the July 1999 PNG specification, version 1.2) or 64 (if you are writing a PNG datastream that is to be embedded in a MNG datastream). The third parameter is a flag that indicates which filter type(s) are to be tested for each scanline. See the PNG specification for details on the specific filter types.

/* turn on or off filtering, and/or choose specific filters. You can use either a single PNG_FILTER_VALUE_NAME or the logical OR of one or more PNG_FILTER_NAME masks. */ png_set_filter(png_ptr, 0, PNG_FILTER_NONE | PNG_FILTER_VALUE_NONE | PNG_FILTER_SUB | PNG_FILTER_VALUE_SUB | PNG_FILTER_UP | PNG_FILTER_VALUE_UP | PNG_FILTER_AVE | PNG_FILTER_VALUE_AVE | PNG_FILTER_PAETH | PNG_FILTER_VALUE_PAETH| PNG_ALL_FILTERS);

If an application wants to start and stop using particular filters during compression, it should start out with all of the filters (to ensure that the previous row of pixels will be stored in case it's needed later), and then add and remove them after the start of compression.

If you are writing a PNG datastream that is to be embedded in a MNG datastream, the second parameter can be either 0 or 64.

The png_set_compression_*() functions interface to the zlib compression library, and should mostly be ignored unless you really know what you are doing. The only generally useful call is png_set_compression_level() which changes how much time zlib spends on trying to compress the image data. See the Compression Library (zlib.h and algorithm.txt, distributed with zlib) for details on the compression levels.

/* set the zlib compression level */ png_set_compression_level(png_ptr, Z_BEST_COMPRESSION);

/* set other zlib parameters */ png_set_compression_mem_level(png_ptr, 8); png_set_compression_strategy(png_ptr, Z_DEFAULT_STRATEGY); png_set_compression_window_bits(png_ptr, 15); png_set_compression_method(png_ptr, 8); png_set_compression_buffer_size(png_ptr, 8192)

extern PNG_EXPORT(void,png_set_zbuf_size)

You now need to fill in the png_info structure with all the data you wish to write before the actual image. Note that the only thing you are allowed to write after the image is the text chunks and the time chunk (as of PNG Specification 1.2, anyway). See png_write_end() and the latest PNG specification for more information on that. If you wish to write them before the image, fill them in now, and flag that data as being valid. If you want to wait until after the data, don't fill them until png_write_end(). For all the fields in png_info and their data types, see png.h. For explanations of what the fields contain, see the PNG specification.

Some of the more important parts of the png_info are:

png_set_IHDR(png_ptr, info_ptr, width, height, bit_depth, color_type, interlace_type, compression_type, filter_method) width - holds the width of the image in pixels (up to 2^31). height - holds the height of the image in pixels (up to 2^31). bit_depth - holds the bit depth of one of the image channels. (valid values are 1, 2, 4, 8, 16 and depend also on the color_type. See also significant bits (sBIT) below). color_type - describes which color/alpha channels are present. PNG_COLOR_TYPE_GRAY (bit depths 1, 2, 4, 8, 16) PNG_COLOR_TYPE_GRAY_ALPHA (bit depths 8, 16) PNG_COLOR_TYPE_PALETTE (bit depths 1, 2, 4, 8) PNG_COLOR_TYPE_RGB (bit_depths 8, 16) PNG_COLOR_TYPE_RGB_ALPHA (bit_depths 8, 16)

PNG_COLOR_MASK_PALETTE PNG_COLOR_MASK_COLOR PNG_COLOR_MASK_ALPHA

interlace_type - PNG_INTERLACE_NONE or PNG_INTERLACE_ADAM7 compression_type - (must be PNG_COMPRESSION_TYPE_DEFAULT) filter_method - (must be PNG_FILTER_TYPE_DEFAULT or, if you are writing a PNG to be embedded in a MNG datastream, can also be PNG_INTRAPIXEL_DIFFERENCING)

png_set_PLTE(png_ptr, info_ptr, palette, num_palette); palette - the palette for the file (array of png_color) num_palette - number of entries in the palette

png_set_gAMA(png_ptr, info_ptr, gamma); gamma - the gamma the image was created at (PNG_INFO_gAMA)

png_set_sRGB(png_ptr, info_ptr, srgb_intent); srgb_intent - the rendering intent (PNG_INFO_sRGB) The presence of the sRGB chunk means that the pixel data is in the sRGB color space. This chunk also implies specific values of gAMA and cHRM. Rendering intent is the CSS-1 property that has been defined by the International Color Consortium (http://www.color.org). It can be one of PNG_sRGB_INTENT_SATURATION, PNG_sRGB_INTENT_PERCEPTUAL, PNG_sRGB_INTENT_ABSOLUTE, or PNG_sRGB_INTENT_RELATIVE.

png_set_sRGB_gAMA_and_cHRM(png_ptr, info_ptr, srgb_intent); srgb_intent - the rendering intent (PNG_INFO_sRGB) The presence of the sRGB chunk means that the pixel data is in the sRGB color space. This function also causes gAMA and cHRM chunks with the specific values that are consistent with sRGB to be written.

png_set_iCCP(png_ptr, info_ptr, name, compression_type, profile, proflen); name - The profile name. compression - The compression type; always PNG_COMPRESSION_TYPE_BASE for PNG 1.0. You may give NULL to this argument to ignore it. profile - International Color Consortium color profile data. May contain NULs. proflen - length of profile data in bytes.

png_set_sBIT(png_ptr, info_ptr, sig_bit); sig_bit - the number of significant bits for (PNG_INFO_sBIT) each of the gray, red, green, and blue channels, whichever are appropriate for the given color type (png_color_16)

png_set_tRNS(png_ptr, info_ptr, trans, num_trans, trans_values); trans - array of transparent entries for palette (PNG_INFO_tRNS) trans_values - graylevel or color sample values of the single transparent color for non-paletted images (PNG_INFO_tRNS) num_trans - number of transparent entries (PNG_INFO_tRNS)

png_set_hIST(png_ptr, info_ptr, hist); (PNG_INFO_hIST) hist - histogram of palette (array of png_uint_16)

png_set_tIME(png_ptr, info_ptr, mod_time); mod_time - time image was last modified (PNG_VALID_tIME)

png_set_bKGD(png_ptr, info_ptr, background); background - background color (PNG_VALID_bKGD)

png_set_text(png_ptr, info_ptr, text_ptr, num_text); text_ptr - array of png_text holding image comments text_ptr[i].compression - type of compression used on "text" PNG_TEXT_COMPRESSION_NONE PNG_TEXT_COMPRESSION_zTXt PNG_ITXT_COMPRESSION_NONE PNG_ITXT_COMPRESSION_zTXt text_ptr[i].key - keyword for comment. Must contain 1-79 characters. text_ptr[i].text - text comments for current keyword. Can be NULL or empty. text_ptr[i].text_length - length of text string, after decompression, 0 for iTXt text_ptr[i].itxt_length - length of itxt string, after decompression, 0 for tEXt/zTXt text_ptr[i].lang - language of comment (NULL or empty for unknown). text_ptr[i].translated_keyword - keyword in UTF-8 (NULL or empty for unknown). num_text - number of comments

png_set_sPLT(png_ptr, info_ptr, &palette_ptr, num_spalettes); palette_ptr - array of png_sPLT_struct structures to be added to the list of palettes in the info structure. num_spalettes - number of palette structures to be added.

png_set_oFFs(png_ptr, info_ptr, offset_x, offset_y, unit_type); offset_x - positive offset from the left edge of the screen offset_y - positive offset from the top edge of the screen unit_type - PNG_OFFSET_PIXEL, PNG_OFFSET_MICROMETER

png_set_pHYs(png_ptr, info_ptr, res_x, res_y, unit_type); res_x - pixels/unit physical resolution in x direction res_y - pixels/unit physical resolution in y direction unit_type - PNG_RESOLUTION_UNKNOWN, PNG_RESOLUTION_METER

png_set_sCAL(png_ptr, info_ptr, unit, width, height) unit - physical scale units (an integer) width - width of a pixel in physical scale units height - height of a pixel in physical scale units (width and height are doubles)

png_set_sCAL_s(png_ptr, info_ptr, unit, width, height) unit - physical scale units (an integer) width - width of a pixel in physical scale units height - height of a pixel in physical scale units (width and height are strings like "2.54")

png_set_unknown_chunks(png_ptr, info_ptr, &unknowns, num_unknowns) unknowns - array of png_unknown_chunk structures holding unknown chunks unknowns[i].name - name of unknown chunk unknowns[i].data - data of unknown chunk unknowns[i].size - size of unknown chunk's data unknowns[i].location - position to write chunk in file 0: do not write chunk PNG_HAVE_IHDR: before PLTE PNG_HAVE_PLTE: before IDAT PNG_AFTER_IDAT: after IDAT The "location" member is set automatically according to what part of the output file has already been written. You can change its value after calling png_set_unknown_chunks() as demonstrated in pngtest.c. Within each of the "locations", the chunks are sequenced according to their position in the structure (that is, the value of "i", which is the order in which the chunk was either read from the input file or defined with png_set_unknown_chunks).

A quick word about text and num_text. text is an array of png_text structures. num_text is the number of valid structures in the array. Each png_text structure holds a language code, a keyword, a text value, and a compression type.

The compression types have the same valid numbers as the compression types of the image data. Currently, the only valid number is zero. However, you can store text either compressed or uncompressed, unlike images, which always have to be compressed. So if you don't want the text compressed, set the compression type to PNG_TEXT_COMPRESSION_NONE. Because tEXt and zTXt chunks don't have a language field, if you specify PNG_TEXT_COMPRESSION_NONE or PNG_TEXT_COMPRESSION_zTXt any language code or translated keyword will not be written out.

Until text gets around 1000 bytes, it is not worth compressing it. After the text has been written out to the file, the compression type is set to PNG_TEXT_COMPRESSION_NONE_WR or PNG_TEXT_COMPRESSION_zTXt_WR, so that it isn't written out again at the end (in case you are calling png_write_end() with the same struct.

The keywords that are given in the PNG Specification are:

Title Short (one line) title or caption for image Author Name of image's creator Description Description of image (possibly long) Copyright Copyright notice Creation Time Time of original image creation (usually RFC 1123 format, see below) Software Software used to create the image Disclaimer Legal disclaimer Warning Warning of nature of content Source Device used to create the image Comment Miscellaneous comment; conversion from other image format

The keyword-text pairs work like this. Keywords should be short simple descriptions of what the comment is about. Some typical keywords are found in the PNG specification, as is some recommendations on keywords. You can repeat keywords in a file. You can even write some text before the image and some after. For example, you may want to put a description of the image before the image, but leave the disclaimer until after, so viewers working over modem connections don't have to wait for the disclaimer to go over the modem before they start seeing the image. Finally, keywords should be full words, not abbreviations. Keywords and text are in the ISO 8859-1 (Latin-1) character set (a superset of regular ASCII) and can not contain NUL characters, and should not contain control or other unprintable characters. To make the comments widely readable, stick with basic ASCII, and avoid machine specific character set extensions like the IBM-PC character set. The keyword must be present, but you can leave off the text string on non-compressed pairs. Compressed pairs must have a text string, as only the text string is compressed anyway, so the compression would be meaningless.

PNG supports modification time via the png_time structure. Two conversion routines are provided, png_convert_from_time_t() for time_t and png_convert_from_struct_tm() for struct tm. The time_t routine uses gmtime(). You

don't have to use either of these, but if you wish to fill in the png_time structure directly, you should provide the time in universal time (GMT) if possible instead of your local time. Note that the year number is the full year (e.g. 1998, rather than 98 - PNG is year 2000 compliant!), and that months start with 1.

If you want to store the time of the original image creation, you should use a plain tEXt chunk with the "Creation Time" keyword. This is necessary because the "creation time" of a PNG image is somewhat vague, depending on whether you mean the PNG file, the time the image was created in a non-PNG format, a still photo from which the image was scanned, or possibly the subject matter itself. In order to facilitate machine-readable dates, it is recommended that the "Creation Time" tEXt chunk use RFC 1123 format dates (e.g. "22 May 1997 18:07:10 GMT"), although this isn't a requirement. Unlike the tIME chunk, the "Creation Time" tEXt chunk is not expected to be automatically changed by the software. To facilitate the use of RFC 1123 dates, a function png_convert_to_rfc1123(png_timep) is provided to convert from PNG time to an RFC 1123 format string.

You can use the png_set_unknown_chunks function to queue up chunks for writing. You give it a chunk name, raw data, and a size; that's all there is to it. The chunks will be written by the next following png_write_info_before_PLTE, png_write_info, or png_write_end function. Any chunks previously read into the info structure's unknown-chunk list will also be written out in a sequence that satisfies the PNG specification's ordering rules.

At this point there are two ways to proceed; through the high-level write interface, or through a sequence of low-level write operations. You can use the high-level interface if your image data is present in the info structure. All defined output transformations are permitted, enabled by the following masks.

PNG_TRANSFORM_IDENTITY No transformation PNG_TRANSFORM_PACKING Pack 1, 2 and 4-bit samples PNG_TRANSFORM_PACKSWAP Change order of packed pixels to LSB first PNG_TRANSFORM_INVERT_MONO Invert monochrome images PNG_TRANSFORM_SHIFT Normalize pixels to the sBIT depth PNG_TRANSFORM_BGR Flip RGB to BGR, RGBA to BGRA PNG_TRANSFORM_SWAP_ALPHA Flip RGBA to ARGB or GA to AG PNG_TRANSFORM_INVERT_ALPHA Change alpha from opacity to transparency PNG_TRANSFORM_SWAP_ENDIAN Byte-swap 16-bit samples PNG_TRANSFORM_STRIP_FILLER Strip out filler bytes.

If you have valid image data in the info structure (you can use png_set_rows() to put image data in the info structure), simply do this:

png_write_png(png_ptr, info_ptr, png_transforms, NULL)

where png_transforms is an integer containing the logical OR of some set of transformation flags. This call is equivalent to png_write_info(), followed the set of transformations indicated by the transform mask, then png_write_image(), and finally png_write_end().

(The final parameter of this call is not yet used. Someday it might point to transformation parameters required by some future output transform.)

If you are going the low-level route instead, you are now ready to write all the file information up to the actual image data. You do this with a call to png_write_info().

png_write_info(png_ptr, info_ptr);

Note that there is one transformation you may need to do before png_write_info(). In PNG files, the alpha channel in an image is the level of opacity. If your data is supplied as a level of transparency, you can invert the alpha channel before you write it, so that 0 is fully transparent and 255 (in 8-bit or paletted images) or 65535 (in 16-bit images) is fully opaque, with

png_set_invert_alpha(png_ptr);

This must appear before png_write_info() instead of later with the other transformations because in the case of paletted images the tRNS chunk data has to be inverted before the tRNS chunk is written. If your image is not a paletted image, the tRNS data (which in such cases represents a single color to be rendered as transparent) won't need to be changed, and you can safely do this transformation after your png_write_info() call.

If you need to write a private chunk that you want to appear before the PLTE chunk when PLTE is present, you can write the PNG info in two steps, and insert code to write your own chunk between them:

png_write_info_before_PLTE(png_ptr, info_ptr); png_set_unknown_chunks(png_ptr, info_ptr, ...); png_write_info(png_ptr, info_ptr);

After you've written the file information, you can set up the library to handle any special transformations of the image data. The various ways to transform the data will be described in the order that they should occur. This is important, as some of these change the color type and/or bit depth of the data, and some others only work on certain color types and bit depths. Even though each transformation checks to see if it has data that it can do something with, you should make sure to only enable a transformation if it will be valid for the data. For example, don't swap red and blue on grayscale data.

PNG files store RGB pixels packed into 3 or 6 bytes. This code tells the library to strip input data that has 4 or 8 bytes per pixel down to 3 or 6 bytes (or strip 2 or 4-byte grayscale+filler data to 1 or 2 bytes per pixel).

png_set_filler(png_ptr, 0, PNG_FILLER_BEFORE);

where the 0 is unused, and the location is either PNG_FILLER_BEFORE or PNG_FILLER_AFTER, depending upon whether the filler byte in the pixel is stored XRGB or RGBX.

PNG files pack pixels of bit depths 1, 2, and 4 into bytes as small as they can, resulting in, for example, 8 pixels per byte for 1 bit files. If the data is supplied at 1 pixel per byte, use this code, which will correctly pack the pixels into a single byte:

png_set_packing(png_ptr);

PNG files reduce possible bit depths to 1, 2, 4, 8, and 16. If your data is of another bit depth, you can write an sBIT chunk into the file so that decoders can recover the original data if desired.

/* Set the true bit depth of the image data */ if (color_type & PNG_COLOR_MASK_COLOR) { sig_bit.red = true_bit_depth; sig_bit.green = true_bit_depth; sig_bit.blue = true_bit_depth; } else { sig_bit.gray = true_bit_depth; } if (color_type & PNG_COLOR_MASK_ALPHA) { sig_bit.alpha = true_bit_depth; }

png_set_sBIT(png_ptr, info_ptr, &sig_bit);

If the data is stored in the row buffer in a bit depth other than one supported by PNG (e.g. 3 bit data in the range 0-7 for a 4-bit PNG), this will scale the values to appear to be the correct bit depth as is required by PNG.

png_set_shift(png_ptr, &sig_bit);

PNG files store 16 bit pixels in network byte order (big-endian, ie. most significant bits first). This code would be used if they are supplied the other way (little-endian, i.e. least significant bits first, the way PCs store them):

if (bit_depth > 8) png_set_swap(png_ptr);

If you are using packed-pixel images (1, 2, or 4 bits/pixel), and you need to change the order the pixels are packed into bytes, you can use:

if (bit_depth < 8) png_set_packswap(png_ptr);

PNG files store 3 color pixels in red, green, blue order. This code would be used if they are supplied as blue, green, red:

png_set_bgr(png_ptr);

PNG files describe monochrome as black being zero and white being one. This code would be used if the pixels are supplied with this reversed (black being one and white being zero):

png_set_invert_mono(png_ptr);

Finally, you can write your own transformation function if none of the existing ones meets your needs. This is done by setting a callback with

png_set_write_user_transform_fn(png_ptr, write_transform_fn);

You must supply the function

void write_transform_fn(png_ptr ptr, row_info_ptr row_info, png_bytep data)

See pngtest.c for a working example. Your function will be called before any of the other transformations are processed.

You can also set up a pointer to a user structure for use by your callback function.

png_set_user_transform_info(png_ptr, user_ptr, 0, 0);

The user_channels and user_depth parameters of this function are ignored when writing; you can set them to zero as shown.

You can retrieve the pointer via the function png_get_user_transform_ptr(). For example:

voidp write_user_transform_ptr = png_get_user_transform_ptr(png_ptr);

It is possible to have libpng flush any pending output, either manually, or automatically after a certain number of lines have been written. To flush the output stream a single time call:

png_write_flush(png_ptr);

and to have libpng flush the output stream periodically after a certain number of scanlines have been written, call:

png_set_flush(png_ptr, nrows);

Note that the distance between rows is from the last time png_write_flush() was called, or the first row of the image if it has never been called. So if you write 50 lines, and then png_set_flush 25, it will flush the output on the next scanline, and every 25 lines thereafter, unless png_write_flush() is called before 25 more lines have been written. If nrows is too small (less than about 10 lines for a 640 pixel wide RGB image) the image compression may decrease noticeably (although this may be acceptable for real-time applications). Infrequent flushing will only degrade the compression performance by a few percent over images that do not use flushing.

That's it for the transformations. Now you can write the image data. The simplest way to do this is in one function call. If you have the whole image in memory, you can just call png_write_image() and libpng will write the image. You will need to pass in an array of pointers to each row. This function automatically handles interlacing, so you don't need to call png_set_interlace_handling() or call this function multiple times, or any of that other stuff necessary with png_write_rows().

png_write_image(png_ptr, row_pointers);

where row_pointers is:

png_byte *row_pointers[height];

You can point to void or char or whatever you use for pixels.

If you don't want to write the whole image at once, you can use png_write_rows() instead. If the file is not interlaced, this is simple:

png_write_rows(png_ptr, row_pointers, number_of_rows);

row_pointers is the same as in the png_write_image() call.

If you are just writing one row at a time, you can do this with a single row_pointer instead of an array of row_pointers:

png_bytep row_pointer = row;

png_write_row(png_ptr, row_pointer);

When the file is interlaced, things can get a good deal more complicated. The only currently (as of the PNG Specification version 1.2, dated July 1999) defined interlacing scheme for PNG files is the "Adam7" interlace scheme, that breaks down an image into seven smaller images of varying size. libpng will build these images for you, or you can do them yourself. If you want to build them yourself, see the PNG specification for details of which pixels to write when.

If you don't want libpng to handle the interlacing details, just use png_set_interlace_handling() and call png_write_rows() the correct number of times to write all seven sub-images.

If you want libpng to build the sub-images, call this before you start writing any rows:

number_of_passes = png_set_interlace_handling(png_ptr);

This will return the number of passes needed. Currently, this is seven, but may change if another interlace type is added.

Then write the complete image number_of_passes times.

png_write_rows(png_ptr, row_pointers, number_of_rows);

As some of these rows are not used, and thus return immediately, you may want to read about interlacing in the PNG specification, and only update the rows that are actually used.

After you are finished writing the image, you should finish writing the file. If you are interested in writing comments or time, you should pass an appropriately filled png_info pointer. If you are not interested, you can pass NULL.

png_write_end(png_ptr, info_ptr);

When you are done, you can free all memory used by libpng like this:

png_destroy_write_struct(&png_ptr, &info_ptr);

It is also possible to individually free the info_ptr members that point to libpng-allocated storage with the following function:

png_free_data(png_ptr, info_ptr, mask, n) mask - identifies data to be freed, a mask containing the logical OR of one or more of PNG_FREE_PLTE, PNG_FREE_TRNS, PNG_FREE_HIST, PNG_FREE_ICCP, PNG_FREE_PCAL, PNG_FREE_ROWS, PNG_FREE_SCAL, PNG_FREE_SPLT, PNG_FREE_TEXT, PNG_FREE_UNKN, or simply PNG_FREE_ALL n - sequence number of item to be freed (-1 for all items)

This function may be safely called when the relevant storage has already been freed, or has not yet been allocated, or was allocated by the user and not by libpng, and will in those cases do nothing. The "n" parameter is ignored if only one item of the selected data type, such as PLTE, is allowed. If "n" is not -1, and multiple items are allowed for the data type identified in the mask, such as text or sPLT, only the n'th item is freed.

If you allocated data such as a palette that you passed in to libpng with png_set_*, you must not free it until just before the call to png_destroy_write_struct().

The default behavior is only to free data that was allocated internally by libpng. This can be changed, so that libpng will not free the data, or so that it will free data that was allocated by the user with png_malloc() or png_zalloc() and passed in via a png_set_*() function, with

png_data_freer(png_ptr, info_ptr, freer, mask) mask - which data elements are affected same choices as in png_free_data() freer - one of

PNG_DESTROY_WILL_FREE_DATA PNG_SET_WILL_FREE_DATA PNG_USER_WILL_FREE_DATA

For example, to transfer responsibility for some data from a read structure to a write structure, you could use

png_data_freer(read_ptr, read_info_ptr, PNG_USER_WILL_FREE_DATA, PNG_FREE_PLTE | PNG_FREE_tRNS | PNG_FREE_hIST)
png_data_freer(write_ptr, write_info_ptr, PNG_DESTROY_WILL_FREE_DATA, PNG_FREE_PLTE | PNG_FREE_tRNS | PNG_FREE_hIST)

thereby briefly reassigning responsibility for freeing to the user but immediately afterwards reassigning it once more to the write_destroy function. Having done this, it would then be safe to destroy the read structure and continue to use the PLTE, tRNS, and hIST data in the write structure.

This function only affects data that has already been allocated. You can call this function before calling after the png_set_*() functions to control whether the user or png_destroy_*() is supposed to free the data. When the user assumes responsibility for libpng-allocated data, the application must use png_free() to free it, and when the user transfers responsibility to libpng for data that the user has allocated, the user must have used png_malloc() or png_zalloc() to allocate it.

If you allocated text_ptr.text, text_ptr.lang, and text_ptr.translated_keyword separately, do not transfer responsibility for freeing text_ptr to libpng, because when libpng fills a png_text structure it combines these members with the key member, and png_free_data() will free only text_ptr.key. Similarly, if you transfer responsibility for free'ing text_ptr from libpng to your application, your application must not separately free those members. For a more compact example of writing a PNG image, see the file example.c.

V. Modifying/Customizing libpng:

There are three issues here. The first is changing how libpng does standard things like memory allocation, input/output, and error handling. The second deals with more complicated things like adding new chunks, adding new transformations, and generally changing how libpng works. Both of those are compile-time issues; that is, they are generally determined at the time the code is written, and there is rarely a need to provide the user with a means of changing them. The third is a run-time issue: choosing between and/or tuning one or more alternate versions of computationally intensive routines; specifically, optimized assembly-language (and therefore compiler- and platform-dependent) versions.

Memory allocation, input/output, and error handling

All of the memory allocation, input/output, and error handling in libpng goes through callbacks that are user-settable. The default routines are in pngmem.c, pngrio.c, pngwio.c, and pngerror.c, respectively. To change these functions, call the appropriate png_set_*_fn() function.

Memory allocation is done through the functions png_malloc(), png_zalloc(), and png_free(). These currently just call the standard C functions. If your pointers can't access more then 64K at a time, you will want to set MAXSEG_64K in zlib.h. Since it is unlikely that the method of handling memory allocation on a platform will change between applications, these functions must be modified in the library at compile time. If you prefer to use a different method of allocating and freeing data, you can use

png_set_mem_fn(png_structp png_ptr, png_voidp mem_ptr, png_malloc_ptr malloc_fn, png_free_ptr free_fn)

This function also provides a void pointer that can be retrieved via

mem_ptr=png_get_mem_ptr(png_ptr);

Your replacement memory functions must have prototypes as follows:

png_voidp malloc_fn(png_structp png_ptr, png_uint_32 size); void free_fn(png_structp png_ptr, png_voidp ptr);

Input/Output in libpng is done through png_read() and png_write(), which currently just call fread() and fwrite(). The FILE * is stored in png_struct and is initialized via png_init_io(). If you wish to change the method of I/O, the library supplies callbacks that you can set through the function png_set_read_fn() and png_set_write_fn() at run time, instead of calling the png_init_io() function. These functions also provide a void pointer that can be retrieved via the function png_get_io_ptr(). For example:

png_set_read_fn(png_structp read_ptr, voidp read_io_ptr, png_rw_ptr read_data_fn)

png_set_write_fn(png_structp write_ptr, voidp write_io_ptr, png_rw_ptr write_data_fn, png_flush_ptr output_flush_fn);

voidp read_io_ptr = png_get_io_ptr(read_ptr); voidp write_io_ptr = png_get_io_ptr(write_ptr);

The replacement I/O functions must have prototypes as follows:

void user_read_data(png_structp png_ptr, png_bytep data, png_uint_32 length); void user_write_data(png_structp png_ptr, png_bytep data, png_uint_32 length); void user_flush_data(png_structp png_ptr);

Supplying NULL for the read, write, or flush functions sets them back to using the default C stream functions. It is an error to read from a write stream, and vice versa.

Error handling in libpng is done through png_error() and png_warning(). Errors handled through png_error() are fatal, meaning that png_error() should never return to its caller. Currently, this is handled via setjmp() and longjmp() (unless you have compiled libpng with PNG_SETJMP_NOT_SUPPORTED, in which case it is handled via PNG_ABORT()), but you could change this to do things like exit() if you should wish.

On non-fatal errors, png_warning() is called to print a warning message, and then control returns to the calling code. By default png_error() and png_warning() print a message on stderr via fprintf() unless the library is compiled with PNG_NO_CONSOLE_IO defined (because you don't want the messages) or PNG_NO_STDIO defined (because fprintf() isn't available). If you wish to change the behavior of the error functions, you will need to set up your own message callbacks. These functions are normally supplied at the time that the png_struct is created. It is also possible to redirect errors and warnings to your own replacement functions after png_create_*_struct() has been called by calling:

png_set_error_fn(png_structp png_ptr, png_voidp error_ptr, png_error_ptr error_fn, png_error_ptr warning_fn);

png_voidp error_ptr = png_get_error_ptr(png_ptr);

If NULL is supplied for either error_fn or warning_fn, then the libpng default function will be used, calling fprintf() and/or longjmp() if a problem is encountered. The replacement error functions should have parameters as follows:

void user_error_fn(png_structp png_ptr, png_const_charp error_msg); void user_warning_fn(png_structp png_ptr, png_const_charp warning_msg);

The motivation behind using setjmp() and longjmp() is the C++ throw and catch exception handling methods. This makes the code much easier to write, as there is no need to check every return code of every function call. However, there are some uncertainties about the status of local variables after a longjmp, so the user may want to be careful about doing anything after setjmp returns non-zero besides returning itself. Consult your compiler documentation for more details. For an alternative approach, you may wish to use the "cexcept" facility (see http://cexcept.sourceforge.net).

If you need to read or write custom chunks, you may need to get deeper into the libpng code. The library now has mechanisms for storing and writing chunks of unknown type; you can even declare callbacks for custom chunks. Hoewver, this may not be good enough if the library code itself needs to know about interactions between your chunk and existing 'intrinsic' chunks.

If you need to write a new intrinsic chunk, first read the PNG specification. Acquire a first level of understanding of how it works. Pay particular attention to the sections that describe chunk names, and look at how other chunks were designed, so you can do things similarly. Second, check out the sections of libpng that read and write chunks. Try to find a chunk that is similar to yours and use it as a template. More details can be found in the comments inside the code. It is best to handle unknown chunks in a generic method, via callback functions, instead of by modifying libpng functions.

If you wish to write your own transformation for the data, look through the part of the code that does the transformations, and check out some of the simpler ones to get an idea of how they work. Try to find a similar transformation to the one you want to add and copy off of it. More details can be found in the comments inside the code itself.

You will want to look into zconf.h to tell zlib (and thus libpng) that it cannot allocate more then 64K at a time. Even if you can, the memory won't be accessible. So limit zlib and libpng to 64K by defining MAXSEG_64K.

For DOS users who only have access to the lower 640K, you will have to limit zlib's memory usage via a png_set_compression_mem_level() call. See zlib.h or zconf.h in the zlib library for more information.

Libpng's support for medium model has been tested on most of the popular compilers. Make sure MAXSEG_64K gets defined, USE_FAR_KEYWORD gets defined, and FAR gets defined to far in pngconf.h, and you should be all set. Everything in the library (except for zlib's structure) is expecting far data. You must use the typedefs with the p or pp on the end for pointers (or at least look at them and be careful). Make note that the rows of data are defined as png_bytepp, which is an unsigned char far * far *.

You will need to write new error and warning functions that use the GUI interface, as described previously, and set them to be the error and warning functions at the time that png_create_*_struct() is called, in order to have them available during the structure initialization. They can be changed later via png_set_error_fn(). On some compilers, you may also have to change the memory allocators (png_malloc, etc.).

All includes for libpng are in pngconf.h. If you need to add/change/delete an include, this is the place to do it. The includes that are not needed outside libpng are protected by the PNG_INTERNAL definition, which is only defined for those routines inside libpng itself. The files in libpng proper only include png.h, which includes pngconf.h.

There are special functions to configure the compression. Perhaps the most useful one changes the compression level, which currently uses input compression values in the range 0 - 9. The library normally uses the default compression level (Z_DEFAULT_COMPRESSION = 6). Tests have shown that for a large majority of images, compression values in the range 3-6 compress nearly as well as higher levels, and do so much faster. For online applications it may be desirable to have maximum speed (Z_BEST_SPEED = 1). With versions of zlib after v0.99, you can also specify no compression (Z_NO_COMPRESSION = 0), but this would create files larger than just storing the raw bitmap. You can specify the compression level by calling:

png_set_compression_level(png_ptr, level);

Another useful one is to reduce the memory level used by the library. The memory level defaults to 8, but it can be lowered if you are short on memory (running DOS, for example, where you only have 640K).

png_set_compression_mem_level(png_ptr, level);

The other functions are for configuring zlib. They are not recommended for normal use and may result in writing an invalid PNG file. See zlib.h for more information on what these mean.

png_set_compression_strategy(png_ptr, strategy); png_set_compression_window_bits(png_ptr, window_bits); png_set_compression_method(png_ptr, method); png_set_compression_buffer_size(png_ptr, size);

If you want to control whether libpng uses filtering or not, which filters are used, and how it goes about picking row filters, you can call one of these functions. The selection and configuration of row filters can have a significant impact on the size and encoding speed and a somewhat lesser impact on the decoding speed of an image. Filtering is enabled by default for RGB and grayscale images (with and without alpha), but not for paletted images nor for any images with bit depths less than 8 bits/pixel.

The 'method' parameter sets the main filtering method, which is currently only '0' in the PNG 1.2 specification. The 'filters' parameter sets which filter(s), if any, should be used for each scanline. Possible values are PNG_ALL_FILTERS and PNG_NO_FILTERS to turn filtering on and off, respectively.

Individual filter types are PNG_FILTER_NONE, PNG_FILTER_SUB, PNG_FILTER_UP, PNG_FILTER_AVG, PNG_FILTER_PAETH, which can be bitwise ORed together with '|' to specify one or more filters to use. These filters are described in more detail in the PNG specification. If you intend to change the filter type during the course of writing the image, you should start with flags set for all of the filters you intend to use so that libpng can initialize its internal structures appropriately for all of the filter types.

filters = PNG_FILTER_NONE | PNG_FILTER_SUB PNG_FILTER_UP | PNG_FILTER_AVE | PNG_FILTER_PAETH | PNG_ALL_FILTERS; or filters = one of PNG_FILTER_VALUE_NONE, PNG_FILTER_VALUE_SUB, PNG_FILTER_VALUE_UP, PNG_FILTER_VALUE_AVE, PNG_FILTER_VALUE_PAETH

png_set_filter(png_ptr, PNG_FILTER_TYPE_BASE, filters); The second parameter can also be PNG_INTRAPIXEL_DIFFERENCING if you are writing a PNG to be embedded in a MNG datastream. This parameter must be the same as the value of filter_method used in png_set_IHDR().

It is also possible to influence how libpng chooses from among the available filters. This is done in two ways - by telling it how important it is to keep the same filter for successive rows, and by telling it the relative computational costs of the filters.

double weights[3] = {1.5, 1.3, 1.1}, costs[PNG_FILTER_VALUE_LAST] = {1.0, 1.3, 1.3, 1.5, 1.7};

png_set_filter_selection(png_ptr, PNG_FILTER_SELECTION_WEIGHTED, 3, weights, costs);

The weights are multiplying factors that indicate to libpng that the row filter should be the same for successive rows unless another row filter is that many times better than the previous filter. In the above example, if the previous 3 filters were SUB, SUB, NONE, the SUB filter could have a "sum of absolute differences" 1.5 x 1.3 times higher than other filters and still be chosen, while the NONE filter could have a sum 1.1 times higher than other filters and still be chosen. Unspecified weights are taken to be 1.0, and the specified weights should probably be declining like those above in order to emphasize recent filters over older filters.

The filter costs specify for each filter type a relative decoding cost to be considered when selecting row filters. This means that filters with higher costs are less likely to be chosen over filters with lower costs, unless their "sum of absolute differences" is that much smaller. The costs do not necessarily reflect the exact computational speeds of the various filters, since this would unduly influence the final image size.

Note that the numbers above were invented purely for this example and are given only to help explain the function usage. Little testing has been done to find optimum values for either the costs or the weights.

There are a bunch of #define's in pngconf.h that control what parts of libpng are compiled. All the defines end in _SUPPORTED. If you are never going to use a capability, you can change the #define to #undef before recompiling libpng and save yourself code and data space, or you can turn off individual capabilities with defines that begin with PNG_NO_.

You can also turn all of the transforms and ancillary chunk capabilities off en masse with compiler directives that define PNG_NO_READ[or WRITE]_TRANSFORMS, or PNG_NO_READ[or WRITE]_ANCILLARY_CHUNKS, or all four, along with directives to turn on any of the capabilities that you do want. The PNG_NO_READ[or WRITE]_TRANSFORMS directives disable the extra transformations but still leave the library fully capable of reading and writing PNG files with all known public chunks Use of the PNG_NO_READ[or WRITE]_ANCILLARY_CHUNKS directive produces a library that is incapable of reading or writing ancillary chunks. If you are not using the progressive reading capability, you can turn that off with PNG_NO_PROGRESSIVE_READ (don't confuse this with the INTERLACING capability, which you'll still have).

All the reading and writing specific code are in separate files, so the linker should only grab the files it needs. However, if you want to make sure, or if you are building a stand alone library, all the reading files start with pngr and all the writing files start with pngw. The files that don't match either (like png.c, pngtrans.c, etc.) are used for both reading and writing, and always need to be included. The progressive reader is in pngpread.c

If you are creating or distributing a dynamically linked library (a .so or DLL file), you should not remove or disable any parts of the library, as this will cause applications linked with different versions of the library to fail if they call functions not available in your library. The size of the library itself should not be an issue, because only those sections that are actually used will be loaded into memory.

The macro definition PNG_DEBUG can be used to request debugging printout. Set it to an integer value in the range 0 to 3. Higher numbers result in increasing amounts of debugging information. The information is printed to the "stderr" file, unless another file name is specified in the PNG_DEBUG_FILE macro definition.

When PNG_DEBUG > 0, the following functions (macros) become available:

png_debug(level, message) png_debug1(level, message, p1) png_debug2(level, message, p1, p2)

in which "level" is compared to PNG_DEBUG to decide whether to print the message, "message" is the formatted string to be printed, and p1 and p2 are parameters that are to be embedded in the string according to printf-style formatting directives. For example,

png_debug1(2, "foo=%d\n", foo);

is expanded to

if(PNG_DEBUG > 2) fprintf(PNG_DEBUG_FILE, "foo=%d\n", foo);

When PNG_DEBUG is defined but is zero, the macros aren't defined, but you can still use PNG_DEBUG to control your own debugging:

#ifdef PNG_DEBUG fprintf(stderr, ... #endif

When PNG_DEBUG = 1, the macros are defined, but only png_debug statements having level = 0 will be printed. There aren't any such statements in this version of libpng, but if you insert some they will be printed.

VI. MNG support

The MNG specification (available at http://www.libpng.org/pub/mng) allows certain extensions to PNG for PNG images that are embedded in MNG datastreams. Libpng can support some of these extensions. To enable them, use the png_permit_mng_features() function:

feature_set = png_permit_mng_features(png_ptr, mask) mask is a png_uint_32 containing the logical OR of the features you want to enable. These include PNG_FLAG_MNG_EMPTY_PLTE PNG_FLAG_MNG_FILTER_64 PNG_ALL_MNG_FEATURES feature_set is a png_32_uint that is the logical AND of your mask with the set of MNG features that is supported by the version of libpng that you are using.

It is an error to use this function when reading or writing a standalone PNG file with the PNG 8-byte signature. The PNG datastream must be wrapped in a MNG datastream. As a minimum, it must have the MNG 8-byte signature and the MHDR and MEND chunks. Libpng does not provide support for these or any other MNG chunks; your application must provide its own support for them. You may wish to consider using libmng (available at http://www.libmng.com) instead.

VII. Changes to Libpng from version 0.88

It should be noted that versions of libpng later than 0.96 are not distributed by the original libpng author, Guy Schalnat, nor by Andreas Dilger, who had taken over from Guy during 1996 and 1997, and distributed versions 0.89 through 0.96, but rather by another member of the original PNG Group, Glenn Randers-Pehrson. Guy and Andreas are still alive and well, but they have moved on to other things.

The old libpng functions png_read_init(), png_write_init(), png_info_init(), png_read_destroy(), and png_write_destory() have been moved to PNG_INTERNAL in version 0.95 to discourage their use. These functions will be removed from libpng version 2.0.0.

The preferred method of creating and initializing the libpng structures is via the png_create_read_struct(), png_create_write_struct(), and png_create_info_struct() because they isolate the size of the structures from the application, allow version error checking, and also allow the use of custom error handling routines during the initialization, which the old functions do not. The functions png_read_destroy() and png_write_destroy() do not actually free the memory that libpng allocated for these structs, but just reset the data structures, so they can be used instead of png_destroy_read_struct() and png_destroy_write_struct() if you feel there is too much system overhead allocating and freeing the png_struct for each image read.

Setting the error callbacks via png_set_message_fn() before png_read_init() as was suggested in libpng-0.88 is no longer supported because this caused applications that do not use custom error functions to fail if the png_ptr was not initialized to zero. It is still possible to set the error callbacks AFTER png_read_init(), or to change them with png_set_error_fn(), which is essentially the same function, but with a new name to force compilation errors with applications that try to use the old method.

Starting with version 1.0.7, you can find out which version of the library you are using at run-time:

png_uint_32 libpng_vn = png_access_version_number();

The number libpng_vn is constructed from the major version, minor version with leading zero, and release number with leading zero, (e.g., libpng_vn for version 1.0.7 is 10007).

You can also check which version of png.h you used when compiling your application:

png_uint_32 application_vn = PNG_LIBPNG_VER;

VIII. Y2K Compliance in libpng

January 31, 2001

Since the PNG Development group is an ad-hoc body, we can't make an official declaration.

This is your unofficial assurance that libpng from version 0.71 and upward through 1.0.9 are Y2K compliant. It is my belief that earlier versions were also Y2K compliant.

Libpng only has three year fields. One is a 2-byte unsigned integer that will hold years up to 65535. The other two hold the date in text format, and will hold years up to 9999.

The integer is "png_uint_16 year" in png_time_struct.

The strings are "png_charp time_buffer" in png_struct and "near_time_buffer", which is a local character string in png.c.

There are seven time-related functions:

png_convert_to_rfc_1123() in png.c (formerly png_convert_to_rfc_1152() in error) png_convert_from_struct_tm() in pngwrite.c, called in pngwrite.c png_convert_from_time_t() in pngwrite.c png_get_tIME() in pngget.c png_handle_tIME() in pngrutil.c, called in pngread.c png_set_tIME() in pngset.c png_write_tIME() in pngwutil.c, called in pngwrite.c

All appear to handle dates properly in a Y2K environment. The png_convert_from_time_t() function calls gmtime() to convert from system clock time, which returns (year - 1900), which we properly convert to the full 4-digit year. There is a possibility that applications using libpng are not passing 4-digit years into the png_convert_to_rfc_1123() function, or that they are incorrectly passing only a 2-digit year instead of "year - 1900" into the png_convert_from_struct_tm() function, but this is not under our control. The libpng documentation has always stated that it works with 4-digit years, and the APIs have been documented as such.

The tIME chunk itself is also Y2K compliant. It uses a 2-byte unsigned integer to hold the year, and can hold years as large as 65535.

zlib, upon which libpng depends, is also Y2K compliant. It contains no date-related code.

Glenn Randers-Pehrson libpng maintainer PNG Development Group

NOTE

Note about libpng version numbers:

Due to various miscommunications, unforeseen code incompatibilities and occasional factors outside the authors' control, version numbering on the library has not always been consistent and straightforward. The following table summarizes matters since version 0.89c, which was the first widely used release:

source png.h png.h shared-lib version string int version ------- ------ ----- ---------- 0.89c ("1.0 beta 3") 0.89 89 1.0.89 0.90 ("1.0 beta 4") 0.90 90 0.90 [should have been 2.0.90]

0.95 ("1.0 beta 5") 0.95 95 0.95 [should have been 2.0.95] 0.96 ("1.0 beta 6") 0.96 96 0.96 [should have been 2.0.96] 0.97b ("1.00.97 beta 7") 1.00.97 97 1.0.1 [should have been 2.0.97] 0.97c 0.97 97 2.0.97 0.98 0.98 98 2.0.98 0.99 0.99 98 2.0.99 0.99a-m 0.99 99 2.0.99 1.00 1.00 100 2.1.0 [100 should be 10000] 1.0.0 1.0.0 100 2.1.0 [100 should be 10000] 1.0.1 1.0.1 10001 2.1.0 1.0.1a-e 1.0.1a-e 10002 2.1.0.1a-e 1.0.2 1.0.2 10002 2.1.0.2 1.0.2a-b 1.0.2a-b 10003 2.1.0.2a-b 1.0.3 1.0.3 10003 2.1.0.3 1.0.3a-d 1.0.3a-d 10004 2.1.0.3a-d 1.0.4 1.0.4 10004 2.1.0.4 1.0.4a-f 1.0.4a-f 10005 2.1.0.4a-f 1.0.5 (+ 2 patches) 1.0.5 10005 2.1.0.5 1.0.5a-d 1.0.5a-d 10006 2.1.0.5a-d 1.0.5e-r 1.0.5e-r 10100 2.1.0.5e-r (not compatible) 1.0.5s-v 1.0.5s-v 10006 2.1.0.5s-v (compatible) 1.0.6 (+ 3 patches) 1.0.6 10006 2.1.0.6 1.0.6d 1.0.6d 10007 2.1.0.6d 1.0.7 1.0.7 10007 2.1.0.7 (still compatible)

Henceforth the source version will match the shared-library minor and patch numbers; the shared-library major version number will be used for changes in backward compatibility, as it is intended. The PNG_PNGLIB_VER macro, which is not used within libpng but is available for applications, is an unsigned integer of the form xyyzz corresponding to the source version x.y.z (leading zeros in y and z). Beta versions are given the previous public release number plus a letter or two.

SEE ALSO

libpngpf(3), png(5) *libpng :* ftp://ftp.uu.net/graphics/png http://www.libpng.org/pub/png

zlib : (generally) at the same location as *libpng* or at ftp://ftp.uu.net/pub/archiving/zip/zlib ftp://ftp.info-zip.org/pub/infozip/zlib

PNG specification: RFC 2083 (generally) at the same location as *libpng* or at ftp://ds.internic.net/rfc/rfc2083.txt or (as a W3C Recommendation) at http://www.w3.org/TR/REC-png.html

In the case of any inconsistency between the PNG specification and this library, the specification takes precedence.

AUTHORS

This man page: Glenn Randers-Pehrson <randeg@alum.rpi.edu>

The contributing authors would like to thank all those who helped with testing, bug fixes, and patience. This wouldn't have been possible without all of you.

Thanks to Frank J. T. Wojcik for helping with the documentation.

Libpng version 1.0.9 - January 31, 2001: Initially created in 1995 by Guy Eric Schalnat, then of Group 42, Inc. Currently maintained by Glenn Randers-Pehrson (randeg@alum.rpi.edu).

Supported by the PNG development group (png-implement@ccrc.wustl.edu).

COPYRIGHT NOTICE, DISCLAIMER, and LICENSE:

(This copy of the libpng notices is provided for your convenience. In case of any discrepancy between this copy and the notices in the file png.h that is included in the libpng distribution, the latter shall prevail.)

If you modify libpng you may insert additional notices immediately following this sentence.

libpng versions 1.0.7, July 1, 2000, through 1.0.9, January 31, 2001, are Copyright (c) 2000 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-1.0.6 with the following individuals added to the list of Contributing Authors

Simon-Pierre Cadieux Eric S. Raymond Gilles Vollant

and with the following additions to the disclaimer:

There is no warranty against interference with your enjoyment of the library or against infringement. There is no warranty that our efforts or the library will fulfill any of your particular purposes or needs. This library is provided with all faults, and the entire risk of satisfactory quality, performance, accuracy, and effort is with the user.

libpng versions 0.97, January 1998, through 1.0.6, March 20, 2000, are Copyright (c) 1998, 1999 Glenn Randers-Pehrson Distributed according to the same disclaimer and license as libpng-0.96, with the following individuals added to the list of Contributing Authors:

Tom Lane Glenn Randers-Pehrson Willem van Schaik

libpng versions 0.89, June 1996, through 0.96, May 1997, are Copyright (c) 1996, 1997 Andreas Dilger Distributed according to the same disclaimer and license as libpng-0.88, with the following individuals added to the list of Contributing Authors:

John Bowler Kevin Bracey Sam Bushell Magnus Holmgren Greg Roelofs Tom Tanner

libpng versions 0.5, May 1995, through 0.88, January 1996, are Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.

For the purposes of this copyright and license, "Contributing Authors" is defined as the following set of individuals:

Andreas Dilger Dave Martindale Guy Eric Schalnat Paul Schmidt Tim Wegner

The PNG Reference Library is supplied "AS IS". The Contributing Authors and Group 42, Inc. disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The Contributing Authors and Group 42, Inc. assume no liability for direct, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of the PNG Reference Library, even if advised of the possibility of such damage.

Permission is hereby granted to use, copy, modify, and distribute this source code, or portions hereof, for any purpose, without fee, subject to the following restrictions:

1. The origin of this source code must not be misrepresented.

2. Altered versions must be plainly marked as such and must not be misrepresented as being the original source.

3. This Copyright notice may not be removed or altered from any source or altered source distribution.

The Contributing Authors and Group 42, Inc. specifically permit, without fee, and encourage the use of this source code as a component to supporting the PNG file format in commercial products. If you use this source code in a product, acknowledgment is not required but would be appreciated.

A "png_get_copyright" function is available, for convenient use in "about" boxes and the like:

printf("%s",png_get_copyright(NULL));

Also, the PNG logo (in PNG format, of course) is supplied in the files "pngbar.png" and "pngbar.jpg (88x31) and "pngnow.png" (98x31).

Libpng is OSI Certified Open Source Software. OSI Certified Open Source is a certification mark of the Open Source Initiative.

Glenn Randers-Pehrson randeg@alum.rpi.edu January 31, 2001

Conclusion

This chapter hasn't been as complete as the TIFF chapter, for which I apologize. I have shown you all the important things about the PNG format though -- the layout of the images on disc, how to open and read an image, how to write an image, how to use the client call backs to save data in places other than files, and I have included the libpng documentation.

Further reading

- http://www.libpng.org/pub/png/png.html: The libpng homepage.
- http://www.libpng.org/pub/png/pngintro.html: An introduction to PNG features.
- http://www.libpng.org/pub/png/pngfaq.html: The PNG FAQ.
- http://www.libpng.org/pub/png/pnghist.html: A history of PNG.
- http://www.libpng.org/pub/png/spec/: Version 1.2 of the PNG specification.
- http://www.libpng.org/pub/png/libpng.html: libpng download page

Notes

1. Short for Portable Network Graphics.
2. For instance Netscape 4 and Internet Explorer 4 and later.
3. In reality, the libtiff could just use the **FILE *** method internally, although they don't always, as shown by the TIFFClientOpen examples in the TIFF chapter.

Chapter 6. PDF

"THE ORIGINS OF THE Portable Document Format and the Adobe Acrobat product family date to early 1990. At that time, the PostScript page description language was rapidly becoming the worldwide standard for the production of the printed page. PDF builds on the PostScript page description language by layering a document structure and interactive navigation features on PostScript's underlying imaging model, providing a convenient, efficient mechanism enabling documents to be reliably viewed and printed anywhere. The PDF specification was first published at the same time the first Acrobat products were introduced in 1993." -- PDF Specification

Portable Document Format (PDF) isn't strictly an image format. However, people are increasingly asking for PDF functionality in applications -- especially those on the Internet.

This chapter will focus on PDF 1.3 (second edition), because that is the specification version which I am most familiar with at the moment.

Introduction

In a sense this PDF chapter is the culmination of the tutorial... Many of the formats we have discussed up to this point can be included in some way in PDF files. I think that PDF is probably the most interesting imaging format in common use today.

This chapter is broken into two major sections. These are: a discussion of the PDF format, and then an introduction to Panda, a PDF generation API.

All about the PDF format

The PDF file format is broken into a tree structure. This tree is made up of combinations of a few possible objects. We'll start by describing these objects, and then move onto how they fit together.

File header

Every PDF file starts with a simple header which declares the file to be a valid PDF file. This header will look something like this:

```
%PDF-1.3 °<9F><92><9C><9F>ÔàÎÐÐÐ
```

This header had the following parts:

- *%PDF-*: this is a PDF document.
- *1.3*: it meets version 1.3 of the PDF specification.
- °*<9F><92><9C><9F>ÔàÎÐÐÐ*: random binary stuff. This is just here so that "smart" FTP clients don't decide the file is an ASCII file in error [1] .

Specification versions

There have been five versions of the PDF specification released at the time of writing this document. They are:

- *1.0*: maps functionality available in Adobe Acrobat 1.0
- *1.1*: maps functionality available in Adobe Acrobat 2.0
- *1.2*: maps functionality available in Adobe Acrobat 3.0

- *1.3 (both first and second editions)*: maps functionality available in Adobe Acrobat 4.0
- *1.4*: maps functionality available in Adobe Acrobat 5.0

In theory at least, the PDF specification versions should be backwards compatible -- viewers should ignore object types and dictionary entries that they don't understand. This means that you can also insert your own information into the PDF document without breaking its ability to be viewed in other applications.

Objects

Objects in PDF files have a number, and a generation (version) number. They are represented as an ASCII text sequence in the file, for instance:

```
1 0 obj
<<
        /Type /Catalog
        /Pages 2 0 R
>>
endobj
```

This object is the 0th version of object number 1. The text between the << and the >> is discussed in the next section.

Order of objects in the file

Note that objects can appear in the file in any order, because there is a look up table called an XREF table at the end of the file that they can be looked up from.

Dictionaries

Objects have associated with them a (key, value) pair database. This stores properties of the object -- these would normally include things like the dimensions of the page, or the name of the author of the document, and other interesting things like that.

In the example object above, we had the following dictionary items...

```
        /Type /Catalog
        /Pages 2 0 R
```

This dictionary specifies that the key "Type" has the value "Catalog", and that the value "Pages" has the value "2 0 R". This last value is an object reference, which we discuss in the Redirection section a little bit later in this chapter.

Dictionary data types

What data types are valid in dictionaries? Well, version 1.3 of the PDF specification names the following data types:

Arrays

An array is a one dimensional collection of other values. Unlike most programming languages, the contents of the array can be of several types, for instance we could mix integers with strings in a single array. Arrays start and end with square brackets.

Examples

```
[(foo) (bar) 42.3 /AName]
```

Boolean

Boolean values are represented with the words **true** and **false**.

Examples

```
/ExplodeOnOpening true
/Rotate false
```

Names

A name is a sequence of characters not including whitespace which follow a forward slash. Names are used in object dictionaries for the names of keys, and for some values of keys.

Examples

Numbers

Known as numeric types in the PDF specification (for somewhat obvious reasons), this includes all forms of numbers. These can either be integer or real [2] numbers.

Integers

Integer numbers can be either positive or negative (with a leading sign value if needed) and have a maximum value of ... and a minimum value of ... .

Examples

```
123
43445
+176
-17
0
```

Real numbers

Real numbers can exist in the range ... to ..., and may or may not have leading zeros.

Examples

```
34.5
-3.62
+123.5
4.
```

```
-.002
0.0
```

Caveats

The PDF specification makes the following point: "PDF does not support the PostScript syntax for numbers with non-decimal radices (such as 16#FFFE) or in exponent format (such as 6.02E23)." [3]

Strings

Strings are represented a series of unsigned bytes [4] which is identical to the ASCII strings most programmers are familar with in C. There are two main representations of strings. These are:

Bracket notation

Strings can be wrapped in curved brackets such as () to delimit the start and end of the string. Strings may also contain brackets, so long as they are balanced or escaped with a backslash.

These are known as literal strings in the PDF specification.

Examples

```
(hello)
(hello world)
(hello world \(the people I like\))
(hello world (the people I like))
```

Escaped characters

There are a series of standard characters which are used with the backslash escape. These are:

- *n*: Newline
- *r*: Carriage return
- *t*: Tab
- *b*: Backspace
- *f*: Form feed
- \(: Open bracket
- \): Close bracket
- \\: Backslash
- *ddd*: Arbitary character (ddd is a number in octal)

The backslash operator can also be used to continue text on the next line. For instance:

```
(This is a very long string which we want to \
break over a couple of lines.)
```

This means that we can also embed newlines without using the escape. For instance, these two text blocks are the same:

```
(This is a
string \
with some lines)
```

and

```
(This is a\nstring with some lines)
```

More on strings in this notation can be found in the PDF specification, version 1.3, on page 30.

Hexadecimal notation

Strings can also be written in hexadecimal form, and in this case are enclosed in angle brackets.

Examples

An example of a hexidecimal string is:

```
<4E6F762073686D6F7A206B6120706F702E>
```

If the final digit is missing, then it is assumed to be zero. For instance, the following string:

```
<901FA>
```

Is the same as:

```
<901FA0>
```

A lexer for PDF

Refer to the lexer for PDF documents at the end of this chapter if you are more interested in the format of data types.

Minimal dictionaries

In addition to having full dictionaries, it is possible to have very simple objects which only have one value in the dictionary, these objects look a little different than a normal object, for instance:

```
17 0 obj
203
endobj
```

In this example, the object stores a simple integer value. Why would you want to do this? Well, the answer is that in a dictionary you can have the value for the key stored in another object:

```
42 0 obj
<<
  /Length 24 0 R
>>
endobj

24 0 obj
```

```
462
endobj
```

Here, we don't know the length of the stream associated with the object (I describe these in a second) at the time we wrote the object out, so we can simply insert it later in the document. This is called an object reference, which is in the form:

```
objectnumber revision R
```

Streams

The other type of data that an object can have associated with it is a stream. Streams are used to store less structured data, for instance descriptions of the items on a page, or random binary data. The form for a stream is:

```
42 0 obj
<<
  /Length 24 0 R
>>
stream
...stream data...
endstream
endobj
```

A description for the page layout descriptions is outside the scope of this document. The other type of information which can be stored in a stream is arbitrary information such as the content of images (which would normally be raster information). An in depth discussion of the formatting of the raster information is also outside the scope of this document.

Filters on streams

Streams can be filtered. A filter is an operation which occurs on the stream contents before they are saved into the PDF document, examples include compression, and ASCII85 encoding [5] , which can't really be called compression.

So what are the possible filters? Well, as of PDF 1.3 [6] , they are:

ASCII85Decode

The ASCII85 filter takes binary data, and turns it into base 85 representation. This is needed by some email clients (among other things), because they can't handle embedding binary data into their protocols.

The *comp.text.pdf frequently asked questions* refers to the code examples below by way of explaination:

```
/* encode85 -- convert to ascii85 format */

#include <stdio.h>
#define    atoi(s)    strtol(s, 0, 0)

static unsigned long width = 72, pos = 0, tuple = 0;
static int count = 0;

void init85(void) {
    printf("<~");
    pos = 2;
}

void encode(unsigned long tuple, int count) {
```

```
    int i;
    char buf[5], *s = buf;
    i = 5;
    do {
        *s++ = tuple % 85;
        tuple /= 85;
    } while (--i > 0);
    i = count;
    do {
        putchar(*--s + '!');
        if (pos++ >= width) {
            pos = 0;
            putchar('\n');
        }
    } while (i-- > 0);
}

void put85(unsigned c) {
    switch (count++) {
    case 0:    tuple |= (c << 24); break;
    case 1: tuple |= (c << 16); break;
    case 2:    tuple |= (c <<  8); break;
    case 3:
        tuple |= c;
        if (tuple == 0) {
            putchar('z');
            if (pos++ >= width) {
                pos = 0;
                putchar('\n');
            }
        } else
            encode(tuple, count);
        tuple = 0;
        count = 0;
        break;
    }
}

void cleanup85(void) {
    if (count > 0)
        encode(tuple, count);
    if (pos + 2 > width)
        putchar('\n');
    printf("~>\n");
}

void copy85(FILE *fp) {
    unsigned c;
    while ((c = getc(fp)) != EOF)
        put85(c);
}

void usage(void) {
    fprintf(stderr, "usage: encode85 [-w width] file ...\n");
    exit(1);
}

extern int getopt(int, char *[], const char *);
extern int optind;
extern char *optarg;

int main(int argc, char *argv[]) {
    int i;
    while ((i = getopt(argc, argv, "w:?")) != EOF)
        switch (i) {
        case 'w':
```

```
            width = atoi(optarg);
            if (width == 0)
                width = ~0;
            break;
        case '?':
            usage();
        }

    init85();
    if (optind == argc)
        copy85(stdin);
    else
        for (i = optind; i < argc; i++) {
            FILE *fp = fopen(argv[i], "r");
            if (fp == NULL) {
                perror(argv[i]);
                return 1;
            }
            copy85(fp);
            fclose(fp);
        }
    cleanup85();
    return 0;
}
```

Code: encode85.c

```
/* decode85 -- convert from ascii85 format */

#include <stdio.h>

static unsigned long pow85[] = {
    85*85*85*85, 85*85*85, 85*85, 85, 1
};

void wput(unsigned long tuple, int bytes) {
    switch (bytes) {
    case 4:
        putchar(tuple >> 24);
        putchar(tuple >> 16);
        putchar(tuple >>  8);
        putchar(tuple);
        break;
    case 3:
        putchar(tuple >> 24);
        putchar(tuple >> 16);
        putchar(tuple >>  8);
        break;
    case 2:
        putchar(tuple >> 24);
        putchar(tuple >> 16);
        break;
    case 1:
        putchar(tuple >> 24);
        break;
    }
}

void decode85(FILE *fp, const char *file) {
    unsigned long tuple = 0;
    int c, count = 0;
    for (;;)
        switch (c = getc(fp)) {
        default:
            if (c < '!' || c > 'u') {
                fprintf(stderr, "%s: bad character in ascii85 region: %#o\n", file, c);
```

```
                exit(1);
            }
            tuple += (c - '!') * pow85[count++];
            if (count == 5) {
                wput(tuple, 4);
                count = 0;
                tuple = 0;
            }
            break;
        case 'z':
            if (count != 0) {
                fprintf(stderr, "%s: z inside ascii85 5-tuple\n", file);
                exit(1);
            }
            putchar(0);
            putchar(0);
            putchar(0);
            putchar(0);
            break;
        case '~':
            if (getc(fp) == '>') {
                if (count > 0) {
                    count--;
                    tuple += pow85[count];
                    wput(tuple, count);
                }
                c = getc(fp);
                return;
            }
            fprintf(stderr, "%s: ~ without > in ascii85 section\n", file);
            exit(1);
        case '\n': case '\r': case '\t': case ' ':
        case '\0': case '\f': case '\b': case 0177:
            break;
        case EOF:
            fprintf(stderr, "%s: EOF inside ascii85 section\n", file);
            exit(1);
        }
}

void decode(FILE *fp, const char *file, int preserve) {
    int c;
    while ((c = getc(fp)) != EOF)
        if (c == '<')
            if ((c = getc(fp)) == '~')
                decode85(fp, file);
            else {
                if (preserve)
                    putchar('<');
                if (c == EOF)
                    break;
                if (preserve)
                    putchar(c);
            }
        else
            if (preserve)
                putchar(c);
}

void usage(void) {
    fprintf(stderr, "usage: decode85 [-p] file ...\n");
    exit(1);
}

extern int getopt(int, char *[], const char *);
extern int optind;
```

```
extern char *optarg;

int main(int argc, char *argv[]) {
    int i, preserve;
    preserve = 0;
    while ((i = getopt(argc, argv, "p?")) != EOF)
        switch (i) {
        case 'p': preserve = 1; break;
        case '?': usage();
        }


    if (optind == argc)
        decode(stdin, "decode85", preserve);
    else
        for (i = optind; i < argc; i++) {
            FILE *fp = fopen(argv[i], "r");
            if (fp == NULL) {
                perror(argv[i]);
                return 1;
            }
            decode(fp, argv[i], preserve);
            fclose(fp);
        }
    return 0;
}
```

Code: decode85.c

These two code snippets were written by Paul Haahr, http://www.webcom.com/~haahr/, and is stated to be in the public domain.

ASCIIHexDecode

For similar reasons to ASCII85, you can also represent your binary data as hexidecimal. This filter implements this.

CCITTFaxDecode

This is the compression codec known as group3 and group 4 fax in TIFF. It is about as good as compression gets for black and white images.

DCTDecode

DCT (Descrete Cosine Transform) is the compression codec used by JPEG images. As discussed elsewhere in this document, it is good for color images, but is lossy.

FlateDecode

Flate compression (which is implemented by zlib), is the compression codec using in PNG images. It is very good for colour image and textual data.

LZWDecode

LZW is the compression codec used by GIF images. No publically available libraries implement LZW compression, and it is recommended you don't use it as many Acrobat viewers don't implement LZW decompression. Even Adobe's own products stear away from using this filter.

RunLengthDecode

Run length compression is a very simple compression codec, and it not recommended for most purposes.

Object structure

The diagram below shows the basic object structure of a PDF document. It can be much more complex than this, especially if you reuse commonly used objects like the logo which is on every page of the document. In words, the structure is something like:

Catalog Object

Every PDF document has a catalog object. This catalog object refers to a pages object.

Pages object

The pages object stores a list of pages within the PDF document, in the form of a dictionary array with the key name "/Kids". Each of these pages will have an object.

A Page object per page

Each page object will have a content object.

Content objects

Each content object will refer to the pages object, as well as referring to resources that that needed to draw this page. The resources can be used by other contents objects as well. Resources are things like fonts, and images.

A postscript-like description of the layout of the page is stored in this object's stream.

Resources

A resources object stores information you need to be able to use a given resource such as a font or image.

If the resource is an image or an embedded font, then the additional binary data (such as a raster) is stored in this objects stream.

Typical object structure

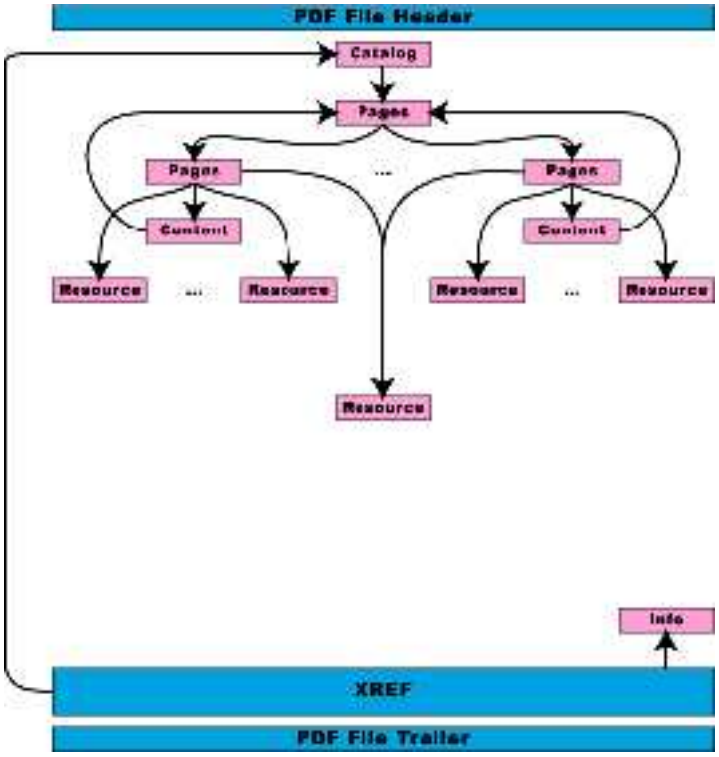

Figure 6-1. A typical PDF object structure

Support for presentations

PDF documents can be used to create presentation slide shows. There are two main features which support this use of PDF -- page duration, and page transitions. The page duration is the amount of time that the page will appear on the screen before the PDF viewer automatically moves onto the next page. The transition is the effect applied to the PDF pages when they are being swapped between -- this is very similar to the Microsoft Power Point presentations.

It is important to note that not all PDF viewers support this functionality.

Panda

Panda is my own PDF generation library. I wrote it because I was not happy with the license and some of the limitations with ClibPDF, and the license for PDFlib wasn't acceptable. To be honest, the best way to learn about something is also to build an implementation of it, which was an additional motivation.

You can get Panda from http://www.stillhq.com, and is licensed under the GNU GPL, which can be found at the start of this tutorial.

Installation

First, download Panda from http://www.stillhq.com, and then follow these simple steps:

Unix

Follow these simple steps to install Panda on your unix system:

- Extract the tarball
- Change into the Panda directory
- ./configure
- make
- make install

This will also build the examples in the examples directory.

win32

Panda currently compiles quite happily on Windows, and if you download the right version of the tarball even includes a Microsoft Visual Studio project file. Note that the current version of Panda (0.5.1), doesn't include a COM wrapper for Panda, so you either need to be programing in C or C++, or a language which can import DLL symbols (for instance Microsoft Visual Basic).

Hello world example

The most basic PDF example is the hello world application. Here is the Panda iteration:

```
#include  <panda/functions.h>
#include  <panda/constants.h>

int
main (int argc, char *argv[])
{
  panda_pdf *demo;
  panda_page *currPage;

  // Initialise the library
  panda_init ();

  // Open our demo PDF
  if ((demo = panda_open ("hello.pdf", "w")) == NULL)
    {
      fprintf (stderr, "Could not open hello.pdf\n");
      exit (1);
    }

  // Create a page
  currPage = panda_newpage (demo, panda_pagesize_a4);

  // Write some text to the page
  panda_setfont (demo, panda_createfont (demo, "Times-Roman", 1,
                       "MacRomanEncoding"));
  panda_textbox (demo, currPage, 600, 10, 700, 300, "Hello world");

  // Finished all the demoing, close the PDF document
  panda_close (demo);
```

```
  return 0;
}
```

Code: hello.c

Which produces...

Figure 6-2.

Initialization

Initialization of Panda is easy. Simply call **panda_init**(). You'll also need to create a PDF document, which is done with the **panda_open** function.

panda_init

Name

`panda_init` — setup Panda ready for use

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_init (void);
```

DESCRIPTION

Performs some simple setup of Panda before it is used for the first time in your application.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_init();

panda_open

Name

panda_open — open a PDF document

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
panda_pdf * panda_open (char *filename, char *mode);
```

DESCRIPTION

Open the named PDF document with the mode specified. The only mode currently supported is "w", but others will be integrated later. The interface to this function is identical in it's behaviour to the **fopen**() function call offered by the ANSI C standard IO library.

RETURNS

A pointer to a panda_pdf structure

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w");

SEE ALSO

panda_init, panda_open_actual, panda_open_suppress, panda_close

Creating pages

Pages are created in Panda using the **panda_newpage** function. You can magically be editing as many pages at a time as you like with Panda without any additional calls being needed.

panda_newpage

Name

`panda_newpage` — create a new page in the PDF

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
panda_page *panda_newpage(panda_pdf *document, char *pagesize);
```

DESCRIPTION

Create a new blank page at the end of the PDF with the specified size. Use the standard pagesize strings that are defined by Panda for most things. These are **panda_pagesize_a4**, and **panda_pagesize_usletter**. If you need to create your own page sizes, then have a look at these for hints.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4);

SEE ALSO

panda_open, panda_close

Object properties

Panda allows you to selectively apply properties to portions of your PDF document. The most common example is choosing which portions of your PDF document to compress.

panda_setobjectproperty

Name

`panda_setobjectproperty` — set a property value for an object

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setobjectproperty (panda_object *target, int scope, int propid, int propval
```

DESCRIPTION

Properties are a way of specifing things about objects. These properties can have either a cascade scope (they affect all subsequently created objects that are children of that object) -- **panda_scope_cascade**, or local (they only occur for that object) -- **panda_scope_local**. Possible properties are defined in the **panda_const_properties** manual page.

RETURNS

None

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_object *obj; panda_init(); document = panda_open("filename.pdf", "w"); obj = panda_newobject(document, panda_object_normal); panda_setproperty(obj, panda_scope_cascade, panda_object_property_compress, panda_true);

SEE ALSO

panda_newobject, panda_const_properties

Finalizing

Pages don't need to be closed in Panda. This is done when the **panda_close** function is called. This function writes the entire PDF document out to disc.

panda_close

Name

`panda_close` — write a PDF document out to disk

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_close (panda_pdf *document);
```

DESCRIPTION

Write out the PDF document we have created to disk, clean up and free memory.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_close(document);

SEE ALSO

panda_open

Inserting text

One of the advantages which ClibPDF has over Panda is that it currently supported word wrap, whereas Panda doesn't. Panda does have a variety of text functionality however. The public text functions in Panda are: **panda_textboxrot**, which creates a text box at a jaunty angle, **panda_textbox**, which creates a horizontal textbox (a zero angle), and **panda_textdirection**, which sets the flow direction for text.

panda_textbox

Name

`panda_textbox` — display some text on the PDF page specified

Synopsis

```
 #include<panda/constants.h>
 #include<panda/functions.h>
 void panda_textbox (panda_pdf * output, panda_page * thisPage, int top, int left, int
tom, int right, char *text);
```

DESCRIPTION

This function call creates a textbox on the specified page, and then displays the specified text within that page. The current font mode and style et cetera will be used. Sometime in the near future, line wrapping will be used...

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_textbox (demo, currPage, 20 + (lineDepth * 20), 200, 40 + (lineDepth * 20), 400, "Demonstration of a text mode");

SEE ALSO

panda_createfont, panda_setfont, panda_panda_setfontsize, panda_getfontobj, panda_setfontmode, panda_setcharacterspacing, panda_setwordspacing, panda_sethorizontalscaling, panda_setleading, panda_textboxrot

panda_textboxrot

Name

`panda_textboxrot` — display some text at a jaunty angle on the PDF page specified

Synopsis

```
 #include<panda/constants.h>
 #include<panda/functions.h>
 void panda_textbox (panda_pdf * output, panda_page * thisPage, int top, int left, int
tom, int right, double angle, char *text);
```

DESCRIPTION

This function call creates a textbox on the specified page, and then displays the specified text within that page. The text is displayed at the spefied angle. The current font mode and style et cetera will be used. Sometime in the near future, line wrapping will be used...

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_textboxrot (demo, currPage, 20 + (lineDepth * 20), 200, 40 + (lineDepth * 20), 400, 33.0, "Demonstration of a text mode");

SEE ALSO

panda_createfont, panda_setfont, panda_panda_setfontsize, panda_getfontobj, panda_setfontmode, panda_setcharacterspacing, panda_setwordspacing, panda_sethorizontalscaling, panda_setleading, panda_textbox

panda_textdirection

Name

`panda_textdirection` — specify the direction that the text flows within the document

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_textdirection (panda_pdf *document, int dir);
```

DESCRIPTION

This function records information within the PDF indicating the direction that the text in the document flows in. The possible values for the dir arguement are: **panda_textdirection_l2r**, text is read left to right; **panda_textdirection_r2l**, text is read right to left. The default for this value is **panda_textdirection_l2r**.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_textdirection(document, panda_textdirection_r2l);

Fonts

Panda currently only supports the standard PDF fonts, and will not allow you to embed arbitrary fonts into your PDF documents (unlike PDFlib). Some users might find this a little limiting for the time being. In Panda, you use fonts by first creating a pointer to the font using **panda_createfont**, and then start using that font with **panda_setfont**. This allows you use efficiently create the five fonts you are going to use in the document, and then swap backwards and forwards within that set of five with no performance penalty.

panda_createfont

Name

`panda_createfont` — return a handle to the requested font

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
char * panda_createfont (panda_pdf * output, char *fontname, int type, char *encoding)
```

DESCRIPTION

PANDA INTERNAL. This funtion call creates a font object for the requested font and returns the identifier that should be used when the font is set within the PDF document.

RETURNS

A font identifier (handle) as a null terminated string.

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *output; char *fonthandle; panda_init(); output = panda_open("output.pdf", "w"); fonthandle = (output, "Helventica", 3, "MacRomanEncoding");

SEE ALSO

panda_setfont

panda_setfont

Name

`panda_setfont` — set the current font to be that specified

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setfont (char *fontident);
```

DESCRIPTION

Once you have generated a font identifier for a given font, you can then set that current font to that font using this call. Create a font identifier with the **panda_createfont**() call.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *output; char *fonthandle; panda_init(); output = panda_open ("output.pdf", "w"); fonthandle = panda_createfont (output, "Helventica", 3, "MacRomanEncoding"); panda_setfont (fonthandle);

SEE ALSO

panda_createfont

Font attributes

Panda also allows you to set a variety of font attributes...

panda_setcharacterspacing

Name

`panda_setcharacterspacing` — set the space between characters

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setcharacterspacing (panda_page *target, double amount);
```

DESCRIPTION

Set the amount of additional space between characters in points.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *output; panda_init(); output = panda_open("output.pdf", "w"); panda_setcharacterspacing(output, 42.3);

SEE ALSO

panda_setwordspacing, panda_sethorizontalscaling, panda_setleading

panda_setfillcolor

Name

`panda_setfillcolor` — set the color to fill a close shape with

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setfillcolor (panda_page *target, int red, int green, int blue);
```

DESCRIPTION

This function sets the color to fill a close shape with when the shape is closed. It is expressed as a combinartion of red, green, and blue. The maximum number for each value is 255 (a number greater than 255 is reduced to 255).

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_setfillcolor (page, 100, 200, 300); panda_addlinesegment (page, 200, 200); panda_addlinesegment (page, 250, 300); panda_closeline (page); panda_endline (page);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setlinecolor

panda_setfontmode

Name

panda_setfontmode — set the current font mode

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setfontmode (panda_page *target, int mode);
```

DESCRIPTION

Set the drawing mode for the current font. Valid modes are: panda_textmode_normal, panda_textmode_outline, panda_textmode_filledoutline, panda_textmode_invisible, panda_textmode_filledclipped, panda_textmode_strokedclipped, panda_textmode_filledstrokedclipped and panda_textmode_clipped.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *output; panda_init(); output = panda_open("output.pdf", "w"); panda_setfontmode(output, panda_textmode_outline);

SEE ALSO

panda_setfontsize, panda_setfont

panda_setfontsize

Name

panda_setfontsize — set the current font size

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setfontsize (panda_page *target, int size);
```

DESCRIPTION

Set the size of the font to be used next (in points).

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *output; panda_init(); output = panda_open("output.pdf", "w"); setfontsize(output, 42);

SEE ALSO

panda_setfontmode, panda_setfont

panda_sethorizontalscaling

Name

`panda_sethorizontalscaling` — set the horizontal scaling of text

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_sethorizontalscaling (panda_pdf *output, double amount);
```

DESCRIPTION

Set the horizontal scaling factor of the text in percent.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *output; panda_init(); output = panda_open("output.pdf", "w"); panda_sethorizontalscaling(output, 42.3);

SEE ALSO

panda_setcharacterspacing, panda_setwordspacing, panda_setleading

panda_setleading

Name

`panda_setleading` — set the amount of space between lines of text

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setleading (panda_pdf *output, double amount);
```

DESCRIPTION

Set the amount of space between lines of text in points.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *output; panda_init(); output = panda_open("output.pdf", "w"); panda_setleading(output, 42.3);

SEE ALSO

panda_setcharacterspacing, panda_setwordspacing, panda_sethorizontalscaling

panda_setwordspacing

Name

`panda_setwordspacing` — set the space between words

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setwordspacing (panda_page *target, double amount);
```

DESCRIPTION

Set the amount of additional space between words in points.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *output; panda_init(); output = panda_open("output.pdf", "w"); panda_setwordspacing(output, 42.3);

SEE ALSO

panda_setcharacterspacing, panda_sethorizontalscaling, panda_setleading

Inserting raster images

Panda probably has the best raster image support of any of the PDF libraries available (not including Adobe's libraries). This is because this is my main area of expertise, and has been the main focus of the Panda development effort. For example ClibPDF only supports TIFF images in a limited set of formats, whereas I am not aware of any TIFF files which cannot be inserting into a PDF with Panda. Panda also never creates temporary files on disc when it needs to convert between TIFF formats, unlike PDFlib.

TIFF support

Panda support TIFF fully.

JPEG support

Panda supports JPEG fully.

PNG support

Panda supports PNG fully.

Inserting images onto pages

In Panda you use the **panda_imagebox**, and the **panda_imageboxrot** functions to insert images.

panda_imagebox

Name

panda_imagebox — insert an image into the PDF document at the specified location

Synopsis

```
 #include<panda/constants.h>
 #include<panda/functions.h>
 void panda_panda_imagebox (panda_pdf * output, panda_page * target, int top, int left,
tom, int right, char *filename, int type);
```

DESCRIPTION

This function call inserts an image into the PDF document at the specified location using a reasonable default for rotation (none). This call is included for backward compatability withprevious releases of the API and it is recommened that new code call **panda_imageboxrot**(). It is unlikely that this call will be retired however. The image types accepted by this call are: panda_image_tiff, panda_image_jpeg and panda_image_png.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *demo; panda_page *currPage; panda_init (); if ((demo = panda_open ("output.pdf", "w")) == NULL) panda_error (panda_true, "demo: could not open output.pdf to write to."); currPage = panda_newpage (demo, panda_pagesize_a4); panda_imagebox (demo, currPage, 0, 0, currPage->height / 2, currPage->width, "input.tif", panda_image_tiff);

SEE ALSO

panda_imageboxrot

panda_imageboxrot

Name

panda_imageboxrot — insert an image into the PDF document at the specified location

Synopsis

```
 #include<panda/constants.h>
 #include<panda/functions.h>
 void panda_panda_imageboxrot (panda_pdf * output, panda_page * target, int top, int le
tom, int right, double angle, char *filename, int type);
```

DESCRIPTION

This function call inserts an image into the PDF document at the specified location, including the ability to rotate the image on the page. It should be noted that xpdf will sometimes make the rotated image look quite sickly. This is in fact a bug in xpdf (which has beenr eported), and not a bug in **Panda**. The image types accepted by this call are: panda_image_tiff, panda_image_jpeg and panda_image_png.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *demo; panda_page *currPage; panda_init (); if ((demo = panda_open ("output.pdf", "w")) == NULL) panda_error (panda_true, "demo: could not open output.pdf to write to."); currPage = panda_newpage (demo, panda_pagesize_a4); panda_imagebox (demo, currPage, 0, 0, currPage->height / 2, currPage->width, 45.0, "input.tif", panda_image_tiff);

SEE ALSO

panda_imagebox

Vector graphics

Panda, supports the full range of PDF vector graphics commands. See the sections below for a description of the functionality available.

Lines

With Panda, the way you draw a line is to first create a line with **panda_setlinestart**, you then draw line segments with **panda_addlinesegment**. When you're finished, you close the line with **panda_closeline**. **panda_strokeline** is used to force the line to be drawn. You can also add curved segments to a line using **panda_addcubiccurvesegment**, **panda_addquadraticcurvesegmentone**, and **panda_addquadraticcurvesegmenttwo**. Finally, if all you want is a rectangle, then use **panda_rectangle** to do it for you.

panda_setlinestart

Name

`panda_setlinestart` — sets the starting point of a curve

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setlinestart (panda_page * target, int x, int y);
```

DESCRIPTION

Set the starting point for the sequence of curves and lines that it to be drawn on the current page. This call is compulsory for almost all of the line drawing functions. It is not required for the **panda_rectangle** call.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200);

SEE ALSO

panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_addlinesegment

Name

`panda_addlinesegment` — add a straight segment to the line shape we are drawing

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_addlinesegment (panda_page * target, int x, int y);
```

DESCRIPTION

Add a point to the shape we are currently drawing with a straight line between the current cursor location and (x,y).

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_addlinesegment (page, 200, 200);

SEE ALSO

panda_setlinestart, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_closeline

Name

`panda_closeline` — close off the line shape we are drawing

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_closeline(panda_page * target);
```

DESCRIPTION

Close the line shape we are drawing by returning to the starting point as set by **panda_setlinestart**();

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_addlinesegment (page, 200, 200); panda_addlinesegment (page, 400, 300); panda_closeline (page);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_endline

Name

`panda_endline` — finalise the current line shape

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_endline( panda_page *target);
```

DESCRIPTION

Finalise the line shape we are drawing. Only one line shape may be drawn at any one time. There is no need for this call with the **panda_rectangle**() call.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_addlinesegment (page, 200, 200); panda_endline (page);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_strokeline

Name

`panda_strokeline` — stroke the line shape we have just drawn

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_strokeline (panda_page * target);
```

DESCRIPTION

This function must be called for the line shape that we have drawn to actually display on the PDF page. This process is known as 'stroking', and hence the name of this function call.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_addlinesegment (page, 200, 200); panda_strokeline (page); panda_endline (page);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_addcubiccurvesegment

Name

`panda_addcubiccurvesegment` — add a curved segment to the line shape we are drawing

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_addcubiccurvesegment (panda_page * target, int x, int y, int cx1, int cy1,
```

DESCRIPTION

Add a point to the shape we are currently drawing with a cubic curve between the current cursor location and (x,y). There are two control points used to generate the cubic curve. They are (cx1, cy1) and (cx2, cy2).

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (currPage, 210, 210); panda_addcubiccurvesegment (currPage, 310, 210, 225, 300, 275, 400);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_addquadraticcurvesegmentone

Name

`panda_addquadraticcurvesegmentone` — add a curved segment to the line shape that we are drawing

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_addquadraticcurvesegmentone (panda_page * target, int x, int y, int cx1, in
```

DESCRIPTION

This function adds a curved segment to the line shape that we are drawing. The curved segment has a control point, namely (cx1, cy1). This call creates slightly different curves from **panda_addquadraticcurvesegmenttwo**();

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_addquadraticcurvesegmentone (page, 200, 200, 12, 32);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_addquadraticcurvesegmenttwo

Name

`panda_addquadraticcurvesegmenttwo` — add a curved segment to the line shape that we are drawing

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_addquadraticcurvesegmenttwo (panda_page * target, int x, int y, int cx1, in
```

DESCRIPTION

This function adds a curved segment to the line shape that we are drawing. The curved segment has a control point, namely (cx1, cy1). This call creates slightly different curves from **panda_addquadraticcurvesegmentone**();

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_addquadraticcurvesegmenttwo (page, 200, 200, 12, 32);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_rectangle

Name

`panda_rectangle` — draw a rectangle

Synopsis

```
 #include<panda/constants.h>
 #include<panda/functions.h>
 void panda_rectangle (panda_page * target, int top, int left, int bot-
tom, int right);
```

DESCRIPTION

Draw a rectangle on the PDF page. There is no need for the **panda_setlinestart**() or **panda_closeline**() calls.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_rectangle(page, 10, 10, 150, 200);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

Fills and other line attributes

Once you have drawn a line, then it can be filled. You can also configure the state of the pen *before* you draw the line to change the way it appears...

panda_fillline

Name

`panda_fillline` — fill the closed shape we just drew

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_fillline (panda_page * target);
```

DESCRIPTION

Fill the shape we have just drawn with the previously defined fill.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_addlinesegment (page, 200, 200); panda_addlinesegment (page, 250, 250); panda_endline (page); panda_fillline (page);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_setlinecap

Name

`panda_setlinecap` — sets the line cap for the lines we are drawing now

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setlinecap ( panda_page *target, int cap);
```

DESCRIPTION

A line cap is used at the ends of lines that do not meet other lines. The different cap styles are defined in panda/constants.h and are: panda_linecap_butt, panda_linecap_round and panda_linecap_projectedsquare.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_setlinecap (page, panda_linecap_round); panda_addlinesegment (page, 200, 200);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_setlinecolor

Name

`panda_setlinecolor` — change the color of the line drawn

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setlinecolor (panda_page *target, int red, int green, int blue);
```

DESCRIPTION

Set the color of lines being drawn using a combination of red, green and blue.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_setlinecolor (page, 100, 200, 450); panda_addlinesegment (page, 200, 200);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor

panda_setlinejoin

Name

`panda_setlinejoin` — is used to set the line join style

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setlinejoin (panda_page *target, int join);
```

DESCRIPTION

A line join is used where the ends of two lines meet. The valid line joins are defined in panda/constants.h and are: panda_linejoin_miter, panda_linejoin_round and panda_linejoin_bevel.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_setlinejoin (page, panda_linejoin_bevel); panda_addlinesegment (page, 200, 200);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setfillcolor, panda_setlinecolor

panda_setlinewidth

Name

`panda_setlinewidth` — sets the width of the line that we are drawing

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setlinewidth (panda_page * target, int width);
```

DESCRIPTION

Set the width of the line that is being drawn... You can use **panda_setlinecap**(), **panda_setlinejoin**() and **panda_setlinedash**() to change other characteristics of the line.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_setlinewidth (page, 42); panda_addlinesegment (page, 200, 200);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinecap, panda_setlinejoin, panda_setlinedash, panda_setfillcolor, panda_setlinecolor

panda_setlinedashing

Name

`panda_setlinedashing` — draw the subsequent lines with the defined dashing pattern

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setlinedashing (panda_page *target, int on, int off, int pahse);
```

DESCRIPTION

This function allows the user to define a line dashing style, which is then applied to subsequent lines drawn on that page. The dashing style is defined as a on and off number, as well as a phase. For example, on = 2, off = 4, phase = 0 should result in a line like:

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_setlinestart (page, 100, 200); panda_setlinedash (page, 2, 4, 0); panda_addlinesegment (page, 200, 200);

SEE ALSO

panda_setlinestart, panda_addlinesegment, panda_addcubiccurvesegment, panda_addquadraticsegmentone, panda_addquadraticcurvesegmenttwo, panda_closeline, panda_rectangle, panda_endline, panda_strokeline, panda_fillline, panda_setlinewidth, panda_setlinecap, panda_setlinejoin, panda_setfillcolor, panda_setlinecolor

Document meta data

PDF supports embedding meta data about a document into the PDF itself, and Panda supports this through the following calls:

panda_setauthor

Name

`panda_setauthor` — set the author string for the PDF document

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setauthor (panda_pdf *output, char *value);
```

DESCRIPTION

This function sets the value of the author within the PDF document.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_setauthor(document, "Mikal");

SEE ALSO

panda_checkinfo, panda_setcreator, panda_settitle, panda_setsubject, panda_setkeywords, panda_setid

panda_setkeywords

Name

`panda_setkeywords` — set the keywords string for the PDF document

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setkeywords (panda_pdf *output, char *value);
```

DESCRIPTION

This function sets the value of the keywords string within the PDF document. The string is merely a list of keywords in the form "cats dogs hamsters chickens"

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_setkeywords(document, "panda documentation pdf api generate");

SEE ALSO

panda_checkinfo, panda_setauthor, panda_setcreator, panda_settitle, panda_setsubject, panda_setid

panda_setsubject

Name

`panda_setsubject` — set the subject string for the PDF document

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_setsubject (panda_pdf *output, char *value);
```

DESCRIPTION

This function sets the value of the subject within the PDF document.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_setsubject(document, "Mikal's homework");

SEE ALSO

panda_checkinfo, panda_setauthor, panda_setcreator, panda_settitle, panda_setkeywords, panda_setid

panda_settitle

Name

`panda_settitle` — set the title string for the PDF document

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_settitle (panda_pdf *output, char *value);
```

DESCRIPTION

This function sets the value of the title within the PDF document.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_settitle(document, "Mikal's excellent PDF document");

SEE ALSO

panda_checkinfo, panda_setcreator, panda_setauthor, panda_setsubject, panda_setkeywords, panda_setid

Presentation support

PDF can be used for presentations, and therefore allows you to specify a number of interesting things which are normally associated more with Microsoft Power Point presentations...

panda_centerwindow

Name

panda_centerwindow — ask the viewer to center the document's window on the screen when the PDF is displayed

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_centerwindow (panda_pdf *document, int onoroff);
```

DESCRIPTION

This function records information in the output PDF document requesting that the viewing application center the displayed PDF document on the screen when it is opened. This option is not supported by all viewers, and therefore should not be relied upon. The on or off arguement is a panda_true value, which does the obvious thing. The default is to not center the window.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_centerwindow(document, panda_true);

panda_fitwindow

Name

panda_fitwindow — ask the viewer to fit the viewer window to the first page of the PDF document when it is opened

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_fitwindow (panda_pdf *document, int onoroff);
```

DESCRIPTION

This function records information in the output PDF document requesting that the viewing application fit the display window to the first page of the PDF document when it is opened. This option is not supported by all viewers, and therefore should not be relied upon. The on or off arguement is a panda_true value, which does the obvious thing. The default is to not fit the document to the window.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_fitwindow(document, panda_true);

panda_fullscreen

Name

`panda_fullscreen` — ask the viewer to display the PDF document in fullscreen mode

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_fullscreen (panda_pdf *document, int onoroff);
```

DESCRIPTION

This function records information in the output PDF document requesting that the viewing application display the document in full screen mode. This option is not supported by all viewers, and therefore should not be relied upon. The on or off arguement is a panda_true value, which does the obvious thing. The default is to not display the PDF in fullscreen mode.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_fullscreen(document, panda_true);

panda_hidemenubar

Name

panda_hidemenubar — ask the viewer to hide it's menu bar when this PDF is displayed

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_hidemenubar (panda_pdf *document, int onoroff);
```

DESCRIPTION

This function records information in the output PDF document requesting that the viewing application's menu bar not be displayed when this PDF is opened. This option is not supported by all viewers, and therefore should not be relied upon. The on or off arguement is a panda_true value, which does the obvious thing. The default is to show the menu bar.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_hidemenubar(document, panda_true);

panda_hidetoolbar

Name

panda_hidetoolbar — ask the viewer to hide it's tool bar when this PDF is displayed

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_hidetoolbar (panda_pdf *document, int onoroff);
```

DESCRIPTION

This function records information in the output PDF document requesting that the viewing application's tool bar not be displayed when this PDF is opened. This option is not supported by all viewers, and therefore should not be relied upon. The on or off arguement is a panda_true value, which does the obvious thing. The default is to show the tool bar.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_hidetoolbar(document, panda_true);

panda_hidewindowui

Name

`panda_hidewindowui` — ask the viewer to hide it's display window user interface when this PDF is displayed

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_hidewindowui (panda_pdf *document, int onoroff);
```

DESCRIPTION

This function records information in the output PDF document requesting that the viewing application's window user interface not be displayed when this PDF is opened. This option is not supported by all viewers, and therefore should not be relied upon. The on or off arguement is a panda_true value, which does the obvious thing. The default is to display the user interface.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_hidewindowui(document, panda_true);

panda_nfspagemodde

Name

panda_nfspagemodde — defines display characteristics for the PDF document if it is using non fullscreen mode after defaulting to fullscreen mode

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_nfspagemode (panda_pdf *document, int pagemode);
```

DESCRIPTION

If the document in question is using fullscreen mode and then exits from fullscreen mode, then this function configures the behaviour of several of the 'eye candy' options available in some viewers. The possible values for pagemode are: **panda_window_usenone**, which displays neither the outline or thumbnails (if present); **panda_window_useoutlines**, which displays only the outline for the document; **panda_window_usethumbs**, which only displays thumbnails. **Please note that this function will only have an effect on the viewer if the page mode has been set to fullscreen with the panda_fullscreen() function call**

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_init(); document = panda_open("filename.pdf", "w"); panda_nfspagemode(document, panda_window_usenone);

SEE ALSO

panda_fullscreen

panda_pageduration

Name

panda_pageduration — specify the maximum number of seconds that a page should be displayed by the viewer before moving on

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_pageduration (panda_page *target, int seconds);
```

DESCRIPTION

This function records information within the PDF indicating the maximum number of seconds that the given page should be displayed within the viewer. This is useful for presentations and the like where you might like to automatically move onto the next page in the document at some point. The default value for this is to never move onto the next page automatically. If this value is changed from the default, there is currently no way to revert back to the default later. The feature may not be implemented by all viewers.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_pageduration (page, 30.5);

panda_transduration

Name

`panda_transduration` — specify the number of seconds that a page transition effect should take to occur

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_transduration (panda_page *target, double seconds);
```

DESCRIPTION

This function records information within the PDF indicating the maximum number of seconds that the given page transition effect should be displayed within the viewer. This is useful for presentations and the like when you realise that you are addicted to Microsoft Powerpoint...

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_transduration (page, 30.5);

panda_transstyle

Name

`panda_transstyle` — specify the type of page change transition that should occur

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_transstyle (panda_page *target, int style);
```

DESCRIPTION

his function records information within the PDF indicating the preferred page transition style to use. The following are valid styles to use:

1. panda_pagetrans_split_yi -- vertical split from the inside of the page
2. panda_pagetrans_split_yo -- vertical split from the outside of the page
3. panda_pagetrans_split_xi -- horizontal split from the inside of the page
4. panda_pagetrans_split_xo -- horizontal split from the outside of the page
5. panda_pagetrans_blinds_y -- vertical blinds effect
6. panda_pagetrans_blinds_x -- horizontal blinds effect
7. panda_pagetrans_box_i -- box expanding from the inside of the page
8. panda_pagetrans_box_o -- box contracting from the outside of the page
9. panda_pagetrans_wipe_0 -- a single line wipes the page away from the left to the right
10. panda_pagetrans_wipe_90 -- a single line wipes the page away from the bottom to the top
11. panda_pagetrans_wipe_180 -- a single line wipes the page away from the right to the left
12. panda_pagetrans_wipe_270 -- a single line wipes the page away from the top to the bottom
13. panda_pagetrans_dissolve -- the old page dissolves slowly into the new page

14. panda_pagetrans_glitter_0 -- a glitter effect that moves from the left to the right of the page
15. panda_pagetrans_glitter_270 -- a glitter effect that moves from the top to the bottom of the page
16. panda_pagetrans_glitter_315 -- a glitter effect that moves from the top left to the bottom right of the page
17. panda_pagetrans_none -- no transition effect

he default transition is to have no transition at all. It should be noted that not all viewers support these transition effects.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *page; panda_init(); document = panda_open("filename.pdf", "w"); page = panda_newpage (document, panda_pagesize_a4); panda_transduration (page, 30.5);

Page templates

A page template is created using **panda_newtemplate**. A template is just like any other page, and uses all the normal drawing functions. The cool bit is that you can then apply a template to another page using **panda_applytemplate**. This allows for the standard parts of the page to only be defined once -- which is very useful for things like letterhead.

panda_newtemplate

Name

`panda_newtemplate` — create a template page in the PDF

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_newtemplate(panda_pdf *document, char *pageSize);
```

DESCRIPTION

This function is used to create 'template' pages which can then be refered to on other pages. For instance, if you were creating a document that used a standard letter head, then it would make sense to construct the letterhead as a template, and then use this on all the pages. The created template looks and feels just like any other page in the

document for the purposes of creating content. Refer to the **panda_newpage** man page for details on how to use pages.

RETURNS

A pointer to the template page

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *templatepage; panda_init(); document = panda_open("filename.pdf", "w"); templatepage = panda_newtemplate (document, panda_pagesize_a4);

SEE ALSO

panda_newpage, panda_applytemplate

panda_applytemplate

Name

`panda_applytemplate` — use a template page previously created

Synopsis

```
#include<panda/constants.h>
#include<panda/functions.h>
void panda_applytemplate (panda_pdf * output, panda_page * target,
panda_page * template)
```

DESCRIPTION

This function is used to use a template created with the **panda_newtemplate** function call.

RETURNS

Nothing

EXAMPLE

#include<panda/constants.h> #include<panda/functions.h> panda_pdf *document; panda_page *templatepage, *realpage; panda_init(); document = panda_open("filename.pdf", "w"); templatepage = panda_newtemplate (document, panda_pagesize_a4); realpage = panda_newpage (document, panda_pagesize_a4); ... the order of the drawing commands to the two pages doesn't matter ... panda_applytemplate(document, realpage, templatepage);

SEE ALSO

panda_newpage, panda_newtemplate

A full Panda example

This section presents a full Panda example showing what the library is capable of. This code comes from the Panda distribution:

```
#include  <panda/functions.h>
#include  <panda/constants.h>

int
main (int argc, char *argv[])
{
  panda_pdf *demo;
  panda_page *currPage, *templatePage;
  int lineDepth, trans;
  char tempString[1024], *tempPtr;

  printf ("Welcome to the Panda 0.4 sample application...\n");

  // Initialise the library
  panda_init ();

  // Open our demo PDF
  if ((demo = panda_open ("output.pdf", "w")) == NULL)
    panda_error (panda_true, "demo: could not open output.pdf to write to.");

  // These are normally commented out because they are annoying
  //panda_hidetoolbar (demo, panda_true);
  //panda_hidemenubar (demo, panda_true);
  //panda_hidewindowui (demo, panda_true);

  if ((argc > 1) && (strcmp (argv[1], "compressed") == 0))
    {
      printf ("Turn on compression\n");

      panda_setproperty (demo->pages, panda_scope_cascade,
             panda_object_property_compress, panda_true);

      printf ("Just before compression level set\n");

      panda_setproperty (demo->pages, panda_scope_cascade,
             panda_object_property_compress_level, 9);
    }
  else
    printf ("For compressed sample, use %s compressed\n", argv[0]);

  // Image functionality
  ////////////////////////////////////////////////////////////////////////

  // Create a page
  currPage = panda_newpage (demo, panda_pagesize_a4);

  // Put in the background images
  panda_imagebox (demo, currPage, 0, 0, currPage->height / 2,
          currPage->width, "images/input.tif", panda_image_tiff);
  panda_imagebox (demo, currPage, currPage->height / 2, 0,
          currPage->height, currPage->width, "images/input2.tif",
          panda_image_tiff);
```

```
  panda_imagebox (demo, currPage, 317, 317, 434, 434, "images/gnu_box.jpg",
          panda_image_jpeg);
  panda_imagebox (demo, currPage, 434, 434, 551, 551, "images/gnu_box.jpg",
          panda_image_jpeg);

  // Do an panda_imageboxrot or two to test the code included by Ceasar Miquel
  panda_imageboxrot (demo, currPage, 600, 0, 717, 117, 15.0,
             "images/gnu_box.jpg", panda_image_jpeg);

  panda_imageboxrot (demo, currPage, 600, 200, 717, 317, 30.0,
             "images/gnu_box.jpg", panda_image_jpeg);

  panda_imageboxrot (demo, currPage, 600, 400, 717, 517, 42.0,
             "images/gnu_box.jpg", panda_image_jpeg);

  panda_imageboxrot (demo, currPage, 700, 0, 817, 117, 90.0,
             "images/gnu_box.jpg", panda_image_jpeg);

  panda_imageboxrot (demo, currPage, 700, 200, 817, 317, 132.0,
             "images/gnu_box.jpg", panda_image_jpeg);

  // Insert a PNG to show that I can
  panda_imageboxrot (demo, currPage, 100, 200, 200, 300, 0.0,
             "images/libpng.png", panda_image_png);

  panda_imageboxrot (demo, currPage, 300, 200, 400, 300, 0.0,
             "images/gnu.png", panda_image_png);

  panda_imageboxrot (demo, currPage, 100, 420, 310, 620, 36.0,
             "images/RedbrushAlpha.png", panda_image_png);

  // (c) statement
  panda_setfont (demo, tempPtr = panda_createfont (demo, "Times-Roman", 1,
                            "MacRomanEncoding"));
  panda_textbox (demo, currPage, 600, 10, 700, 300,
         "The background image on this page is Copyright 2000 Andrew Cagney");
  panda_textbox (demo, currPage, 620, 10, 720, 300,
         "and is distributed under the terms of the GPL...");
  panda_textbox (demo, currPage, 310, 320, 330, 800,
         "Flower (c) 1999 Pieter S. van der Meulen");
  free (tempPtr);

  panda_setfont (demo, tempPtr =
         panda_createfont (demo, "Helvetica-Bold", 1,
                   "MacRomanEncoding"));
  panda_textboxrot (demo, currPage, 200, 30, 230,
         30, 45.0, "With new improved angled text!");
  free (tempPtr);

  ////////////////////////////////////////////////////////////////////////
  // Text functionality (with a few images thrown in as well)
  ////////////////////////////////////////////////////////////////////////

  currPage = panda_newpage (demo, panda_pagesize_a4);

  // I am not drawing a multiline string here because I am not sure how to
  // represent this in the PDF at the moment
  sprintf (tempString,
       "Hello %c5World! %cMy name %c5is Panda!\nAnd I am a PDF gener-
ator\nI handle multiple line text ok .once you have set a leading.",
       4, 6, 5);
  panda_textbox (demo, currPage, 10, 10, 100, 30, tempString);

  panda_setfont (demo, tempPtr = panda_createfont (demo, "Symbol", 1,
                            "MacRomanEncoding"));
  panda_textbox (demo, currPage, 50, 10, 100, 30, "Symbol");
```

```
  free (tempPtr);

  panda_setfont (demo, tempPtr =
         panda_createfont (demo, "Helvetica-Bold", 1,
                    "MacRomanEncoding"));
  panda_textbox (demo, currPage, 70, 30, 100, 30, "A line in Helvetica-
Bold");
  free (tempPtr);

  panda_imagebox (demo, currPage, 100, 100, 150, 150, "images/gnu-head.jpg",
          panda_image_jpeg);
  panda_textbox (demo, currPage, 90, 110, 200, 200, "INFRONTINFRONTINFRONT");

  panda_textbox (demo, currPage, 190, 210, 300, 300, "BEHINDBEHINDBEHIND");
  panda_imagebox (demo, currPage, 200, 200, 317, 317, "images/gnu_box.jpg",
          panda_image_jpeg);

  panda_textbox (demo, currPage, 300, 10, 400, 50,
         "A second textbox on the page");

  ////////////////////////////////////////////////////////////////////////
  // Demonstrate the supported text modes
  ////////////////////////////////////////////////////////////////////////

  currPage = panda_newpage (demo, panda_pagesize_a4);
  panda_setleading (demo, 16.0);

  for (lineDepth = 0; lineDepth < 8; lineDepth++)
    {
      panda_setfontmode (demo, panda_textmode_normal);

      switch (lineDepth)
    {
    case panda_textmode_normal:
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 10,
             40 + (lineDepth * 20), 400, "Normal");
      break;

    case panda_textmode_outline:
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 10,
             40 + (lineDepth * 20), 400, "Outline");
      break;

    case panda_textmode_filledoutline:
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 10,
             40 + (lineDepth * 20), 400, "FilledOutline");
      break;

    case panda_textmode_invisible:
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 10,
             40 + (lineDepth * 20), 400, "Invisible");
      break;

    case panda_textmode_filledclipped:
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 10,
             40 + (lineDepth * 20), 400, "FilledClipped");
      break;

    case panda_textmode_strokedclipped:
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 10,
             40 + (lineDepth * 20), 400, "Stroked Clipped");
      break;

    case panda_textmode_filledstrokedclipped:
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 10,
             40 + (lineDepth * 20), 400,
```

```
            "Filled Stroked Clipped");
      break;

    case panda_textmode_clipped:
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 10,
            40 + (lineDepth * 20), 400, "Clipped");
      break;
    }

      panda_setcharacterspacing (demo, (double) lineDepth);
      panda_setwordspacing (demo, (double) lineDepth * 10);
      panda_sethorizontalscaling (demo, (double) 1 - (lineDepth * 0.1));

      panda_setfontmode (demo, lineDepth);
      panda_textbox (demo, currPage, 20 + (lineDepth * 20), 200,
            40 + (lineDepth * 20), 400,
            "Demonstration of a text mode");
    }

////////////////////////////////////////////////////////////////////////
// Demonstrate the supported lines and curve thingies -- note that no
// graphics state is held from the previous set of lines, so you'll need
// to rebuild it each time.
////////////////////////////////////////////////////////////////////////

currPage = panda_newpage (demo, panda_pagesize_a4);

panda_setfontmode (demo, panda_textmode_normal);
panda_textbox (demo, currPage, 40, 10, 55, 200,
       "Please note that these shapes are lines, and there is no");
panda_textbox (demo, currPage, 60, 10, 75, 200,
       "requirement to have the shapes closed...");

// Straight lines of all types -- stroked
panda_setlinestart (currPage, 110, 110);
panda_addlinesegment (currPage, 160, 130);
panda_addlinesegment (currPage, 210, 186);
panda_addlinesegment (currPage, 96, 22);
panda_closeline (currPage);
panda_strokeline (currPage);
panda_endline (currPage);

// Now some curves -- stroked
panda_setlinestart (currPage, 210, 210);
panda_addcubiccurvesegment (currPage, 310, 210, 225, 300, 275, 400);
panda_addquadraticcurvesegmentone (currPage, 160, 160, 200, 225);
panda_addquadraticcurvesegmenttwo (currPage, 210, 210, 250, 375);
panda_closeline (currPage);
panda_strokeline (currPage);
panda_endline (currPage);

// Rectangles -- stroked
panda_rectangle (currPage, 210, 210, 310, 310);
panda_strokeline (currPage);
panda_endline (currPage);

// Straight lines of all types -- stroked and filled
panda_setlinecolor (currPage, 99, 33, 255);
panda_setfillcolor (currPage, 112, 38, 300);
panda_setlinestart (currPage, 110, 310);
panda_setlinewidth (currPage, 5);
panda_addlinesegment (currPage, 160, 330);
panda_addlinesegment (currPage, 210, 386);
panda_addlinesegment (currPage, 96, 222);
panda_closeline (currPage);
panda_fillline (currPage);
```

```
panda_endline (currPage);

// Now some curves -- stroked and filled
panda_setfillcolor (currPage, 112, 138, 37);
panda_setlinestart (currPage, 210, 410);
panda_setlinewidth (currPage, 5);
panda_addcubiccurvesegment (currPage, 310, 410, 225, 500, 275, 600);
panda_addquadraticcurvesegmentone (currPage, 160, 360, 200, 425);
panda_addquadraticcurvesegmenttwo (currPage, 210, 410, 250, 575);
panda_closeline (currPage);
//panda_strokeline (currPage);
panda_fillline (currPage);
panda_endline (currPage);

// Rectangles -- stroked filled
panda_setfillcolor (currPage, 38, 38, 38);
panda_setlinewidth (currPage, 5);
panda_rectangle (currPage, 410, 210, 510, 310);
//panda_strokeline (currPage);
panda_fillline (currPage);
panda_endline (currPage);

// Straight lines of all types -- stroked and capped
panda_setlinewidth (currPage, 10);
panda_setlinestart (currPage, 100, 600);
panda_addlinesegment (currPage, 200, 600);
panda_strokeline (currPage);
panda_endline (currPage);

panda_setlinewidth (currPage, 10);
panda_setlinecap (currPage, panda_linecap_butt);
panda_setlinestart (currPage, 100, 625);
panda_addlinesegment (currPage, 200, 625);
panda_strokeline (currPage);
panda_endline (currPage);

panda_setlinewidth (currPage, 10);
panda_setlinecap (currPage, panda_linecap_round);
panda_setlinestart (currPage, 100, 650);
panda_addlinesegment (currPage, 200, 650);
panda_strokeline (currPage);
panda_endline (currPage);

panda_setlinewidth (currPage, 10);
panda_setlinecap (currPage, panda_linecap_projectedsquare);
panda_setlinestart (currPage, 100, 675);
panda_addlinesegment (currPage, 200, 675);
panda_strokeline (currPage);
panda_endline (currPage);

// Mitre joints
panda_setlinewidth (currPage, 10);
panda_setlinecap (currPage, panda_linecap_butt);
panda_setlinestart (currPage, 300, 600);
panda_addlinesegment (currPage, 350, 650);
panda_addlinesegment (currPage, 400, 600);
panda_strokeline (currPage);
panda_endline (currPage);

panda_setlinewidth (currPage, 10);
panda_setlinejoin (currPage, panda_linejoin_miter);
panda_setlinestart (currPage, 300, 625);
panda_addlinesegment (currPage, 350, 675);
panda_addlinesegment (currPage, 400, 625);
panda_strokeline (currPage);
panda_endline (currPage);
```

```
panda_setlinewidth (currPage, 10);
panda_setlinejoin (currPage, panda_linejoin_round);
panda_setlinestart (currPage, 300, 650);
panda_addlinesegment (currPage, 350, 700);
panda_addlinesegment (currPage, 400, 650);
panda_strokeline (currPage);
panda_endline (currPage);

panda_setlinewidth (currPage, 10);
panda_setlinejoin (currPage, panda_linejoin_bevel);
panda_setlinestart (currPage, 300, 675);
panda_addlinesegment (currPage, 350, 725);
panda_addlinesegment (currPage, 400, 675);
panda_strokeline (currPage);
panda_endline (currPage);

// Do some work with line dashing
panda_setlinedash (currPage, 1, 0, 0);
panda_setlinejoin (currPage, panda_linejoin_round);

panda_setlinestart (currPage, 100, 800);
panda_addlinesegment (currPage, 100, 750);
panda_addlinesegment (currPage, 140, 800);
panda_closeline (currPage);
panda_strokeline (currPage);
panda_endline (currPage);

panda_setlinedash (currPage, 3, 3, 0);
panda_setlinestart (currPage, 150, 800);
panda_addlinesegment (currPage, 150, 750);
panda_addlinesegment (currPage, 190, 800);
panda_closeline (currPage);
panda_strokeline (currPage);
panda_endline (currPage);

panda_setlinedash (currPage, 2, 1, 1);
panda_setlinestart (currPage, 200, 800);
panda_addlinesegment (currPage, 200, 750);
panda_addlinesegment (currPage, 240, 800);
panda_closeline (currPage);
panda_strokeline (currPage);
panda_endline (currPage);

////////////////////////////////////////////////////////////////////////
// Why not have some annotations as well?
////////////////////////////////////////////////////////////////////////

panda_textannotation(demo, currPage, panda_true, "Hello", "Text annotation",
            10, 10, 50, 50, 1.0, 0.0, 0.0,
            panda_icon_comment, 0);
panda_lineannotation(demo, currPage, "Line annotation", "Line annotation",
            50, 50, 150, 150,
            150, 150, 200, 200,
            0.0, 1.0, 0.0, 0);

////////////////////////////////////////////////////////////////////////
// We can also setup template pages to make life a little easier (and the
// document a little smaller)
////////////////////////////////////////////////////////////////////////

templatePage = panda_newtemplate(demo, panda_pagesize_a4);
panda_setlinestart (templatePage, 100, 800);
panda_addlinesegment (templatePage, 100, 750);
panda_addlinesegment (templatePage, 140, 800);
panda_closeline (templatePage);
```

```
  panda_strokeline (templatePage);
  panda_endline (templatePage);

  panda_setlinedash (templatePage, 3, 3, 0);
  panda_setlinestart (templatePage, 150, 800);
  panda_addlinesegment (templatePage, 150, 750);
  panda_addlinesegment (templatePage, 190, 800);
  panda_closeline (templatePage);
  panda_strokeline (templatePage);
  panda_endline (templatePage);

  currPage = panda_newpage(demo, panda_pagesize_a4);
  panda_applytemplate(demo, currPage, templatePage);

  ////////////////////////////////////////////////////////////////////////
  // Let's try some transitions
  ////////////////////////////////////////////////////////////////////////

  for(trans = 0; trans < panda_pagetrans_none; trans++)
    {
      currPage = panda_newpage(demo, panda_pagesize_a4);
      panda_pageduration(demo, currPage, 5);
      panda_transduration(demo, currPage, 5.0);
      panda_transstyle(demo, currPage, trans);

      panda_setlinecolor (currPage, trans * 20, 0, trans * 10);
      panda_setfillcolor (currPage, trans * 20, 0, trans * 10);
      panda_setlinestart (currPage, 0, 0);
      panda_setlinewidth (currPage, 5);
      panda_addlinesegment (currPage, 1000, 0);
      panda_addlinesegment (currPage, 1000, 1000);
      panda_addlinesegment (currPage, 0, 1000);
      panda_closeline (currPage);
      panda_fillline (currPage);
      panda_endline (currPage);
    }

  currPage = panda_newpage(demo, panda_pagesize_a4);

  // Finished all the demoing, close the PDF document
  panda_close (demo);

  // We should return a value here
  return 0;
}

// Allow a callback to be setup to display a dialog box for an error or
// whatever before we terminate the application
void
panda_errorCallback (char *description)
{
  fprintf (stderr, "Callback: %s\n", description);
}
```

Code: main.c

It produces output like:

Figure 6-3. ~mikal/opensource-unstable/panda/examples/output.pdf

A lexer for PDF

Part of my Panda work has been the development of a lexer for PDF documents known as Panda*Lex*. Whilst Panda*Lex* is by no means complete, it might prove to be useful for your understanding of the PDF document structure. I have therefore included it here. The lexer takes the form of several components: a lexer for the grammar, a grammar, and some c code to hold it all together.

lexer.l

```
%{
#include "parser.h"
#include "pandalex.h"
#include <unistd.h>
#include <stdlib.h>

#undef YY_INPUT
#define YY_INPUT(b, r, ms) (r = pandalex_gettext(b, ms))

void *
pandalex_xmalloc (size_t size)
{
  void *buffer;

  if ((buffer = malloc (size)) == NULL)
    {
      pandalex_error ("pandalex_xmalloc failed to allocate memory");
    }

  return buffer;
}
void *
pandalex_xrealloc (void *memory, size_t size)
{
  void *buffer;

  if ((buffer = realloc (memory, size)) == NULL)
    {
      pandalex_error ("pandalex_xrealloc failed to allocate memory");
    }
```

```
  return buffer;
}

char *
pandalex_xsnprintf (char *format, ...)
{
  char *output = NULL;
  int size, result;
  va_list ap;

  /* We start with the size of the format string as a guess */
  size = strlen (format);
  va_start (ap, format);

  while (1)
    {
      output = pandalex_xrealloc (output, size);
      result = vsnprintf (output, size, format, ap);

      if (result == -1)
        {
          /* Up to glibc 2.0.6 and Microsoft's implementation*/
          size += 100;
        }
      else
        {
          /* Check if we are done */
      if (result < size)
            break;

          /* Glibc from now on */
          size = result + 1;
        }
  }

  va_end (ap);
  return output;
}

void
pandalex_error(char *msg){
fprintf(stderr, "%s\n", msg);
exit(42);
}
%}

%x BINARY
%%

<INITIAL>\%PDF-[0-9]+\.[0-9]+    { debuglex(yytext, -1, "version");
                            yylval.sval.data = (char *) returnStr(yytext, -
1);
                            yylval.sval.len = yyleng;
                            return PDFVER;
                                                                    }

<INITIAL>[ \t\r\n]+              { debuglex(yytext, -1, "whitespace");
                                                                  }

<INITIAL>xref                    { debuglex(yytext, -1, "xref");
                            yylval.sval.data = (char *) returnStr(yytext + 1, -
1);
                            yylval.sval.len = yyleng - 1;
                            return XREF;
                                                                    }
```

```
<INITIAL>trailer                { debuglex(yytext, -1, "trailer");
                            yylval.sval.data = (char *) returnStr(yytext + 1, -
1);
                            yylval.sval.len = yyleng - 1;
                            return TRAILER;
                                                                        }

<INITIAL>\/[#a-zA-Z\-_0-9\.\+,]+ { debuglex(yytext, -1, "name");
                            yylval.sval.data = (char *) returnStr(yytext + 1, -
1);
                            yylval.sval.len = yyleng - 1;
                            return NAME;
                                                                        }

<INITIAL>\<\<\/[#a-zA-Z\-_0-9\.\+]+ { debuglex(yytext, -1, "dbllt-name");
                            yyless(2);
                            return DBLLT;
                                                                        }

<INITIAL>\/[#a-zA-Z\-_0-9\.\+]+\>\>     { debuglex(yytext, -1, "name-
dblgt");
                            yyless(yyleng - 2);
                            yylval.sval.data = (char *) returnStr(yytext + 1, -
1);
                            yylval.sval.len = yyleng - 1;
                            return NAME;
}

                        /* --- stuff required for objects ---              */
<INITIAL>R                      { debuglex(yytext, -1, "object reference");
                            yylval.sval.data = (char *) returnStr(yytext, -
1);
                            yylval.sval.len = yyleng;
                            return OBJREF;
                                                                        }

<INITIAL>R\>\>                          { debuglex(yytext, -1, "object-
reference-dblgt");
                            yyless(yyleng - 2);
                            yylval.sval.data = (char *) returnStr(yytext, -
1);
                            yylval.sval.len = yyleng;
                            return OBJREF;
                                                                        }

<INITIAL>R\/                       { debuglex(yytext, -1, "object-reference-
namestart");
                            yyless(yyleng - 1);
                            yylval.sval.data = (char *) returnStr(yytext, -
1);
                            yylval.sval.len = yyleng;
                            return OBJREF;
                                                                        }

<INITIAL>obj                    { debuglex(yytext, -1, "obj");
                            yylval.sval.data = (char *) returnStr(yytext, -
1);
                            yylval.sval.len = yyleng;
                            return OBJ;
                                                                        }

<INITIAL>endobj                 { debuglex(yytext, -1, "endobj");
                            yylval.sval.data = (char *) returnStr(yytext, -
1);
                            yylval.sval.len = yyleng;
```

```
                            return ENDOBJ;
                                                                    }

<INITIAL>stream                 { debuglex(yytext, -1, "stream");
                          BEGIN(BINARY);
                          return STREAM;                            }

<INITIAL>[+-]?[0-9]+            { debuglex(yytext, -1, "integer");
                            yylval.intVal = atoi(yytext);
                            return INT;
                                                                    }

<INITIAL>[+-]?[0-9]+\>\>            { debuglex(yytext, -1, "integer-
dblgt");
                            yyless(yyleng - 2);
                            yylval.intVal = atoi(yytext);
                            return INT;
                                                                    }


<INITIAL>[+-]?[0-9]+\.[0-9]+    { debuglex(yytext, -1, "floating point");
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return FP;
                                                                    }

<INITIAL>[+-]?[0-9]+\.[0-9]+\>\>    { debuglex(yytext, -1, "floating point");
                            yyless(yyleng - 2);
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return FP;
                                                                    }

<INITIAL>\<\<                  { debuglex(yytext, -1, "<<");
                            return DBLLT;
                                                                    }

<INITIAL>\>\>                  { debuglex(yytext, -1, ">>");
                            return DBLGT;
                                                                    }

<INITIAL>\>\>\>\>                   { debuglex(yytext, -1, ">>>>");
                            yyless(yyleng - 2);
                            return DBLGT;
                                                                    }


<INITIAL,BINARY>([#a-zA-Z0-9\.:'+\-!_]+[a-zA-Z0-9\.:'+\-!_\\]*)+ {
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;

                            if(strcmp(yytext, "endstream") == 0){
                            debuglex(yytext, -1, "string (at the end of a stream)");
                              BEGIN(INITIAL);
                              return ENDSTREAM;
                              }
                            else debuglex(yytext, -1, "string");

                            return STRING;
                                                                    }

<INITIAL,BINARY>\[(([^\[\]])|(\\\[)|(\\\]))*\] {
                          debuglex(yytext, -1, "bracketted string v1");
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return STRING;
```

```
                                                                    }

<INITIAL,BINARY>\((([^\(\)])|(\\\()|(\\\)))*\) {
                          debuglex(yytext, -1, "bracketted string v2");
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return STRING;
                                                                    }

<INITIAL,BINARY>\<(([^\<\>])|(\\\<)|(\\\>))*\> {
                          debuglex(yytext, -1, "bracketted string v3");
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return STRING;
                                                                    }

<INITIAL,BINARY>([#a-zA-Z0-9\.:'+\-!_]+[a-zA-Z0-9\.:'+\-!_\\\/]*)+\>\> {
                          debuglex(yytext, -1, "string-dblgt");
                            yyless(yyleng - 2);
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return STRING;
                                                                    }

<INITIAL,BINARY>[\[\(][#<>a-zA-Z0-9\.\ :'+\-_\\\(\)]+[\)\]]\>\> {
                          debuglex(yytext, -1, "bracketted-string-dblgt");
                            yyless(yyleng - 2);
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return STRING;
                                                                    }

                        /* --- Array handling ---                          */

<INITIAL>\[                     { debuglex(yytext, -1, "[");
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return ARRAY;
                                                                    }

<INITIAL>\]                     { debuglex(yytext, -1, "]");
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return ENDARRAY;
                                                                    }

<INITIAL>\]\>\>                       { debuglex(yytext, -1, "]");
                            yyless(yyleng - 2);
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return ENDARRAY;
                                                                    }

                        /* --- Stuff needed for the xref and trailer -
--    */

<INITIAL>\%\%EOF                { debuglex(yytext, -1, "end of file");
                            return PDFEOF;
                                                                    }

                        /* --- Stuff used to match binary streams ---


                              The following amazing production is
                              used to deal with the massive
                              streams that images can create              */
```

```
<BINARY>[^end]+           { debuglex(yytext, yyleng, "binary mode");
                          yylval.sval.data = (char *) returnStr(yytext, yyleng);
                          yylval.sval.len = yyleng;
                          return ANYTHING;
                                                                          }

.                       { debuglex("!", -1, "the catch all");
                            yylval.sval.data = (char *) returnStr(yytext, yyleng);
                            yylval.sval.len = yyleng;
                            return ANYTHING;
                                                                          }

%%

void debuglex(char *text, int len, char *desc){
#if defined DEBUG
  int i;

  printf("Lexer rule is \"%s\", match is \"", desc);

  for(i = 0; i < ((len == -1) ? strlen(text) : len); i++){
    if(text[i] == '\n') printf(" \\n " );
    else if(text[i] == '\t') printf(" \\t ");
    else if(text[i] == '\r') printf(" \\r ");
    else if(text[i] == ' ') printf(" sp ");
    else if(isprint(text[i])) printf("%c", text[i]);
    else printf(" \\%d ", (unsigned char) text[i]);
    }

  printf("\"\n");
#endif
}

char *returnStr(char *yytext, int len){
  char *lval;

  if((lval = malloc(sizeof(char) *
    ((len == -1) ? strlen(yytext) : len) + 1)) == NULL)
    pandalex_error("Could not make space for lexer return.");
  memcpy(lval, yytext, ((len == -1) ? strlen(yytext) : len) + 1);

  return lval;
}
```

Code: lexer.l

pandalex.h

```
// This file defines the callbacks that users can setup to use PandaLex

#ifndef PANDALEX_H
#define PANDALEX_H

#include <stdarg.h>

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#ifdef __cplusplus
extern "C"
{
#endif
```

```
  enum{
  pandalex_event_begindocument = 0,
    pandalex_event_specver,
    pandalex_event_entireheader,
    pandalex_event_objstart,
    pandalex_event_objend,
    pandalex_event_dictitem_string,
    pandalex_event_dictitem_name,
    pandalex_event_dictitem_arraystart,
    pandalex_event_dictitem_arrayitem,
    pandalex_event_dictitem_arrayend,
    pandalex_event_dictitem_object,
    pandalex_event_dictitem_dict,
    pandalex_event_dictitem_dictend,
    pandalex_event_dictitem_int,
    pandalex_event_stream,
    pandalex_event_dictint,
    pandalex_event_xrefstart,
    pandalex_event_xrefitem,
    pandalex_event_xrefend,
    pandalex_event_trailerstart,
    pandalex_event_trailerend,
    pandalex_event_enddocument,
    pandalex_event_max
    };

  // Callbacks are defined so that they have a type for the arguments they
  // possess associated with them. Where possible the arguments created by the
  // lexer will be converted into the types needed by the callback. If not,
  // an error is returned
  typedef void (*pandalex_callback_type)(int, va_list);


  void pandalex_setupcallback(int, pandalex_callback_type);
  void pandalex_callback(int, ...);


  int pandalex_parse(char *filename);
  int pandalex_endparse();

  // Other stuff
  void pandalex_init();
  void debuglex(char *, int, char *);
  char *returnStr(char *, int);
  void *pandalex_xmalloc(size_t);
  void *pandalex_xrealloc(void *, size_t);
  char *pandalex_xsnprintf(char *, ...);
  void pandalex_error(char *);

  void pandalex_xfree(void *);
  char *pandalex_strmcat(char *, int, char *, int);
  char *pandalex_strmcpy(char *, int);
  int pandalex_intlen(int);

#ifdef __cplusplus
}
#endif

#endif
```

Code: pandalex.h

parser.y

```
%{
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>

#include <pandalex.h>

#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>

#define YYMAXDEPTH 50000
#define YYERROR_VERBOSE 1

  // The callbacks
  pandalex_callback_type pandalex_callbacks[pandalex_event_max];

  // Globals, these should go away
  int fd;
  char *file;
  struct stat sb;
  unsigned int gInset;

  int pandalex_gettext(char *buffer, int maxlen){
    int size;

    // Determine the maximum size to return
    size = pandalex_min(maxlen, sb.st_size - gInset);

    if(size > 0){
      memcpy(buffer, file + gInset, size);
      gInset += size;
    }

    return size;
  }

  int pandalex_min(int a, int b){
    if(a < b)
      return a;
    return b;
  }
%}

/* Define the possible yylval values */
%union {
  int        intVal;

  struct streamVal{
    char *data;
    int len;
  } sval;
}

%token <sval> PDFVER
%token <sval> NAME
%token <sval> STRING
%token <sval> OBJREF <sval> OBJ <sval> ENDOBJ
%token <intVal> INT
```

```
%token <sval> FP
%token <sval> DBLLT <sval> DBLGT
%token <sval> STREAM <sval> ENDSTREAM
%token <sval> ARRAY <sval> ENDARRAY <sval> ENDARRAYDBLGT
%token <sval> PDFEOF XREF TRAILER
%token <sval> ANYTHING

%type <sval> binary
%type <sval> header
%type <sval> objref
%type <sval> arrayvals

%type <intVal> dictionary
%type <intVal> subdictionary

%%

// completely implemented
pdf       : header { pandalex_callback(pandalex_event_entireheader, $1.data); }
            object linear objects xref trailer endcrap
          ;

// completely implemented
header    : PDFVER { pandalex_callback(pandalex_event_specver, $1.data); }
            binary { $$.data = pandalex_strmcat($1.data, $1.len, $3.data, $3.len); $$.l
          ;

linear    : xref trailer { }
          |
          ;

// Clibpdf sometimes puts some binary crap at the end of the file (pointer
// problems?)
// completely implemented
endcrap   : binary { }
          |
          ;

// completely implemented
objects   : object objects
          |
          ;

// todo_mikal: might need a .data here
object    : INT INT OBJ { pandalex_callback(pandalex_event_objstart, $1, $2); }
            dictionary { if($5 != -1) pandalex_callback(pandalex_event_dictint, $1, $2,
            stream ENDOBJ { pandalex_callback(pandalex_event_objend, $1, $2); }
          ;

dictionary: DBLLT dict DBLGT { $$ = -1; }
          | INT { $$ = $1; }
          | ARRAY arrayvals ENDARRAY { $$ = -1; }
          | objref { $$ = -1; }
          | NAME { $$ = -1; }
          | STRING { $$ = -1; }
          | { $$ = -1; }
          ;

subdictionary: DBLLT dict DBLGT { $$ = -1; }
          ;

dict      : NAME STRING { pandalex_callback(pandalex_event_dictitem_string, $1.data, $2
          | NAME NAME { pandalex_callback(pandalex_event_dictitem_name, $1.data, $2.dat
          | NAME ARRAY { pandalex_callback(pandalex_event_dictitem_arraystart, $1.data)
              arrayvals ENDARRAY { pandalex_callback(pandalex_event_dictitem_arrayend,
          | NAME objref { pandalex_callback(pandalex_event_dictitem_object, $1.data, $2
```

```
          | NAME { pandalex_callback(pandalex_event_dictitem_dict, $1.data); }
             subdictionary { pandalex_callback(pandalex_event_dictitem_dictend, $1.dat
          | NAME INT { pandalex_callback(pandalex_event_dictitem_int, $1.data, $2); } d
          | NAME FP {} dict
          |
          ;

arrayvals : objref { pandalex_callback(pandalex_event_dictitem_arrayitem, $1.data); } a
rayvals
          | INT { /*todo*/ } arrayvals
          | NAME { pandalex_callback(pandalex_event_dictitem_arrayitem, $1.data); } arr
          | STRING { pandalex_callback(pandalex_event_dictitem_arrayitem, $1.data); } a
          | ARRAY { pandalex_callback(pandalex_event_dictitem_arrayitem, $1.data); } ar
          | DBLLT { pandalex_callback(pandalex_event_dictitem_dict, "array-
dictitem"); }
             dict DBLGT { pandalex_callback(pandalex_event_dictitem_dictend, "array-
dictitem"); } arrayvals
          | {}
          ;

// completely implemented
objref    : INT INT OBJREF { if(($$.data = (char *) malloc((pandalex_intlen($1) + pan-
dalex_intlen($2) + 5) * sizeof(char))) == NULL){
                fprintf(stderr, "Could not allocate enough space for objref\n");
                exit(42);
                           }

              sprintf($$.data, "%d %d R", $1, $2);
              $$.len = strlen($$.data) + 1;
                              }
          ;

// completely implemented
stream    : STREAM binary ENDSTREAM { pandalex_callback(pandalex_event_stream, $2.data
3); free($2.data); }
          |
          ;

// completely implemented: callbacks are handled in the callers to this
binary    : ANYTHING binary { $$.data = pandalex_strmcat($1.data, $1.len, $2.data, $2.l
          | STRING binary { $$.data = pandalex_strmcat($1.data, -1, $2.data, $2.len); $
          | { $$.data = pandalex_strmcpy("", -1); $$.len = 0; }
          ;

// completely implemented
xref      : XREF INT INT { pandalex_callback(pandalex_event_xrefstart); }
             xrefitems {}
          ;

// completely implemented
xrefitems : INT INT STRING { pandalex_callback(pandalex_event_xrefitem, $1, $2, $3); }
             xrefitems
          | { pandalex_callback(pandalex_event_xrefend); }
          ;

// completely implemented
trailer   : TRAILER { pandalex_callback(pandalex_event_trailerstart); }
             DBLLT dict DBLGT STRING INT { pandalex_callback(pandalex_event_trailerend
             PDFEOF { pandalex_callback(pandalex_event_enddocument); }
          ;

%%

void pandalex_init(){
  int i;
```

```
  // Make sure that the callbacks default to nothing
  for(i = 0; i < pandalex_event_max; ++i){
    pandalex_callbacks[i] = NULL;
  }
}

void pandalex_setupcallback(int callback, pandalex_callback_type functoid){
#if defined DEBUG
  printf("Defining a callback\n");
#endif

  pandalex_callbacks[callback] = functoid;
}

// Here we call the callbacks. This includes converting to the types that the
// callback function expects.
void pandalex_callback(int event, ...){
  va_list argptr;

#if defined DEBUG
  printf("Attempting to call callback\n");
#endif

  // Start accessing the arguements from the end
  va_start(argptr, event);

  // If no event handler is setup, then we ignore the event
  if(pandalex_callbacks[event] != NULL){
    pandalex_callbacks[event] (event, argptr);
  }
#if defined DEBUG
  else printf("No callback defined\n");
#endif

  // Stop with the arguements
  va_end(argptr);
}

int pandalex_parse(char *filename){
  // Map the input file into memory
  if ((fd = open (filename, O_RDONLY)) < 0)
    {
      perror("Could not open file");
      return(-1);
    }

  if(fstat(fd, &sb) < 0){
    perror("Could not stat file");
    return(-1);
  }

  if ((file =
       (char *) mmap (NULL, sb.st_size, PROT_READ, MAP_SHARED, fd, 0)) == -
1)
    {
      perror("Could not mmap file");
      return(-1);
    }

  // We are not looking into a stream at the moment
  pandalex_callback(pandalex_event_begindocument, filename);
  yyparse();
}

int pandalex_endparse(){
  if(munmap(file, sb.st_size) < 0){
```

```
    perror("Could not unmap memory");
    return(-1);
  }

  close(fd);
}

int yyerror(char *s){
  int i;

  fprintf(stderr, "\n-----------------------------------------------
--------------\n");
  fprintf(stderr, "PandaLex parser error (%s):\n", s);
  fprintf(stderr, "  Please send this error text, along with a copy of your PDF\n");
  fprintf(stderr, "  document (if possible) to mikal@stillhq.com, so that this can\n");
  fprintf(stderr, "  be fixed for the next release...\n\n");
  fprintf(stderr, "version = %s\n", VERSION);

  //  fprintf(stderr, "last token = \"%s\" (%d) or %d\n", yylval.sval.data, yyl-
val.sval.len, yylval.intVal);
  fprintf(stderr, "Last token matched was:\n  ");
  for(i = 0; i < ((yylval.sval.len == -1) ? strlen(yylval.sval.data) : yyl-
val.sval.len); i++){
    if(yylval.sval.data[i] == '\n') fprintf(stderr, " \\n " );
    else if(yylval.sval.data[i] == '\t') fprintf(stderr, " \\t ");
    else if(yylval.sval.data[i] == '\r') fprintf(stderr, " \\r ");
    else if(yylval.sval.data[i] == ' ') fprintf(stderr, " sp ");
    else if(isprint(yylval.sval.data[i])) fprintf(stderr, "%c", yylval.sval.data[i]);
    else fprintf(stderr, " \\%d ", (unsigned char) yylval.sval.data[i]);
    }
  fprintf(stderr, "\"\n");

  fprintf(stderr, "\n-----------------------------------------------
--------------\n");

  exit(42);
}

// Buffer overrun safe strcat
char *pandalex_strmcat(char *dest, int destLen, char *append, int appendLen){
  char *new;
  int count, len;

  // What length do we need?
  if((new = (char *) malloc(sizeof(char) *
               (((destLen == -1) ? strlen(dest) : destLen) +
               ((appendLen == -1) ? strlen(append) : appendLen) +
               2))) == NULL){
    fprintf(stderr, "Could not malloc enough space\n");
    exit(42);
  }

  if((destLen == -1) && (appendLen == -1))
    sprintf(new, "%s%s", dest, append);
  else{
    // We need to copy characters the hard way -- change this to a memcpy
    count = 0;

    for(len = 0; len < ((destLen == -1) ? strlen(dest) : destLen); len++){
      new[count] = dest[len];
      count++;
    }

    for(len = 0; len < ((appendLen == -1) ? strlen(append) : appendLen); len++){
      new[count] = append[len];
      count++;
```

```
    }

    new[count] = '\0';
  }
  return new;
}

// Buffer overrun safe strcpy
char *pandalex_strmcpy(char *data, int len){
  return pandalex_strmcat(data, len, "", 0);
}

int pandalex_intlen(int number){
  int length = 0;

  while(number > 0){
    length ++;
    number /= 10;
  }

  return number;
}

void pandalex_xfree(void *ptr){
  if(ptr != NULL)
    free(ptr);
}
```

Code: parser.y

samples.c

```
/* A sample application using pandalex -- this is pdfdump */

#include <stdarg.h>
#include <stdio.h>
#include <zlib.h>
#include "samples.h"
#include "pandalex.h"

enum{
  pdfdump_dump = 0,
    pdfdump_meta
    };
int pdfdump_application;

pdfdump_dictint_list *dictint_list;

char *filter = NULL;

// Some demo code for how to use PandaLex
int main(int argc, char *argv[]){
  pandalex_init();

  // Parse the command line to find out what we are doing today -- this needs more thou
  if(strcmp(argv[0], "pdfmeta") == 0){
    pdfdump_application = pdfdump_meta;
  }
  else
    pdfdump_application = pdfdump_dump;

  // Setup the callbacks
  pandalex_setupcallback(pandalex_event_begindocument, pdfdump_begindocument);
```

```
  pandalex_setupcallback(pandalex_event_specver, pdfdump_specversion);
  pandalex_setupcallback(pandalex_event_entireheader, pdfdump_entireheader);
  pandalex_setupcallback(pandalex_event_objstart, pdfdump_objstart);
  pandalex_setupcallback(pandalex_event_objend, pdfdump_objend);

  pandalex_setupcallback(pandalex_event_dictitem_string, pdfdump_dictitem_string);
  pandalex_setupcallback(pandalex_event_dictitem_name, pdfdump_dictitem_name);
  pandalex_setupcallback(pandalex_event_dictitem_arraystart, pdfdump_dictitem_arraystar
  pandalex_setupcallback(pandalex_event_dictitem_arrayitem, pdfdump_dictitem_arrayitem)
  pandalex_setupcallback(pandalex_event_dictitem_arrayend, pdfdump_dictitem_arrayend);
  pandalex_setupcallback(pandalex_event_dictitem_object, pdfdump_dictitem_object);
  pandalex_setupcallback(pandalex_event_dictitem_dict, pdfdump_dictitem_dict);
  pandalex_setupcallback(pandalex_event_dictitem_dictend, pdfdump_dictitem_dictend);
  pandalex_setupcallback(pandalex_event_dictitem_int, pdfdump_dictitem_int);

  pandalex_setupcallback(pandalex_event_stream, pdfdump_stream);
  pandalex_setupcallback(pandalex_event_dictint, pdfdump_dictint);

  // Initialise the dictint_list structure;
  if((dictint_list = (pdfdump_dictint_list *)
      malloc(sizeof(pdfdump_dictint_list))) == NULL){
    fprintf(stderr, "Could not initialise the dictint list\n");
    exit(42);
  }

  dictint_list->next = NULL;

  // Start parsing
  pandalex_parse(argv[1]);

  return 0;
}

char *pandalex_xsnprintf(char *, ...);

// Arguement is the name of the file as a char *
void pdfdump_begindocument(int event, va_list argptr){
  char *filename;

  filename = va_arg(argptr, char *);
  printf("Information for document: \"%s\"\n\n", filename);
}

void pdfdump_specversion(int event, va_list argptr){
  printf("Specification version is: %s\n", (char *) va_arg(argptr, char *));
}

void pdfdump_entireheader(int event, va_list argptr){
  int    i;
  char   *textMatch = (char *) va_arg(argptr, char *);

  printf("Entire document header is: ");

  for(i = 0; i < strlen(textMatch); i++){
    if(isprint(textMatch[i])) printf("%c ", textMatch[i]);
    else printf("\\%d ", textMatch[i]);
  }

  printf("\n");
}

void pdfdump_objstart(int event, va_list argptr){
  int generation, number;

  number = va_arg(argptr, int);
  generation = va_arg(argptr, int);
```

```
  printf("Object %d started (generation %d)\n",
     number, generation);

  // The default filter is none
  pandalex_xfree(filter);
  filter = NULL;
}

void pdfdump_objend(int event, va_list argptr){
  int generation, number;

  number = va_arg(argptr, int);
  generation = va_arg(argptr, int);

  printf("Object %d ended (generation %d)\n",
     number, generation);
}

void pdfdump_dictitem_string(int event, va_list argptr){
  char *name, *value;

  name = va_arg(argptr, char *);
  value = va_arg(argptr, char *);
  printf("  [String] %s = \"%s\"\n", name, value);

  if(strcmp(name, "Filter") == 0){
    filter = value;
  }
}

void pdfdump_dictitem_name(int event, va_list argptr){
  char *name, *value;

  name = va_arg(argptr, char *);
  value = va_arg(argptr, char *);
  printf("  [Name] %s = %s\n", name, value);
}

void pdfdump_dictitem_arraystart(int event, va_list argptr){
  char *name;

  name = va_arg(argptr, char *);
  printf("  Array %s starts\n", name);
}

void pdfdump_dictitem_arrayitem(int event, va_list argptr){
  char *value;

  value = va_arg(argptr, char *);
  printf("  [Array] %s\n", value);
}

void pdfdump_dictitem_arrayend(int event, va_list argptr){
  char *name;

  name = va_arg(argptr, char *);
  printf("  Array %s ends\n", name);
}

void pdfdump_dictitem_object(int event, va_list argptr){
  char *name, *value;

  name = va_arg(argptr, char *);
  value = va_arg(argptr, char *);
  printf("  [Object reference] %s = %s\n", name, value);
```

```
}

void pdfdump_dictitem_dict(int event, va_list argptr){
  char *name;

  name = va_arg(argptr, char *);
  printf("Subdictionary \"%s\" starts\n", name);
}

void pdfdump_dictitem_dictend(int event, va_list argptr){
  char *name;

  name = va_arg(argptr, char *);
  printf("Subdictionary \"%s\" ends\n", name);
}

void pdfdump_dictitem_int(int event, va_list argptr){
  int value;
  char *name;

  name = va_arg(argptr, char *);
  value = va_arg(argptr, int);

  printf("  [Integer] %s = %d\n", name, value);
}

void pdfdump_stream(int event, va_list argptr){
  char *streamData;
  int streamDataLen;
  pdfdump_dictint_list  *now;
  int found;

  printf("  [Stream, filter %s]\n", filter);
  streamData = va_arg(argptr, char *);
  streamDataLen = va_arg(argptr, int);

  printf("  Length = %d\n", streamDataLen);
  printf("\n\n--------------------------------\n%s\n----------------
--------------\n\n",
     streamData);
}

void pdfdump_dictint(int event, va_list argptr){
  int found;
  int objnum, objgen, value;
  char *objref;
  pdfdump_dictint_list *now;

  // Get the passed information
  objnum = va_arg(argptr, int);
  objgen = va_arg(argptr, int);
  value = va_arg(argptr, int);

  printf("Do something with the dictint %d %d R = %d\n", objnum, obj-
gen, value);

  // Information is handed to this event in a slightly different man-
ner to
  // the stream event handler. Fix this.
  if((objref = (char *) malloc((pandalex_intlen(objnum) + pandalex_intlen(objgen) + 5)
    fprintf(stderr, "Could not allocate enough space for objref\n");
    exit(42);
  }

  sprintf(objref, "%d %d R", objnum, objgen);
```

```
  // Are we already waiting?
  now = dictint_list;
  found = 0;

  while((now->next != NULL) && (found == 0)){
    if(strcmp(objref, now->value) == 0){
      // Yes -- do something
      pdfdump_procstream(now->filter, value, now->stream, now->streamlen);
      found = 1;
    }

    now = now->next;
  }

  // No -- save data and wait
  if(found == 0){
    // now is already the end of the list
    if((now->next = (pdfdump_dictint_list *)
    malloc(sizeof(pdfdump_dictint_list))) == NULL){
      fprintf(stderr, "Could not add to list of waiting streams\n");
      exit(42);
    }

    now->value = (char *) pandalex_strmcpy(objref, -1);
    now->filter = NULL;
    now->stream = NULL;
    now->waiting = 2;
    now = now->next;
    now->next = NULL;
  }
}

void pdfdump_procstream(char *filter, int length, char *data, int dataLen){
  char *uncompressed, *dataPtr, *linhintdesc[17];
  uLong srcLen, dstLen = 512;
  int result, i, linhintlens[17], number, count;

  linhintlens[0] = 32;
  linhintdesc[0] = pandalex_xsnprintf("Least number of objects in a page");
  linhintlens[1] = 32;
  linhintdesc[1] = pandalex_xsnprintf("Location of the first page object");
  linhintlens[2] = 16;
  linhintdesc[2] = pandalex_xsnprintf("Page objects delta bits");
  linhintlens[3] = 32;
  linhintdesc[3] = pandalex_xsnprintf("Least page length");
  linhintlens[4] = 16;
  linhintdesc[4] = pandalex_xsnprintf("Page length delta bits");
  linhintlens[5] = 32;
  linhintdesc[5] = pandalex_xsnprintf("Least content stream offset");
  linhintlens[6] = 16;
  linhintdesc[6] = pandalex_xsnprintf("Content stream offset delta bits");
  linhintlens[7] = 32;
  linhintdesc[7] = pandalex_xsnprintf("Least content stream length");
  linhintlens[8] = 16;
  linhintdesc[8] = pandalex_xsnprintf("Contents stream length delta bits");
  linhintlens[9] = 16;
  linhintdesc[9] = pandalex_xsnprintf("Greatest shared object number bits");
  linhintlens[10] = 16;
  linhintdesc[10] = pandalex_xsnprintf("Numerically greatest shared ob-
ject number bits");
  linhintlens[11] = 16;
  linhintdesc[11] = pandalex_xsnprintf("Numeration object fraction bits");
  linhintlens[12] = 16;
  linhintdesc[12] = pandalex_xsnprintf("Denominator object fraction bits");
  linhintlens[13] = 16;
  linhintdesc[13] = pandalex_xsnprintf("?????");
```

```
// Check length
if(length < 1){
  fprintf(stderr, "Stream length is not believable\n");
  return;
}

// Check there is a filter at all
if(filter == NULL){
  fprintf(stderr, "This stream is not compressed!\n");
  return;
}

// If the stream starts with a \r or a \n or a \r\n, then these should be stripped of
dataPtr = data;
while((dataPtr[0] == '\r') || (dataPtr[0] == '\n')) dataPtr++;

// Do something with the stream
if(strcmp(filter, "FlateDecode") == 0){
  printf("Do something involving Flate\n");

  //    printf("--------------------------------------------------");
  //for(i = 0; i < dataLen; i++)
  //  printf("%c", data[i]);
  //printf("--------------------------------------------------");


  // - 1
  for(i = -10; i < 1; i++){
    srcLen = dataLen + i;
    dstLen = 512;
    printf("[%d] ", i);

    if((uncompressed = (char *) malloc(sizeof(char) * dstLen)) == NULL){
  fprintf(stderr, "Could not make enough space to decompress Flate stream\n");
  exit(42);
    }

    // We grow the output buffer until we no longer get buffer size errors
    while((result = uncompress(uncompressed, &dstLen, dataPtr, srcLen)) == Z_BUF_ERRO
  printf(".");
  fflush(stdout);

  dstLen *= 2;
  if(((uncompressed = (char *) realloc(uncompressed, dstLen)) == NULL) ||
     (dstLen > 10000000)){
    // We could not grow the buffer, so we exit
    printf("!");
    fflush(stdout);
    free(uncompressed);
    break;
  }
    }

    if(result == Z_OK) printf(" HIT");
    printf(" *\n");
  }

  if(result != Z_OK){
    fprintf(stderr, "Flate decompression failed because of ");

    switch(result){
    case Z_MEM_ERROR:
  fprintf(stderr, "not enough memory\n");
  break;
```

```
      case Z_DATA_ERROR:
    fprintf(stderr, "corrupt input data\n");
    break;

      case Z_BUF_ERROR:
    fprintf(stderr, "buffer error\n");
    break;

      default:
    fprintf(stderr, "unknown error (%d)\n", result);
    break;
      }

      debuglex(data, srcLen, "Flate compression failure");
      exit(46);
    }

    printf("\n");
    printf("----------- UNCOMPRESSED STREAM IS ----------------------
--------------------\n");
    // printf("%s\n", uncompressed);
    printf("Total uncompressed size: %d\n\n", dstLen);

    count = 0;

    for(i = 0; count < 13;){
      number = 0;
      if(linhintlens[count] == 16){
    number = uncompressed[i] << 8 | uncompressed[i + 1];
    i += 2;
      }
      else{
    number = uncompressed[i] << 24 | uncompressed[i + 1] << 16 |
      uncompressed[i + 2] << 8 | uncompressed[i + 3];
    i += 4;
      }

      printf("%s [%d]: %d\n", linhintdesc[count], linhintlens[count], number);
      count++;
    }

    printf("\nTotal bytes used: %d\n", i);
    printf("\n----------------------------------------------------------
----------------------\n");
  }
  else if(strcmp(filter, "LZWDecode") == 0){
    printf("LZW compression is encumbered by Patents and therefore not sup-
ported\n");
  }
  else if(strcmp(filter, "CCITTFaxDecode") == 0){
    printf("Do something involving CCITTFax compression (TIFF)\n");
  }
  else{
    printf("Unknown filter \"%s\"\n", filter);
  }
}
```

Code: samples.c

samples.h

```
void pdfdump_begindocument(int, va_list);
void pdfdump_specversion(int, va_list);
void pdfdump_entireheader(int, va_list);
void pdfdump_objstart(int, va_list);
void pdfdump_objend(int, va_list);

void pdfdump_dictitem_string(int, va_list);
void pdfdump_dictitem_name(int, va_list);
void pdfdump_dictitem_arraystart(int, va_list);
void pdfdump_dictitem_arrayitem(int, va_list);
void pdfdump_dictitem_arrayend(int, va_list);
void pdfdump_dictitem_object(int, va_list);
void pdfdump_dictitem_dict(int, va_list);
void pdfdump_dictitem_dictend(int, va_list);
void pdfdump_dictitem_int(int, va_list);

void pdfdump_stream(int, va_list);
void pdfdump_dictint(int, va_list);
void pdfdump_procstream(char *, int, char *, int);

// This data type is needed for pdfdump_stream and
// pdfdump_dictint
typedef struct pdfdump_internal_dictint_list{
  char *value;
  int waiting;
  int number;

  char *stream;
  int streamlen;
  char *filter;

  struct pdfdump_internal_dictint_list  *next;
} pdfdump_dictint_list;
```

Code: samples.h

Conclusion

In this chapter we have discussed the inner workings of the PDF format, which is quite different to the other formats we have discussed in this document. We have also examined Panda, and seen how to use it to generate our PDF documents.

Further reading

The following resources might be of use if you need more information:

- http://developer.adobe.com has many useful resources, including the PDF specification (at the time of writing the latest version is 1.4, although this chapter is based on 1.3, because this is what is supported by the two libraries discussed here).
- http://www.stillhq.com: has the comp.text.pdf frequently asked questions, as well as the Panda pages.

Notes

1. Whilst the vast majority of PDF structure is represented with ASCII text, this doesn't stop you from embedding binary into the file
2. Non integer
3. PDF Specification 1.3, second edition, page 27
4. That is, in the range 0 to 255.
5. ASCII85 encoding is when you take binary data (which will be in the range 0x00 to 0xFF per byte, and force it into the range 0x00 to 0x55. This is done because some transport mediums (such as email), cannot handle binary data, so the data is made to look more like ASCII information. Note that this bloats the size of the data somewhat.
6. More have been added to this list in PDF 1.4