GIMP-Print

The print plugin for the GIMP
Version 4.2.0, 18 November 2001

The GIMP-Print Project

Short Contents

Table of Contents

Preface

This manual documents the use of the GIMP-Print package, focusing mainly on the libgimpprint library that is the core of GIMP-Print. Parts of the manual which describe the use of libgimpprint are aimed primarily at programmers, and do assume that the reader is familiar with C programming, and using standard programming tools on GNU or UNIX systems. Other parts of the manual document the use of the programs that make up the GIMP-Print package, and assume no knowledge other than the basics of using a shell and editor.

The best way to learn how to use libgimpprint in your own programs is to read the manual from start to finish, using the examples given in the text to learn how it works. The bulk of the manual can be used as a reference once one understands the basics.

The manual is split into several parts for the programmer. It starts with a simple usage example of how to link a program with libgimpprint, then how to integrate this into package build scripts, using `make`, `autoconf` and `automake`. This is followed by a detailed function reference, including descriptions of all the data types used.

For the end-user, there is a section on all of the programs that come with GIMP-Print, including the GIMP `print` plugin, and the CUPS and Ghostscript drivers.

The appendices at the end of the manual detail the dither and weave algorithms used in libgimpprint, the ESC/P2 printer control language used in Epson printers and how to add support for a new printer to libgimpprint.

We hope you enjoy using GIMP-Print!
—The GIMP-Print project

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

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.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. 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.
2. 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.
3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

11. 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

12. 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.
13. 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 an 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., 675 Mass Ave, Cambridge, MA 02139, 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.

Copying summary

GIMP-Print is *free*; this means that everyone is free to use it and free to redistribute it on a free basis. GIMP-Print is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of GIMP-Print that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of GIMP-Print, that you receive source code or else can get it if you want it, that you can change GIMP-Print or use pieces of it in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of GIMP-Print, 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 tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for GIMP-Print. If GIMP-Print is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will no reflect on our reputation.

1 Overview

The GIMP-Print package is a collection of programs and a library for high quality printing on modern inkjets, including 'photographic quality' models which offer very high resolutions and several inks.

META: Short history of the project

GIMP-Print is the print facility of the *GNU Image Manipulation Program* (GIMP). It is in addition a suite of drivers that may be used with common UNIX spooling systems using GhostScript or CUPS. These drivers provide printing quality for UNIX/Linux on a par with proprietary vendor-supplied drivers in many cases, and can be used for many of the most demanding printing tasks. The core of GIMP-Print is a shared library (libgimpprint) which may be used by any program that wishes to produce high-qulaity printed output.

2 Using libgimpprint

This chapter describes how to write programs that use libgimpprint.

2.1 Code prerequisites

To use libgimpprint with a program, several steps must be taken:

1. Include the master libgimpprint header `<gimp-print/gimp-print.h>`.
2. Call `stp_init()`.
3. Link with libgimpprint.

The following is a short example program. It does not do anything useful, but it does everything required to link with libgimpprint and call other functions from libgimpprint.

```
#include <gimp-print/gimp-print.h>

int
main (int argc, char *argv[])
{
  stp_init ();
  return (0);
}
```

2.2 Linking with libgimpprint

To link a program with libgimpprint, then '`-lgimpprint`' needs to be passed to the compiler when linking. For example, to compile and link '`prog.c`' the following commands would be used:

```
gcc -c prog.c
gcc -o prog -lgimpprint prog.o
```

The compiler and linker flags needed may vary depending on the options GIMP-Print was configured with when it was built. The `gimpprint-config` script will give the correct parameters for the local installation (see Section 3.1 [gimpprint-config], page 12).

3 Integrating libgimpprint

This chapter describes how to integrate the compiling and linking of programs using libgimpprint with build scripts. Commonly used systems include `make`, but more commonly '`Makefile`' files are generated by using tools such as `autoconf` and `automake`.

3.1 gimpprint-config

Depending on the setup of the computer system GIMP-Print was installed on, as well as the options passed to `configure` when configuring the package when it was built, the `CFLAGS` and `LIBS` parameters needed to compile and link programs with libgimpprint may vary. To make it simple to determine what these are on any given system, the script `gimpprint-config` was created. It's job is to output the correct parameters for the setup on your system. The following options are available:

```
roger@whinlatter:~/gimpprint/devel$ gimpprint-config --help
Usage: gimpprint-config [OPTIONS] [LIBRARIES]
Options:
        [--prefix[=DIR]]
        [--exec-prefix[=DIR]]
        [--version]
        [--libs]
        [--cflags]
Libraries:
        gimpprint
```

The '`--prefix`' and '`--exec-prefix`' options are only needed if the installed locations of parts of GIMP-Print are different from the configured locations. These should never be needed if GIMP-Print was properly configured and installed.

The installed version of GIMP-Print can be obtained with the '`--version`' option:

```
roger@whinlatter:~/gimpprint/devel$ gimpprint-config --version
4.2.0
```

The correct `CFLAGS` to use can be obtained with the '`--cflags`' option:

```
roger@whinlatter:~/gimpprint/devel$ gimpprint-config --cflags

```

In this case, there are no special `CFLAGS` required to compile programs.

The correct `LIBS` to use can the obtained with the '`--libs`' option:

```
roger@whinlatter:~/gimpprint/devel$ gimpprint-config --libs
-L/usr/lib -lgimpprint -lm
```

The command can be used from the shell by enclosing it in backquotes '``':

```
gcc `gimpprint-config --cflags` -c prog.c
gcc `gimpprint-config --libs` -o prog prog.o
```

However, this is not the way it it typically used. Normally it is used in a '`Makefile`' (see Section 3.2 [make], page 13) or by an `m4` macro in a `configure` script (see Section 3.3 [autoconf], page 13).

3.2 make

If you use make with your own 'Makefile' files, then you are on your own. This manual offers no assistance with doing this. Only the following suggestion is offered:

```
GIMPPRINT_VERSION = $(shell gimpprint-config --version)
GIMPPRINT_CFLAGS = $(shell gimpprint-config --cflags)
GIMPPRINT_LIBS = $(shell gimpprint-config --libs)
```

How you choose to use these variables is entirely up to you. See Info file 'make', node 'Top', for more information.

3.3 autoconf

The autoconf program produces a Bourne shell script called 'configure' from a template file called 'configure.in'. 'configure.in' contains both Bourne shell script, and m4 macros. autoconf expands the m4 macros into 'real' shell script. The resulting 'configure' script performs various checks for installed programs, compiler characteristics and other system information such as available headers and libraries. See Info file 'autoconf', node 'Top', for more information.

GIMP-Print provides an m4 macro, AM_PATH_GIMPPRINT, suitable for use in a 'configure.in'. It defines the environment variables GIMPPRINT_CFLAGS, GIMPPRINT_LIBS and GIMPPRINT_CONFIG. You can optionally specify a minimum version of the library to use, and shell script to run if the test suceeds or fails.

AM_PATH_GIMPPRINT ([*minimum-version* [, *action-if-found* [, *action-if-not-found*]]]) Macro

Check for an installed version of GIMP-Print greater than or equal to *minimum-version*.

action-if-found is a list of shell commands to run if the check for the library succeeds; *action-if-not-found* is a list of shell commands to run if the check fails.

The macro sets the following environment variables: GIMPPRINT_CFLAGS, GIMPPRINT_LIBS and GIMPPRINT_CONFIG. It also will substitute them into any 'Makefile.in' you specify in AC_OUTPUT because it calls AC_SUBST for each of them. However, you will probably be using automake to generate your 'Makefile.in' files (see Section 3.4 [automake], page 13).

3.4 automake

The automake program can be used to generate 'Makefile.in' files suitable for use with a 'configure' script generated by autoconf. As automake *requires* autoconf, this section will assume the use of a 'configure' script which uses the AM_PATH_GIMPPRINT macro (there is little point in *not* using it!).

It is highly recommeded that you use GNU autoconf and automake. They will allow you to make your software build on most platforms with most compilers. automake makes writing complex 'Makefile' files very easy, by expressing how to build your packages in terms of what files are required to build a project and the installation locations of the files. It imposes a few limitations over using plain 'Makefile' files, such as in the use of conditionals, but these problems are vastly outweighed by the benefits it brings. It also creates many extra targets in the generated 'Makefile.in' files such as dist, distcheck, clean, distclean, maintainer-clean and tags, and there are many more more available. See Info file 'automake', node 'Top', for more information.

Because AM_PATH_GIMPPRINT calls AC_SUBST to substitute GIMPPRINT_CFLAGS, GIMPPRINT_LIBS and GIMPPRINT_CONFIG, automake will automatically set these variables in the 'Makefile.in' files it generates, requiring no additional effort on your part!

As in previous examples, we will make a program prog from a file 'prog.c'. This is how one might build write a 'Makefile.am' to do this:

```
AUTOMAKE_OPTIONS = 1.4 gnu
MAINT_CHARSET = latin1

@SET_MAKE@

CFLAGS = @CFLAGS@

INCLUDES = @INCLUDES@ $(GIMPPRINT_CFLAGS)

bin_PROGRAMS = prog
prog_SOURCES = prog.c
prog_LDADD = $(GIMPPRINT_LIBS)

MAINTAINERCLEANFILES = Makefile.in
```

That's all there is to it! Please note that this example also requires the macro AC_PROG_MAKE_SET to be used in 'configure.in' and the use of AC_SUBST to substitute CFLAGS and INCLUDES where @CFLAGS@ and @INCLUDES@ are found in the file, respectively.

4 Functions

This section of the manual is a complete reference to all of the functions comprising libgimpprint, with examples.

All of the libgimpprint functions can be declared by including the master libgimpprint header, '`<gimp-print/gimp-print.h>`'. This should be done in every source file that uses gimpprint functions by putting

```
#include <gimp-print/gimp-print.h>
```

at the top of each file.

4.1 Main functions

These functions are essential to the correct functioning of libgimpprint. All or most other functions in the library depend absolutely upon them. In the case of `stp_init`, this function *must* be called before any of the other functions in the library.

`int` **stp_init** (`void`) Function

This function initialises the libgimpprint library. It must be called before any of the other libgimpprint functions are called. It is responsible for setting up message catalogues (for internationalisation). This function may be called more than once, at any stage during the execution of a program.

It returns zero on success, nonzero on failure.

`stp_init` might be used as follows:

```
int
main (int argc, char **argv)
{
  stp_init();

  ...

}
```

`void *` **stp_malloc** (`size_t` *size*) Function

Where *size* is the amount of memory to allocate (in bytes).

This function allocates memory. It will always return a pointer to the allocated memory. It will not return on failure.

It returns a pointer to the allocated memory.

4.2 `stp_vars_t` functions

`void *` **stp_vars_t** Data type

This is an opaque data type, whose structure is not visible to the user. This object contains all of the information about settings for a given printer, such as color (contrast, brightness), the type of printer, the dithering algorithm in use, and so forth. Please see the `stp_set_*` and `stp_get_*` functions below for the accessors and mutators for this data type.

stp_vars_t **stp_allocate_vars** (void) Function

Allocate a new stp_vars_t with default settings for all members.

void **stp_copy_vars** (stp_vars_t *vd*, const stp_vars_t *vs*) Function

Copy the settings from *vs* to *vd*.

stp_vars_t **stp_allocate_copy** (const stp_vars_t *vs*) Function

Allocate a new stp_vars_t, copying settings from *vs*.

void **stp_free_vars** (stp_vars_t *vv*) Function

Free all resources associated with *vv*. *vv* must not be used in any way following this call.

const char * **stp_get_output_to** (const stp_vars_t *vv*) Function
void **stp_set_output_to** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_output_to_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the name of the command that this job will be printed to. This is used by front ends; the driver library always prints to a stream provided by the front end and never uses this directly.

const char * **stp_get_driver** (const stp_vars_t *vv*) Function
void **stp_set_driver** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_driver_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the name of the driver (the type of printer).

const char * **stp_get_ppd_file** (const stp_vars_t *vv*) Function
void **stp_set_ppd_file** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_ppd_file_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the name of the PPD file used by this print job. Normally, only PostScript printers use PPD files.

const char * **stp_get_resolution** (const stp_vars_t *vv*) Function
void **stp_set_resolution** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_resolution_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the resolution to be used in this print job. Different drivers support different resolutions, and many drivers support multiple quality settings for a given DPI resolution.

const char * **stp_get_media_size** (const stp_vars_t *vv*) Function
void **stp_set_media_size** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_media_size_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the name of the media size (e. g. A3, letter, legal) to be used in this print job.

const char * **stp_get_media_type** (const stp_vars_t *vv*) Function
void **stp_set_media_type** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_media_type_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the name of the media type (e. g. plain paper, photo quality inkjet paper) to be used in this print job.

const char * **stp_get_media_source** (const stp_vars_t *vv*) Function
void **stp_set_media_source** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_media_source_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the name of the media source (e. g. manual feed, tray A) to be used in this print job.

const char * **stp_get_ink_type** (const stp_vars_t *vv*) Function
void **stp_set_ink_type** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_ink_type_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the name of the ink type (e. g. four color standard, six color photo) to be used in this print job.

const char * **stp_get_dither_algorithm** (const stp_vars_t *vv*) Function
void **stp_set_dither_algorithm** (stp_vars_t *vv*, const char **val*) Function
void **stp_set_dither_algorithm_n** (stp_vars_t *vv*, const char **val*, int *bytes*) Function

Get or set the dither algorithm to be used in this print job.

int **stp_get_output_type** (const stp_vars_t *vv*) Function
void **stp_set_output_type** (stp_vars_t *vv*, int *val*) Function

Get or set the output type (color, grayscale, black and white) for this print job.

int **stp_get_orientation** (const stp_vars_t *vv*) Function
void **stp_set_orientation** (stp_vars_t *vv*, int *val*) Function

Get or set the paper orientation for this print job.

int **stp_get_left** (const stp_vars_t *vv*) Function
void **stp_set_left** (stp_vars_t *vv*, int *val*) Function

Get or set the left margin (in 1/72 inch units, or "points") for this print job.

int **stp_get_top** (const stp_vars_t *vv*) Function
void **stp_set_top** (stp_vars_t *vv*, int *val*) Function

Get or set the top margin (in 1/72 inch units, or "points") for this print job.

int **stp_get_image_type** (const stp_vars_t *vv*) Function
void **stp_set_image_type** (stp_vars_t *vv*, int *val*) Function

Get or set the image type (line art, continuous tone, solid colors) for this print job.

`int` **stp_get_unit** (`const stp_vars_t` *vv*) Function
`void` **stp_set_unit** (`stp_vars_t` *vv*, `int` *val*) Function

Get or set the base unit (inches or centimeters) for this print job. This is provided for front ends; the package itself uses points as its unit of measurement.

`int` **stp_get_page_width** (`const stp_vars_t` *vv*) Function
`void` **stp_set_page_width** (`stp_vars_t` *vv*, `int` *val*) Function

Get or set the width of the printed region of the page.

`int` **stp_get_page_height** (`const stp_vars_t` *vv*) Function
`void` **stp_set_page_height** (`stp_vars_t` *vv*, `int` *val*) Function

Get or set the height of the printed region of the page.

`int` **stp_get_input_color_model** (`const stp_vars_t` *vv*) Function
`void` **stp_set_input_color_model** (`stp_vars_t` *vv*, `int` *val*) Function

Get or set the color model (currently RGB or CMY) of the input to the driver. Most front ends will use RGB input.

`int` **stp_get_output_color_model** (`const stp_vars_t` *vv*) Function
`void` **stp_set_output_color_model** (`stp_vars_t` *vv*, `int` *val*) Function

Get or set the color model (currently RGB or CMY) of the output of the driver. Most printers will use CMY.

`float` **stp_get_scaling** (`const stp_vars_t` *vv*) Function
`void` **stp_set_scaling** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the scaling factor of the image. If the scaling factor is greater than 0, it is interpreted as a percent (5.0-100.0 is the valid range) of the printable page region, using the more restrictive axis. For example, if the image to be printed should be 3" (wide) x 2" (high), and the printable page region is 8"x10.5", the scale factor should be 37.5 (3"/8").

If the scaling is less than zero, it is interpreted as pixels per inch.

It is likely that in the future this will be migrated into the front end. There is no particular reason why the driver needs to know about this.

`float` **stp_get_gamma** (`const stp_vars_t` *vv*) Function
`void` **stp_set_gamma** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the gamma of the print job (valid range: 0.1-4.0; default 1.0). Note that this is not the absolute gamma used by the print job; it is scaled by the gamma appropriate for the printer. This is true for all of the numerical parameters.

`float` **stp_get_brightness** (`const stp_vars_t` *vv*) Function
`void` **stp_set_brightness** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the brightness of the print job (valid range: 0.0-2.0; default: 1.0). Any value other than 1.0 will result in some possible values not being used; if brightness is less than 1.0, no output point will be pure white, and if brightness is greater than 1.0, no output point will be pure black (or cyan, or magenta, or yellow).

`float` **stp_get_contrast** (`const stp_vars_t` *vv*) Function
`void` **stp_set_contrast** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the contrast of the print job (valid range: 0.0-4.0; default: 1.0). Values less than 1.0 will result in pure white or black not being used (0.0 will make the entire image 50% gray). Values greater than 1.0 do not hard clip; while the contrast in the midtones increases, it only asymptotically approaches the limits.

`float` **stp_get_cyan** (`const stp_vars_t` *vv*) Function
`void` **stp_set_cyan** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the cyan adjustment of the print job (range: 0.0-4.0; default: 1.0). This currently adjusts the gamma of the cyan curve. It is scaled by the cyan adjustment for the printer model in question.

`float` **stp_get_magenta** (`const stp_vars_t` *vv*) Function
`void` **stp_set_magenta** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the magenta adjustment of the print job (range: 0.0-4.0; default: 1.0). This currently adjusts the gamma of the magenta curve. It is scaled by the magenta adjustment for the printer model in question.

`float` **stp_get_yellow** (`const stp_vars_t` *vv*) Function
`void` **stp_set_yellow** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the yellow adjustment of the print job (range: 0.0-4.0; default: 1.0). This currently adjusts the gamma of the yellow curve. It is scaled by the yellow adjustment for the printer model in question.

`float` **stp_get_saturation** (`const stp_vars_t` *vv*) Function
`void` **stp_set_saturation** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the saturation of the print job (range: 0.0-9.0; default: 1.0). Saturation of 0.0 produces grayscale output using composite (CMY or CMYK, as appropriate for the printer) color.

`float` **stp_get_density** (`const stp_vars_t` *vv*) Function
`void` **stp_set_density** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the density of the print job (range: 0.0-2.0; default: 1.0). This adjusts the amount of ink deposited in a linear fashion for all channels. It is scaled by the density appropriate for the choice of printer, resolution, paper type, and other factors that the driver may deem appropriate.

`float` **stp_get_app_gamma** (`const stp_vars_t` *vv*) Function
`void` **stp_set_app_gamma** (`stp_vars_t` *vv*, `float` *val*) Function

Get or set the gamma of the input (i. e. what the driving application uses).

`void *` **stp_get_lut** (`const stp_vars_t` *vv*) Function
`void` **stp_set_lut** (`stp_vars_t` *vv*, `void *`*val*) Function

Get or set the color lookup table for the print job. This is useful outside of the library for computing a preview of the printed result. The lookup table itself should be treated as an opaque handle.

unsigned char * **stp_get_cmap** (const stp_vars_t *vv*) Function
void **stp_set_cmap** (stp_vars_t *vv*, unsigned char **val*) Function

Get or set the color map for the print job. This is a table of R,G,B values for 8-bit indexed input. This may be moved outside of the library in the future; in this case, the front end would be required to do its own mapping and supply true-color RGB to the driver.

The following methods are used to perform output and error reporting by the driver. The driver supplies a stream of output bytes; the front end is responsible for providing methods that accept this output and handle it appropriately.

stp_outfunc_t **stp_get_outfunc** (const stp_vars_t *vv*) Function
void **stp_set_outfunc** (const stp_vars_t *vv*, stp_outfunc_t *val*) Function
void * **stp_get_outdata** (const stp_vars_t *vv*) Function
void **stp_set_outdata** (stp_vars_t *vv*, void **val*) Function

stp_outfunc_t Data type

```
typedef void (*stp_outfunc_t) (void *data,
                    const char *buffer, size_t bytes);
```

Get or set the output function the driver will use. The front end must supply a suitable function for accepting the output data. The `stp_set_outdata` method provides a way of passing an appropriate object to the output function.

stp_outfunc_t **stp_get_errfunc** (const stp_vars_t *vv*) Function
void **stp_set_errfunc** (const stp_vars_t *vv*, stp_outfunc_t *val*) Function
void * **stp_get_errdata** (const stp_vars_t *vv*) Function
void **stp_set_errdata** (stp_vars_t *vv*, void **val*) Function

Get or set the error reporting function that the driver will use. This is used to report errors or debugging information, and must be supplied. A typical errfunc will simply print whatever it's passed to stderr.

/* * hue_map is an array of 49 doubles representing the mapping of hue * from (0..6) to (0..6) in increments of .125. The hue_map is in CMY space, * so hue=0 is cyan. */ typedef void (*stp_convert_t)(const stp_vars_t vars, const unsigned char *in, unsigned short *out, int *zero_mask, int width, int bpp, const unsigned char *cmap, const double *hue_map, const double *lum_map, const double *sat_map);

void **stp_merge_printvars** (stp_vars_t *user*, const stp_vars_t *print*) Function
void **stp_allocate_lut** (stp_vars_t *v*, size_t *steps*) Function
void **stp_free_lut** (stp_vars_t *v*) Function
void **stp_compute_lut** (stp_vars_t *v*, size_t *steps*) Function

4.3 Dither functions

size_t **stp_dither_algorithm_count** (void) Function
const char * **stp_dither_algorithm_name** (int *id*) Function
const char * **stp_default_dither_algorithm** (void) Function

4.4 Options functions

`void *` **stp_option_t** — Data type

This is an opaque data type, whose structure is not visible to the user.

`void` **stp_set_option**(`stp_vars_t` *v*, `const char *`*name*, `const char *`*data*, `int` *bytes*) — Function

`void` **stp_clear_option** (`stp_vars_t` *v*, `const char *`*name*) — Function

`void` **stp_clear_all_options** (`stp_vars_t` *v*) — Function

`size_t` **stp_option_count** (`const stp_vars_t` *v*) — Function

`const stp_option_t` **stp_get_option_by_index** (`const stp_vars_t` *v*, `size_t` *idx*) — Function

`const stp_option_t` **stp_get_option_by_name** (`const stp_vars_t` *v*, `const char *`*name*) — Function

`const char *` **stp_option_data** (`const stp_option_t` *option*) — Function

Note: not null delimited!

`const char *` **stp_option_name** (`const stp_option_t` *option*) — Function

`size_t` **stp_option_length** (`const stp_option_t` *option*) — Function

4.5 Paper functions

`void *` **stp_papersize_t** — Data type

This is an opaque data type, whose structure is not visible to the user.

stp_papersize_unit_t — Data type

```
typedef enum papersize_unit
{
  PAPERSIZE_ENGLISH,
  PAPERSIZE_METRIC
} stp_papersize_unit_t;
```

`int` **stp_known_papersizes** (`void`) — Function

`const stp_papersize_t` **stp_get_papersize_by_name** (`const char *`*name*) — Function

`const stp_papersize_t` **stp_get_papersize_by_size** (`int` *l*, `int` *w*) — Function

`const stp_papersize_t` **stp_get_papersize_by_index** (`int` *index*) — Function

`const char *` **stp_papersize_get_name** (`const stp_papersize_t` *pt*) — Function

`unsigned` **stp_papersize_get_width** (`const stp_papersize_t` *pt*) — Function

`unsigned` **stp_papersize_get_height** (`const stp_papersize_t` *pt*) — Function

`unsigned` **stp_papersize_get_top** (`const stp_papersize_t` *pt*) — Function

`unsigned` **stp_papersize_get_left** (`const stp_papersize_t` *pt*) — Function

`unsigned` **stp_papersize_get_bottom** (`const stp_papersize_t` *pt*) — Function

`unsigned` **stp_papersize_get_right** (`const stp_papersize_t` *pt*) — Function

`stp_papersize_unit_t` **stp_papersize_get_unit** (`const stp_papersize_t` *pt*) — Function

4.6 Printer functions

void * **stp_printer_t** — Data type

This is an opaque data type, whose structure is not visible to the user.

int **stp_known_printers** (void) — Function

const stp_printer_t **stp_get_printer_by_index** (int *idx*) — Function

const char * **stp_printer_get_long_name** (const stp_printer_t *p*) — Function

const stp_printer_t **stp_get_printer_by_long_name** (const char **long_name*) — Function

const stp_printer_t **stp_get_printer_by_driver** (const char **driver*) — Function

int **stp_get_printer_index_by_driver** (const char **driver*) — Function

const char * **stp_printer_get_driver** (const stp_printer_t *p*) — Function

int **stp_printer_get_model** (const stp_printer_t *p*) — Function

const stp_printfuncs_t * **stp_printer_get_printfuncs** (const stp_printer_t *p*) — Function

stp_printfuncs_t — Data type

```
typedef struct
{
  char  **(*parameters)(const stp_printer_t printer,
                        const char *ppd_file,
                        const char *name, int *count);
  void  (*media_size)(const stp_printer_t printer,
                      const stp_vars_t v, int *width,
                      int *height);
  void  (*imageable_area)(const stp_printer_t printer,
                          const stp_vars_t v,
                          int *left, int *right,
                          int *bottom, int *top);
  void  (*limit)(const stp_printer_t printer,
                 const stp_vars_t v,
                 int *width, int *height);
  void  (*print)(const stp_printer_t printer,
                 stp_image_t *image, const stp_vars_t v);
  const char *(*default_parameters)(const stp_printer_t printer,
                                    const char *ppd_file,
                                    const char *name);
  void  (*describe_resolution)(const stp_printer_t printer,
                               const char *resolution,
                               int *x, int *y);
  int   (*verify)(const stp_printer_t p, const stp_vars_t v);
} stp_printfuncs_t;
```

stp_image_t — Data type

```
typedef struct stp_image
{
  void (*init)(struct stp_image *image);
```

```
    void (*reset)(struct stp_image *image);
    void (*transpose)(struct stp_image *image);
    void (*hflip)(struct stp_image *image);
    void (*vflip)(struct stp_image *image);
    void (*crop)(struct stp_image *image,
                 int left, int top, int right,
                 int bottom);
    void (*rotate_ccw)(struct stp_image *image);
    void (*rotate_cw)(struct stp_image *image);
    void (*rotate_180)(struct stp_image *image);
    int  (*bpp)(struct stp_image *image);
    int  (*width)(struct stp_image *image);
    int  (*height)(struct stp_image *image);
    void (*get_row)(struct stp_image *image,
          unsigned char *data, int row);
    const char *(*get_appname)(struct stp_image *image);
    void (*progress_init)(struct stp_image *image);
    void (*note_progress)(struct stp_image *image,
          double current, double total);
    void (*progress_conclude)(struct stp_image *image);
    void *rep;
  } stp_image_t;
```

This is an abstract data type for interfacing with the program which created the image.

`const stp_vars_t` **stp_printer_get_printvars** (`const stp_printer_t` *p*) — Function

4.7 Settings functions

`stp_convert_t` **stp_choose_colorfunc** (`int` *output_type*, `int` *image_bpp*, `const unsigned char *`*cmap*, `int *`*out_bpp*, `const stp_vars_t` *v*) — Function

`void` **stp_compute_page_parameters** (`int` *page_right*, `int` *page_left*, `int` *page_top*, `int` *page_bottom*, `double` *scaling*, `int` *image_width*, `int` *image_height*, `stp_image_t *`*image*, `int *`*orientation*, `int *page_width`, `int *`*page_height*, `int *`*out_width*, `int *`*out_height*, `int *`*left*, `int *`*top*) — Function

`const stp_vars_t` **stp_default_settings** (`void`) — Function

`const stp_vars_t` **stp_maximum_settings** (`void`) — Function

`const stp_vars_t` **stp_minimum_settings** (`void`) — Function

4.8 Version functions

`const char *` **stp_check_version** (`unsigned int` *required_major*, unsigned int *required_minor*, unsigned int *required_micro*) — Function

This function checks whether the version of libgimpprint that the program is linked with is equal to the version number passed to it. If the version is the same, the function returns `NULL`. If any of the version numbers do not match (i.e. the library version is too old or too new), a string containing a desription of the difference is returned. The first error found is returned. The function checks in the order major, minor, micro.

4.8.1 Version macros

GIMPPRINT_CHECK_VERSION (*major,minor,micro*) Macro

This macro returns zero if the version of the libgimpprint headers are greater or equal to the version given as an argument. It returns nonzero if the version of the libgimpprint headers are less than the argument.

GIMPPRINT_MAJOR_VERSION Macro
GIMPPRINT_MINOR_VERSION Macro
GIMPPRINT_MICRO_VERSION Macro
GIMPPRINT_CURRENT_INTERFACE Macro
GIMPPRINT_BINARY_AGE Macro
GIMPPRINT_INTERFACE_AGE Macro

These macros are integers holding the version numbers. They should be used for compile-time checking only. To check version numbers at run-time, use the equivalent variables. Note that at present (4.1.x development branch) the library interface version numbers are not used.

4.8.2 Version variables

`const unsigned int` **gimpprint_major_version** Variable
`const unsigned int` **gimpprint_minor_version** Variable
`const unsigned int` **gimpprint_micro_version** Variable
`const unsigned int` **gimpprint_current_interface** Variable
`const unsigned int` **gimpprint_binary_age** Variable
`const unsigned int` **gimpprint_interface_age** Variable

These variables hold the library version numbers. Because the version of the library may change on a system using shared libraries, these should be used instead of the equivalent macros when checking the library version at run-time. Note that library interface version numbers are not used in the development branch, but are in the stable branch.

5 Programs

This chapter of the manual describes the use of some of the programs which use the GIMP-Print library (libgimpprint). Note that there is now a user manual in DocBook/SGML format, currently provided in HTML, PostScript and PDF formats which is distributed with GIMP-Print. This manual currently covers the use of the GIMP Print plugin and CUPS drivers.

5.1 The GIMP Print plugin

The GIMP Print plugin is the printing facility for the GNU Image Manipulation Program[1]. This section examines the features offered by the Print plugin.

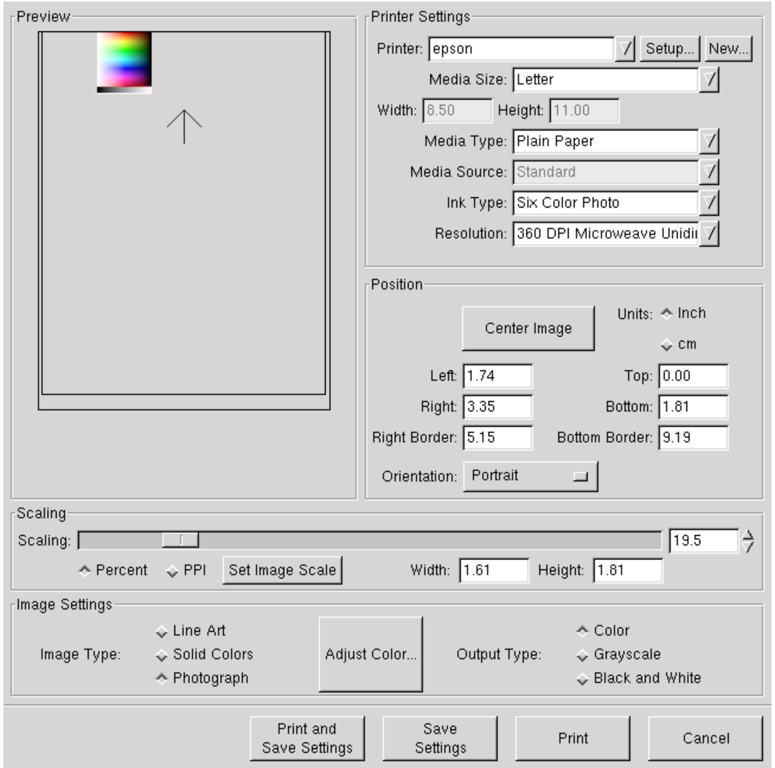
The main window is divided into five panes:

5.1.1 Preview

The Preview pane contains a positioning widget that allows interactively positioning the output on the page. It contains an outer border, representing the sheet of paper; an inner border, representing the printable area of the printer; an arrow, pointing to the top of the page (the end that's fed into the printer); and a black rectangle, representing the position of the image on the page. The image can be moved around on the paper. When the first (left) button is used, the image is moved in screen pixels; when any other button is used,

[1] `http://www.gimp.org`

the image is moved in points[2]. The arrow resizes depending upon the media size chosen; the shaft of the arrow is always equal to one inch on the output.



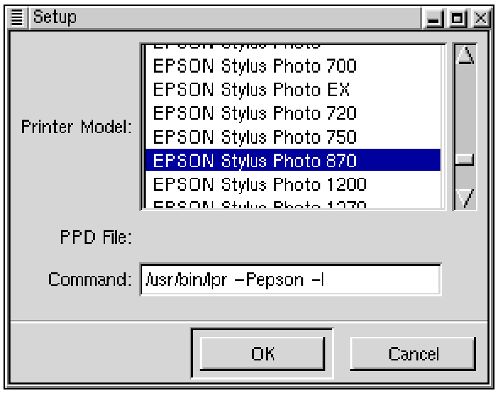
5.1.2 Printer Settings

The Printer Settings pane contains a dropdown menu for selecting a printer to print to. There is a special 'printer' named '`File`' that allows you to choose a file to print to, rather than a printer queue. The Setup box to the right allows specification of a printer type, a PPD file[3], and the command to be used to print. Each distinct printer in the Printer list can have different settings applied to it. Below that is a combo box allowing choice of media size. The choices are constrained to those that the printer supports. Below that are

[2] The output resolution of the plugin.

[3] For Postscript printers.

dropdown menus for choosing media type (what kind of paper), media source (what input tray), ink type, and resolution. All of these settings are printer-specific.



5.1.3 Position

The Position pane contains various widgets to place the image on the paper. These widgets work in conjunction with the Preview pane. At the top of the pane is a button to center the image on the paper (not on the printable area), and on either side buttons to center vertically and horizontally. Below these are four boxes that allow entry of the left, top, right, and bottom of the image. These positions are relative to the top left of the paper[4]. There are two additional boxes that allow specification of the right margin and bottom margin if you prefer; these are relative to the bottom right corner of the paper. Any of these may have values entered into them; the preview image will be moved appropriately.

Note: These entries do not resize the image.

Finally, there is a pick box for orientation (landscape or portrait). There is an '`Auto`' mode that picks the orientation that yields the orientation that best matches that of the image to be printed.

5.1.4 Scaling

The Scaling pane contains a slider that allows scaling of the image. The image can be scaled in either percent of the printable area (**not** the page in this case) or pixels per inch (PPI) via a radio button below the slider. PPI allows matching image resolution to printer resolution. The image may be scaled using either method to between 5 and 100% of the imageable area. It is not possible to crop with the Print plugin. In Percent mode, the image is scaled so that neither axis will be longer than the percent of the printable area specified. For example, if you print an image at 20%, it will be possible to tile the image 5 times on one axis and at least 5 times on the other. To the right of the radio button is a button called Set Image Scale. This sets the scaling to PPI, and sets the resolution as closely as possible to the resolution stored in the image. To the right of the Set Image Scale button are two boxes that allow entry of width and height of the image. These set the scaling mode to PPI. Specifying one automatically sets the other, and the image is repositioned as needed to prevent it from falling off the edge of the page.

[4] Again, that's relative to the paper corner, not the printable area, which is usually smaller.

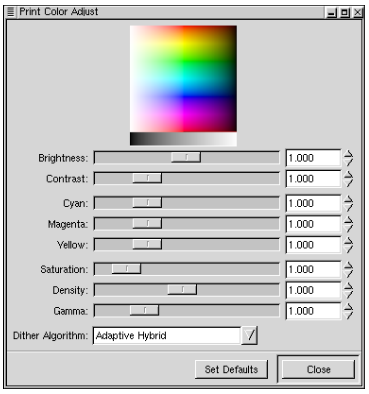
To its right is a button group that allows choosing English (inch) units or metric (centimeter) units.

5.1.5 Image Settings

The Image Settings pane allows choice of Line Art, Solid Colors, or Photograph image type. Line art or Solid Colors should be used for graphics containing mostly solid areas of color. They're very similar to each other. Photograph mode dithers more slowly, but produces more accurate colors. To the right of these three radio buttons is a button called Adjust Color. This pops up a new window that controls various output quality settings. That will be described separately. Finally, there is a choice of Black and White, Color and Monochrome output. Monochrome output can be used to print absolute black and white very quickly.

5.1.5.1 Adjust Output

The Adjust Output button button pops up a non-modal dialog that allows adjustment of various parameters related to the print quality. These are independent of the controls within the GIMP itself and only affect the print.



At the top of the window is a thumbnail of the image that changes to reflect the color settings of the image. This enables you to get an idea of how the image will print out as you adjust settings.

Below that there are eight sliders:

Brightness
: (0–2.0, default 1.0) Adjust the brightness of the image.

Contrast
: (0–4.0, default 1.0) Adjust the output contrast.

Cyan, Magenta, Yellow
: (0–4.0, default 1.0) Adjust the cyan, magenta, and yellow in the output. These should not normally need to be adjusted very much; even very small adjustments can go quite a long way to restoring color balance.

Saturation
: (0–9.0, default 1.0) Adjust the color brilliance (saturation) of the output. Saturation of 0 means pure gray scale, with no color. Saturation of 9.0 will make just about anything but pure grays brilliantly colored.

Density (0.1–2.0, default 1.0) Adjust the density (amount of ink) in the print. The density is automatically corrected for the particular printer, resolution, and in some cases paper choices. If solid black in the input is not solid in the print, the density needs to be increased; if there is excessive ink bleed-through and muddy dark colors, the density should be decreased.

Note: the density will not increase beyond a certain amount no matter what the slider is set to.

Gamma (0.1–4.0, default 1.0) Adjust the output gamma. The gamma value is automatically corrected for the choice of printer; this is used if you believe the automatic setting is incorrect.

5.1.5.2 Dither Algorithm

There is also a selection box for the dither algorithm to be used in the pop-up dialog. There are currently seven choices:

Adaptive Hybrid
Adaptive Hybrid usually yields the best output quality; it chooses a modified Floyd-Steinberg error diffusion algorithm or ordered dithering depending upon the image characteristics.

Ordered Ordered uses a pure ordered dither. It generally yields excellent quality for simple black and white or four color printers without variable drop size or drop modulation; it is not recommended if high quality is desired on six color printers. It is considerably faster than Adaptive Hybrid.

Fast Fast also uses a pure ordered dither, but uses a very simple black model and makes no attempt to handle multi-level (6-color, variable drop size, or drop modulation) at all cleanly. It is substantially faster than Ordered dither. The quality tends to be quite poor except on simple four color printers. On three color printers, quality is probably competitive with anything else.

Very Fast Very Fast is similar to Fast, except that it uses a very simple dither matrix that can be looked up much more quickly than the matrix used in the Fast dither. For simple pure black and white images dominated by horizontal and vertical lines, this may actually yield the best results; for other types of image, the quality will be poor.

Adaptive Random
Adaptive Random is similar to Adaptive Hybrid, except that the modifications to the Floyd-Steinberg algorithm are slightly different. This is slower than Adaptive Hybrid on most systems. For some images the quality may be better than Adaptive Hybrid, but generally Adaptive Hybrid should yield slightly superior images.

Hybrid Floyd-Steinberg
Hybrid Floyd-Steinberg uses the modified Floyd-Steinberg algorithm of Adaptive Hybrid on the entire image. Generally, the results are poor in pale regions.

Random Floyd-Steinberg
: Random Floyd-Steinberg uses the modified Floyd-Steinberg algorithm of Adaptive Random on the entire image. Generally, the results are poor in pale regions.

5.1.6 Action Buttons

The last pane contains four action buttons:

Print and Save Settings
: Immediately print the image (or, if the File printer is chosen, display a file selection window to pick the output file), and save all current settings for all printers.

Save Settings
: Immediately save the settings, and continue working in the Print plugin.

Print
: Immediately print the image (or, if the '`File`' printer is chosen, display a file selection window to pick the output file), but do not save settings.

Cancel
: Immediately quit without saving settings or printing.

5.2 Ghostscript driver

5.3 CUPS driver

6 Reporting Bugs

If you find a bug in GIMP-Print or have any suggestions for modification or improvement, please send electronic mail to the GIMP-Print bug reporting address[1]. Include the version number, which you can find by running '`gimpprint-config --version`'. Also include in your message the output that the program produced and the output you expected, if applicable, otherwise the best description of the problem that you can provide.

If you have other questions, comments or suggestions about GIMP-Print, contact the developers via electronic mail to the GIMP-Print mailing list[2]. They will try to help you out, although they may not have time to fix your problems.

---

[1] `gimp-print-devel@lists.sourceforge.net`

[2] `gimp-print-devel@lists.sourceforge.net`

7 Appendices

Appendix A Dithering

The dithering code in 'print-dither.c' attempts to reproduce various shades of gray (or all colors) from only a few different inks (black, cyan, magenta, yellow, and sometimes light cyan and light magenta). The dots can't vary in darkness or size (except for certain special printers), and so we need to lay down a certain fraction of dots to represent each distinct level.

This sounds straightforward; in practice, it isn't. Completely random distribution of dots (simple probabilistic dithering) would create grainy clumps and light spots. The smoothest pattern results from an equidistant spacing of dots. Approximating this requires sophisticated algorithms. We have two dithering algorithms, an ordered dither algorithm that uses a grid (matrix) to decide whether to print, and a modified Floyd-Steinberg error diffusion algorithm that uses a grid in a slightly different way.

We currently have three dithering functions:

1. `dither_fastblack` produces pure black or white from a pre-dithered input. This is used for two purposes: for printing pure black and white very quickly (e. g. text), and for printing pre-screened monochrome output that was rasterized externally.
2. `dither_black` produces black from grayscale input. The new dither_black can produce either a single or multiple levels of black, for printers supporting variable dot size.
3. `dither_cmyk` produces 3, 4, 5, 6, or 7 color output (CMY, CMYK, CcMmYK, CcMmYy, CcMmYyK, or any variants). The new routine can handle single or multiple levels of each color.

There is a choice of dithering algorithms. Four of them are based on a basic error diffusion, with a few tweaks of my own. The other one is 'ordered'. However, they all share the basic operation in common. First, the algorithm picks what kind of dot (if there are multiple dot sizes and/or tones that may be picked) is the candidate to be printed. This decision is made based on the darkness at the point being dithered. Then, it decides whether the dot will be printed at all. What this is based on depends upon which algorithm family we use. This is all described in more detail below.

Ordered dithering works by comparing the value at a given point with the value of a tiled matrix. If the value at the point is greater than the value in the matrix, the dot is printed. The matrix should consist of a set of evenly spaced points between 0 and the upper limit. The choice of matrix is very important for print quality. A good dither matrix will emphasize high frequency components, which distributes dots evenly with a minimum of clumping. The matrices used here are all simple matrices that are expanded recursively to create larger matrices with the same kind of even point distribution. This is described below.

Note that it is important to use different matrices for the two sub-operations, because otherwise the choice about whether to print and the choice of dot size will be correlated. The usual result is that the print is either too dark or too light, but there can be other problems.

Ordered dithering works quite well on single dot size, four color printers. It has not been well tested on four color, variable dot size printers. It should be avoided on six color printers.

Error diffusion works by taking the output error at a given pixel and "diffusing" it into surrounding pixels. Output error is the difference between the amount of ink output and the input level at each pixel. For simple printers, with one or four ink colors and only one dot size, the amount of ink output is either 65536 (i. e. full output) or 0 (no output). The difference between this and the input level is the error. Normal error diffusion adds part of this error to the adjoining pixels in the next column and the next row (the algorithm simply scans each row in turn, never backing up). The error adds up until it reaches a threshold (half of the full output level, or 32768), at which point a dot is output, the output is subtracted from the current value, and the (now negative) error is diffused similarly.

Error diffusion works quite well in general, but it tends to generate artifacts which usually appear as worm-like lines or areas of anomalous density. I have devised some ways, as described below, of ameliorating these artifacts.

There are two sub-classes of error diffusion that we use here, 'random' and 'hybrid'. One of the techniques that we use to ameliorate the artifacts is to use a fuzzy threshold rather than the hard threshold of half of the output level. Random error diffusion uses a pseudo-random number to perturb the threshold, while hybrid error diffusion uses a matrix. Hybrid error diffusion worked very poorly in 3.1.3, and I couldn't figure out why until I found a bug. It now works very well.

There is one additional variant (on both sub-classes), called 'adaptive hybrid' and 'adaptive random'. The adaptive variant takes advantage of the fact that the patterns that ordered dithering create are less visible at very low densities, while the artifacts created by error diffusion are more objectionable at low densities. At low densities, therefore, it uses ordered dithering; at higher densities it uses error diffusion.

Handling multiple output levels makes life a bit more complicated. In principle, it shouldn't be much harder: simply figure out what the ratio between the available output levels is and have multiple thresholds. In practice, getting these right involves a lot of trial and error. The other thing that's important is to maximize the number of dots that have some ink. This will reduce the amount of speckling. More on this later.

The next question: how do we handle black when printing in color? Black ink is much darker than colored inks. It's possible to produce black by adding some mixture of cyan, magenta, and yellow—in principle. In practice, the black really isn't very black, and different inks and different papers will produce different color casts. However, by using CMY to produce gray, we can output a lot more dots! This makes for a much smoother image. What's more, one cyan, one magenta, and one yellow dot produce less darkness than one black dot, so we're outputting that many more dots. Better yet, with 6 or 7 color printers, we have to output even more light ink dots. So Epson Stylus Photo printers can produce really smooth grays—if we do everything right. The right idea is to use CMY at lower black levels, and gradually mix in black as the overall amount of ink increases, so the black dots don't really become visible within the ink mass.

Variable dot sizes are handled by dividing the range between 0 and 65536 into segments. Each segment can either represent a range in which all of one kind of ink (color and/or dot size) is used, with varying amounts of ink, or a transition region between inks, in which equal numbers of dots are printed but the amount of each ink will be adjusted throughout the range. Each range is represented by four numbers:

1. bottom of the range

2. top of the range
3. value of the lighter ink
4. value of the darker ink

In addition, the bit patterns and which type of ink are also represented, but they don't affect the actual algorithm.

As mentioned above, the basic algorithm is the same whether we use ordered dither or error diffusion. We perform the following steps on each color of each pixel:

1. Compute the value of the particular color we're printing. This isn't usually the pure CMY value; it's adjusted to improve saturation and to limit the use of black in light toned regions (to avoid speckling).
2. Find the range containing this value.
3. Compute where this value lies within the range. We scale the endpoints between 0 and 65536 for this purpose. So for example, if the bottom of the range is 10,000 and the top of the range is 20,000, and the value is 12,500, we're 1/4 of the way between the bottom and the top of the range, so our scale point is 16384.
4. Compute the "virtual value". The virtual value is the distance between the value of the lighter and the value of the darker ink. So if the value of the light ink is 32768 and the dark ink is 65536, we compute a virtual value scaled appropriately between these two values, which is 40960 in this case.
5. Using either error diffusion or ordered dither, the standard threshold is 1/2 of the value (20480 in this case). Using ordered dither, we want to compute a value between 0 and 40960 that we will compare the input value against to decide whether to print. Using pure error diffusion, we would compare the accumulated error against 20480 to decide whether to print. In practice, we use the same matrix method to decide whether to print. The correct amount of ink will be printed this way, but we minimize the squiggly lines characteristic of error diffusion by dithering the threshold in this fashion. A future enhancement will allow us to control the amount of dithering applied to the threshold.

The matrices were generated by Thomas Tonino `<ttonino@bio.vu.nl>` with an algorithm of his devising. The algorithm is designed to maximize the spacing between dots at any given density by searching the matrix for holes and placing a dot in the largest available hole. It requires careful selection of initial points to achieve good results, and is very time consuming. For best results, a different matrix must be used for modes with 2:1 aspect ratio (e.g. 1440x720) than for 1:1 (e. g. 720x720). It is essential with any of these matrices that every point be used. Skipping points generates low-frequency noise.

It's essential to use different matrices for deciding whether to print and for deciding what color (dark or light) to print. This should be obvious; the decision about whether to print at all should be as independent as possible from the decision about what color to print, because any bias will result in excess light or dark ink being printed, shifting the tonal balance. We actually use the same matrices, but we shift them vertically and horizontally. Assuming that the matrices are not self-correlated, this will yield good results.

The ranges are computed from a list of ink values (between 0 and 1 for each possible combination of dot size and ink tone, where the value represents the darkness of the ink) and the desired maximum density of the ink. This is done in dither_set_ranges, and needs more documentation.

I stated earlier that I've tweaked the basic error diffusion algorithm. Here's what I've done to improve it:

1. We use a variable threshold to decide when to print, as discussed above. This does two things for us: it reduces the slightly squiggly diagonal lines that are the mark of error diffusion; and it allows us to lay down some ink even in very light areas near the edge of the image. The squiggly lines that error diffusion algorithms tend to generate are caused by the gradual accumulation of error. This error is partially added horizontally and partially vertically. The horizontal accumulation results in a dot eventually being printed. The vertical accumulation results in a dot getting laid down in roughly the same horizontal position in the next row. The diagonal squigglies result from the error being added to pixels one forward and one below the current pixel; these lines slope from the top right to the bottom left of the image.

   Error diffusion also results in pale areas being completely white near the top left of the image (the origin of the printing coordinates). This is because enough error has to accumulate for anything at all to get printed. In very pale areas it takes quite a long time to build up anything printable at all; this results in the bare spots.

   Randomizing the threshold somewhat breaks up the diagonals to some degree by randomizing the exact location that the accumulated output crosses the threshold. It reduces the false white areas by allowing some dots to be printed even when the accumulated output level is very low. It doesn't result in excess ink because the full output level is still subtracted and diffused.

   Excessive randomization leads to blobs at high densities. Therefore, as the density increases, the degree of randomization decreases.

2. Alternating scan direction between rows (first row is scanned left to right, second is scanned right to left, and so on). This also helps break up white areas, and it also seems to break up squigglies a bit. Furthermore, it eliminates directional biases in the horizontal direction. This isn't necessary for ordered dither, but it doesn't hurt either.

3. Diffusing the error into more pixels. Instead of diffusing the entire error into $(X+1, Y)$ and $(X, Y+1)$, we diffuse it into $(X+1, Y)$, $(X+K, Y+1)$, $(X, Y+1)$, $(X - K, Y+1)$ where K depends upon the output level (it never exceeds about 10 dots, and is greater at higher output levels). This really reduces squigglies and graininess. The amount of this spread can be controlled; for line art, it should be less than for photographs (of course, line art doesn't usually contain much light color, but the **error** value can be small in places!) In addition to requiring more computation, a wide ink spread results in patterning at high dot densities (note that the dot density can be high even in fairly pale regions if multiple dot sizes are in use).

4. Don't lay down any colored ink if we're laying down black ink. There's no point; the colored ink won't show. We still pretend that we did for purposes of error diffusion (otherwise excessive error will build up, and will take a long time to clear, resulting in heavy bleeding of ink into surrounding areas, which is very ugly indeed), but we don't bother wasting the ink. How well this will do with variable dot size remains to be seen.

5. Oversampling. This is how to print 1440x720 with Epson Stylus printers. Printing full density at 1440x720 will result in excess ink being laid down. The trick is to print only every other dot. We still compute the error as though we printed every dot. It turns out that randomizing which dots are printed results in very speckled output. This can

be taken too far; oversampling at 1440x1440 or 1440x2880 virtual resolution results in other problems. However, at present 1440x1440 (which is more accurately called "1440x720 enhanced", as the Epson printers cannot print 1440 rows per inch) does quite well, although it's slow.

What about multiple output levels? For 6 and 7 color printers, simply using different threshold levels has a problem: the pale inks have trouble being seen when a lot of darker ink is being printed. So rather than just using the output level of the particular color to decide which ink to print, we look at the total density (sum of all output levels). If the density's high enough, we prefer to use the dark ink. Speckling is less visible when there's a lot of ink, anyway. I haven't yet figured out what to do for multiple levels of one color.

You'll note that I haven't quoted a single source on color or printing theory. I simply did all of this empirically.

There are various other tricks to reduce speckling. One that I've seen is to reduce the amount of ink printed in regions where one color (particularly cyan, which is perceived as the darkest) is very pale. This does reduce speckling all right, but it also results in strange tonal curves and weird (to my eye) colors.

Before any dither routine is used, `init_dither()` must be called. This takes three arguments: the input width (number of pixels in the input), the output width (number of pixels in the output), and a `vars_t` structure containing the parameters for the print job.

`init_dither()` returns a pointer to an opaque object representing the dither. This object is passed as the first argument to all of the dither-related routines.

After a page is fully dithered, `free_dither()` must be called to free the dither object and perform any cleanup. In the future, this may do more (such as flush output). This arrangement permits using these routines with programs that create multiple output pages, such as GhostScript.

The dithering routines themselves have a number of control knobs that control internal aspects of the dithering process. These knobs are accessible via a number of functions that can be called after `init_dither()`.

- `dither_set_density()` takes a double between 0 and 1 representing the desired ink density for printing solid colors. This is used in a number of places in the dithering routine to make decisions.
- `dither_set_black_density()` takes a double between 0 and 1 representing the desired ink density for printing black ink in color printing. This is used to balance black against color ink. By default, this is equal to the density set by `dither_set_density()`. By setting it higher, more black ink will be printed. For example, if the base density is .4 and the black density is .8, twice as much black ink will be printed as would otherwise be called for.

  This is not used when printing in monochrome. When printing monochrome, the base density (`dither_set_density`) should be adjusted appropriately.
- `dither_set_ink_budget()` takes an unsigned number representing the most ink that may be deposited at a given point. This number is arbitrary; the limit is computed by summing the size of each ink dot, which is supplied as a parameter in `dither_set_`*X*`_ranges`. By default, there is no limit.

- `dither_set_black_lower()` takes a double that should be between 0 and 1 that represents the lowest density level at which black ink will start to mix in with colored ink to generate grays. The lower this is, the less density is required to use black ink. Setting this too low will result in speckling from black dots, particularly on 6 and 7 color printers. Setting this too high will make it hard to get satisfactory black or may result in sharp transition between blended colors and black. Default: 0.0468.

  It is important to note that since the density scale is never linear (and since this value is adjusted via other things happening during the dithering process) that this does not mean that 95% gray will use any black ink. At this setting, there will be no black ink used until about 50% gray.

  This only applies to color mode.

  This value should be set lower for printers capable of variable dot size, since more dots can be laid down close to each other.

- `dither_set_black_upper()` takes a double that should be between 0 and 1 that represents the highest density level at which colored inks will be mixed to create gray. Setting this too low will result in speckly dark grays because there is not enough ink to fill all the holes, or sharp transition between blended colors and black if it is too close to the value of dither_set_black_upper(). Setting this too high will result in poor black and dark tone quality. Default: 0.5. This results in 10% and darker grays being printed with essentially all black.

  This only applies to color mode.

- `dither_set_black_levels()` takes three doubles that represent the amount of cyan, magenta, and yellow respectively that are blended to create gray. The defaults are 1.0 for each, which is probably too low for most printers. These values are adjusted to create a good gray balance. Setting these too low will result in pale light and midtone grays, with a sharp transition to darker tones as black mixes in. Setting them too high will result in overly dark grays and use of too much ink, possibly creating bleed-through.

  This only applies to color mode.

- `dither_set_randomizers()` takes four integer values representing the degree of randomness used for cyan, magenta, yellow, and black. This is used to allow some printing to take place in pale areas. Zero is the most random; greater than 8 or so gives very little randomness at all. Defaults are 0 for cyan, magenta, and yellow, and 4 for black. Setting the value for black too low will result in black speckling in pale areas. Setting values too high will result in pale areas getting no ink at all.

  This currently only applies to single dot size in color and black. It should be extended to operate in variable dot size mode, although actually applying it correctly will be tricky.

- `dither_set_ink_darkness()` takes three doubles representing the contribution to perceived darkness of cyan, magenta, and yellow. This is used to help decide when to switch between light and dark inks in 6 and 7 color printers (with light cyan, light magenta, and possibly light yellow). Setting these too low will result in too much light ink being laid down, creating flat spots in the darkness curves and bleed-through. Setting them too high will result in dark ink being used in pale areas, creating speckle. The defaults are .4 for cyan, .3 for magenta, and .2 for yellow. Dark cyan will show against yellow much more than dark magenta will show against cyan, since the cyan appears much darker than the yellow.

- `dither_set_light_inks()` takes three doubles between 0 and 1 representing the ratio in darkness between the light and dark versions of the inks. Setting these too low will result in too much dark ink being used in pale areas, creating speckling, while setting them too high will result in very smooth texture but too much use of light ink, resulting in flat spots in the density curves and ink bleed-through. There are no defaults. Any light ink specified as zero indicates that there is no light ink for that color.

  This only applies to 6 and 7 color printers in single dot size color mode, and only to those inks which have light versions (usually cyan and magenta).
- `dither_set_ink_spread()` takes a small integer representing the amount of ink spread in the dither. Larger numbers mean less spread. Larger values are appropriate for line art and solid tones; they will yield sharper transitions but more dither artifacts. Smaller values are more appropriate for photos. They will reduce resolution and sharpness but reduce dither artifacts up to a point. A value of 16 or higher implies minimum ink spread at any resolution no matter what the overdensity. A value of 14 is typical for photos on single dot size, 6 color printers. For 4 color printers, subtract 1 (more spread; the dots are farther apart). For variable dot size printers, add 1 (more small dots are printed; less spread is desirable).
- `dither_set_adaptive_divisor()` takes a float representing the transition point between error diffusion and ordered dither if adaptive dithering is used. The float is a fraction of the printing density. For example, if you wish the transition to be at 1/4 of the maximum density (which works well on simple 4-color printers), you would pass .25 here. With six colors and/or with multiple dot sizes, the values should be set lower.
- `dither_set_transition()` takes a float representing the exponent of the transition curve between light and dark inks/dot sizes. A value less than 1 (typical when using error diffusion) mixes in less dark ink/small dots at lower ends of the range, to reduce speckling. When using ordered dithering, this must be set to 1.
- `dither_set_`*X*`_ranges_simple` (*X*=‘`c`’, ‘`m`’, ‘`y`’, or ‘`k`’) describes the ink choices available for each color. This is useful in typical cases where a four color printer with variable dot sizes is in use. It is passed an array of doubles between (0, 1] representing the relative darkness of each dot size. The dot sizes are assigned bit patterns (and ink quantities, see `dither_set_ink_budget()` above) from 1 to the number of levels. This also requires a density, which is the desired density for this color. This density need not equal the density specified in `dither_set_density()`. Setting it lower will tend to print more dark ink (because the curves are calculated for this color assuming a lower density than is actually supplied).
- `dither_set_`*X*`_ranges` (*X*=‘`c`’, ‘`m`’, ‘`y`’, or ‘`k`’) describes in a more general way the ink choices available for each color. For each possible ink choice, a bit pattern, dot size, value (i. e. relative darkness), and whether the ink is the dark or light variant ink is specified.

—Robert Krawitz `<rlk@alum.mit.edu>` May 8, 2000

Appendix B Weaving for inkjet printers

by Charles Briscoe-Smith and Robert Krawitz.

B.1 Introduction

The Epson Stylus Color/Photo printers don't have memory to print using all of the nozzles in the print head. For example, the Stylus Photo 700/EX has 32 nozzles. At 720 dpi, with an 8" wide image, a single line requires $(8 * 720 * 6/8)$ bytes, or 4320 bytes (because the Stylus Photo printers have 6 ink colors). To use 32 nozzles per color would require 138240 bytes. It's actually worse than that, though, because the nozzles are spaced 8 rows apart. Therefore, in order to store enough data to permit sending the page as a simple raster, the printer would require enough memory to store 256 rows, or 1105920 bytes. Considering that the Photo EX can print 11" wide, we're looking at more like 1.5 MB. In fact, these printers are capable of 1440 dpi horizontal resolution. This would require 3 MB. The printers actually have 64K-256K.

With the newer (740/750 and later) printers it's even worse, since these printers support multiple dot sizes; of course, the even newer 2880x720 printers don't help either.

Older Epson printers had a mode called *MicroWeave* (tm). In this mode, the host fed the printer individual rows of dots, and the printer bundled them up and sent them to the print head in the correct order to achieve high quality. This MicroWeave mode still works in new printers, but in some cases the implementation is very minimal: the printer uses exactly one nozzle of each color (the first one). This makes printing extremely slow (more than 30 minutes for one 8.5x11" page), although the quality is extremely high with no visible banding whatsoever. It's not good for the print head, though, since no ink is flowing through the other nozzles. This leads to drying of ink and possible permanent damage to the print head.

By the way, although the Epson manual says that microweave mode should be used at 720 dpi, 360 dpi continues to work in much the same way. At 360 dpi, data is fed to the printer one row at a time on all Epson printers. The pattern that the printer uses to print is very prone to banding. However, 360 dpi is inherently a low quality mode; if you're using it, presumably you don't much care about quality. It is possible to do microweave at 360 DPI, with significantly improved quality.

Except for the Stylus Pro printers (5000, 5500, 7000, 7500, 9000, 9500, and when it's released the 10000), which can do microweave at any resolution, printers from roughly the Stylus Color 600 and later do not have the capability to do MicroWeave correctly in many cases (some printers can do MicroWeave correctly at 720 DPI). Instead, the host must arrange the output in the order that it will be sent to the print head. This is a very complex process; the jets in the print head are spaced more than one row (1/720") apart, so we can't simply send consecutive rows of dots to the printer. Instead, we have to pass e. g. the first, ninth, 17th, 25th... rows in order for them to print in the correct position on the paper. This interleaving process is called "soft" weaving.

This decision was probably made to save money on memory in the printer. It certainly makes the driver code far more complicated than it would be if the printer could arrange the output. Is that a bad thing? Usually this takes far less CPU time than the dithering process, and it does allow us more control over the printing process, e.g. to reduce banding. Conceivably, we could even use this ability to map out bad jets.

Interestingly, apparently the Windows (and presumably Macintosh) drivers for most or all Epson printers still list a "microweave" mode. Experiments have demonstrated that this does not in fact use the "microweave" mode of the printer. Possibly it does nothing, or it uses a different weave pattern from what the non-"microweave" mode does. This is unnecessarily confusing, at least for people who write drivers who try to explain them to people who don't.

What makes this interesting is that there are many different ways of of accomplishing this goal. The naive way would be to divide the image up into groups of 256 rows (for a printer with 32 jets and a separation of 8 rows), and print all the mod8=0 rows in the first pass, mod8=1 rows in the second, and so forth. The problem with this approach is that the individual ink jets are not perfectly uniform; some emit slightly bigger or smaller drops than others. Since each group of 8 adjacent rows is printed with the same nozzle, that means that there will be distinct streaks of lighter and darker bands within the image (8 rows is 1/90", which is visible; 1/720" is not). Possibly worse is that these patterns will repeat every 256 rows. This creates banding patterns that are about 1/3" wide.

So we have to do something to break up this patterning.

Epson does not publish the weaving algorithms that they use in their bundled drivers. Indeed, their developer web site (http://www.ercipd.com/isv/edr_docs.htm) does not even describe how to do this weaving at all; it says that the only way to achieve 720 dpi is to use MicroWeave. It does note (correctly) that 1440 dpi horizontal can only be achieved by the driver (i. e. in software). The manual actually makes it fairly clear how to do this (it requires two passes with horizontal head movement between passes), and it is presumably possible to do this with MicroWeave.

The information about how to do this is apparently available under non-disclosure agreement (NDA). It's actually easy enough to reverse engineer what's inside a print file with a simple Perl script, which is supplied with the Gimp-Print distribution as tests/parse-escp2. In any event, we weren't particularly interested in the weaving patterns Epson used. There are many factors that go into choosing a good weaving pattern; we're learning them as we go along. Issues such as drying time (giving the ink a few seconds more or less to dry can have highly visible effects) affect the quality of the output.

The Uniprint GhostScript driver has been able to do weaving for a long time. It uses patterns that must be specified for each choice of resolution and printer. We preferred an algorithmic approach that computes a weave pattern for any given choice of inputs. This obviously requires extensive testing; we developed a test suite specifically for this purpose.

B.2 Weaving algorithms

I considered a few algorithms to perform the weave. The first one I devised let me use only ($jets - distance\_between\_jets{+}1$) nozzles, or 25. This is OK in principle, but it's slower than using all nozzles. By playing around with it some more, I came up with an algorithm that lets me use all of the nozzles, except near the top and bottom of the page.

This still produces some banding, though. Even better quality can be achieved by using multiple nozzles on the same line. How do we do this? In 1440x720 mode, we're printing two output lines at the same vertical position. However, if we want four passes, we have to effectively print each line twice. Actually doing this would increase the density, so what we do is print half the dots on each pass. This produces near-perfect output, and it's far faster than using (pseudo) "MicroWeave".

Yet another complication is how to get near the top and bottom of the page. This algorithm lets us print to within one head width of the top of the page, and a bit more than one head width from the bottom. That leaves a lot of blank space. Doing the weave properly outside of this region is increasingly difficult as we get closer to the edge of the paper; in the interior region, any nozzle can print any line, but near the top and bottom edges, only some nozzles can print. We originally handled this by using the naive way mentioned above near the borders, and switching over to the high quality method in the interior. Unfortunately, this meant that the quality is quite visibly degraded near the top and bottom of the page. We have since devised better algorithms that allow printing to the extreme top and bottom of the region that can physically be printed, with only minimal loss of quality.

Epson does not advertise that the printers can print at the very top of the page, although in practice most of them can. The quality is degraded to some degree, and we have observed that in some cases not all of the dots get printed. Epson may have decided that the degradation in quality is sufficient that printing in that region should not be allowed. That is a valid decision, although we have taken another approach.

B.2.1 Simple weaving algorithms

The initial problem is to calculate the starting position of each pass; the row number of the printer's top jet when printing that pass. Since we assume the paper cannot be reverse-fed, the print head must, for each pass, start either further down the page than the previous pass or at the same position. Each pass's start point is therefore at a non-negative offset from the previous pass's start point.

Once we have a formula for the starting row of each pass, we then turn that "inside out" to get a formula for the pass number containing each row.

First, let's define how our printer works. We measure vertical position on the paper in "rows"; the resolution with which the printer can position the paper vertically. The print head contains J ink jets, which are spaced S rows apart.

Consider a very simple case: we want to print a page as quickly as possible, and we mostly don't care how sparse the printing is, so long as it's fairly even.

It's pretty obvious how to do this. We make one pass with the print head, printing J lines of data, each line S rows after the previous one. We then advance the paper by $S * J$ rows and print the next row. For example, if $J = 7$ and $S = 4$, this method can be illustrated like this:

```
pass number
| row number------->
| |         111111111122222222223333333333444444444455555555556666666666
| 0123456789012345678901234567890123456789012345678901234567890123456789
0 *---*---*---*---*---*---*
1                             *---*---*---*---*---*---*
2 \-----------------------/                                 *---*---*---*---*---*-
```

```
        7 jets              \---/
                            4 rows offset from one jet to the next
\---------------------------/
   7*4=28 rows offset from one pass to the next
```

In these examples, the vertical axis can be thought of as the time axis, with the pass number shown at the left margin, while the row number runs horizontally. A '`*`' shows each row printed by a pass, and a row of '`-`' is used to link together the rows printed by one pass of the print head. The first pass is numbered '`0`' and starts at row 0. Each subsequent pass p starts at row $p * S * J$. Each pass prints J lines, each line being S rows after the previous one. (For ease of viewing this file on a standard terminal, I'm clipping the examples at column 80.)

This method covers the whole page with lines printed evenly S rows apart. However, we want to fill in all the other rows with printing to get a full-density page (we're ignoring oversampling at this stage). Where we have previously printed a single pass, we'll now print a "pass block": we print extra passes to fill in the empty rows. A naive implementation might look like this:

```
0  *---*---*---*---*---*---*
1   *---*---*---*---*---*---*
2    *---*---*---*---*---*---*
3     *---*---*---*---*---*---*
4                              *---*---*---*---*---*---*
5                               *---*---*---*---*---*---*
6                                *---*---*---*---*---*---*
7                                 *---*---*---*---*---*---*
8                                                          *---*---*---*---*---*-
9                                                           *---*---*---*---*---*
10                                                           *---*---*---*---*---
11                                                            *---*---*---*---*--
```

(Now you can see why this process is called "weaving"!)

B.2.2 Perfect weaving

This simple weave pattern prints every row, but will give conspicuous banding patterns for the reasons discussed above.

Let's start improving this for our simple case. We can reduce banding by making sure that any given jet never prints a row too close to another row printed by the same jet. This means we want to space the rows printed by a given jet evenly down the page. In turn, this implies we want to advance the paper by as nearly an equal amount after each pass as possible.

Each pass block prints $S * J$ lines in S passes. The first line printed in each pass block is $S * J$ rows lower on the page than the first line printed in the previous pass block. Therefore, if we advance the paper by J rows between each pass, we can print the right number of passes in each block and advance the paper perfectly evenly.

Here's what this "perfect" weave looks like:

```
                         start of full weave
                         |
```

```
0 *---*---*---*---*---*---*
1        *---*---*---*---*---*---*
2               *---*---*---*---*---*---*
3                      *---*---*---*---*---*---*
4                             *---*---*---*---*---*---*
5                                    *---*---*---*---*---*---*
6                                           *---*---*---*---*---*---*
7                                                  *---*---*---*---*---*---*
8                                                         *---*---*---*---*---*-
9                                                                *---*---*---*--
10                                                                      *---*---
11                                                                             *
```

You'll notice that, for the first few rows, this weave is too sparse. It is not until the row marked "start of full weave" that every subsequent row is printed. We can calculate this start position as follows:

$$start \;=\; (S-1) \;*\; (J-1)$$

For the moment, we will ignore this problem with the weave. We'll consider later how to fill in the missing rows.

Let's look at a few more examples of perfect weaves:

$S = 2$, $J = 7$, $start = (2-1)*(7-1) = 6$:

```
        starting row of full weave
        |
0 *-*-*-*-*-*-*
1        *-*-*-*-*-*-*
2               *-*-*-*-*-*-*
3                      *-*-*-*-*-*-*
4                             *-*-*-*-*-*-*
5                                    *-*-*-*-*-*-*
6                                           *-*-*-*-*-*-*
7                                                  *-*-*-*-*-*-*
```

$S = 7$, $J = 2$, $start = 6$:

```
        start
        |
0 *-------*
1   *-------*
2     *-------*
3       *-------*
4         *-------*
5           *-------*
6             *-------*
7               *-------*
8                 *-------*
9                   *-------*
```

$S = 4$, $J = 13$, $start = 36$:

```
                                      start
                                      |
0 *---*---*---*---*---*---*---*---*---*---*---*---*
```

```
1              *---*---*---*---*---*---*---*---*---*---*---*---*
2                           *---*---*---*---*---*---*---*---*---*---*---*---*
3                                        *---*---*---*---*---*---*---*---*---*--
4                                                     *---*---*---*---*---*---*-
5                                                                  *---*---*---*
```

$S = 13,\ J = 4,\ start = 36$:

```
                                      start
                                      |
0 *------------*------------*------------*
1     *------------*------------*------------*
2         *------------*------------*------------*
3             *------------*------------*------------*
4                 *------------*------------*------------*
5                     *------------*------------*------------*
6                         *------------*------------*------------*
7                             *------------*------------*------------*
8                                 *------------*------------*------------*
9                                     *------------*------------*------------*
10                                        *------------*------------*-----------
11                                            *------------*------------*-------
12                                                *------------*------------*---
13                                                    *------------*------------
14                                                        *------------*--------
15                                                            *------------*----
16                                                                *------------*
17                                                                    *---------
18                                                                        *-----
19                                                                            *-
```

$S = 8,\ J = 5,\ start = 28$:

```
                              start
                              |
0 *-------*-------*-------*-------*
1      *-------*-------*-------*-------*
2           *-------*-------*-------*-------*
3                *-------*-------*-------*-------*
4                     *-------*-------*-------*-------*
5                          *-------*-------*-------*-------*
6                               *-------*-------*-------*-------*
7                                    *-------*-------*-------*-------*
8                                         *-------*-------*-------*-------*
9                                              *-------*-------*-------*-------*
10                                                  *-------*-------*-------*---
11                                                       *-------*-------*------
12                                                            *-------*-------*-
13                                                                 *-------*----
14                                                                      *-------
15                                                                           *--
```

$S = 9,\ J = 5,\ start = 32$:

```
                                  start
                                  |
0 *--------*--------*--------*--------*
1      *--------*--------*--------*--------*
2           *--------*--------*--------*--------*
3                *--------*--------*--------*--------*
4                     *--------*--------*--------*--------*
5                          *--------*--------*--------*--------*
6                               *--------*--------*--------*--------*
7                                    *--------*--------*--------*--------*
8                                         *--------*--------*--------*--------*
9                                              *--------*--------*--------*-----
10                                                  *--------*--------*--------*
11                                                       *--------*--------*----
12                                                            *--------*--------
13                                                                 *--------*---
14                                                                      *-------
15                                                                           *--
```

$S = 6$, $J = 7$, $start = 30$:

```
                                start
                                |
0 *-----*-----*-----*-----*-----*
1        *-----*-----*-----*-----*-----*
2               *-----*-----*-----*-----*-----*
3                      *-----*-----*-----*-----*-----*
4                             *-----*-----*-----*-----*-----*
5                                    *-----*-----*-----*-----*-----*
6                                           *-----*-----*-----*-----*-----*-----
7                                                  *-----*-----*-----*-----*----
8                                                         *-----*-----*-----*---
9                                                                *-----*-----*--
10                                                                      *-----*-
11                                                                             *
```

B.2.3 Weaving collisions

This perfect weave is not possible in all cases. Let's look at another example:

$S = 6$, $J = 4$:

```
0 *-----*-----*-----*
1     *-----*-----*-----*
2         *-----*-----*-----*
3             *-----*-----*-----*
4             ^   *-^---*-----*-----*
5             |   ^ | *-^---*-----*-----*
              OUCH!   ^ |   ^
                      |     |
```

Here we have a collision. Some lines printed in later passes overprint lines printed by earlier passes. We can see why by considering which row number is printed by a given jet number j (numbered from 0) of a given pass, p:

$$row(p,\ j) \ = \ p * J \ + \ j * S$$

Because $J = 4$ and $S = 6$ have a common factor of 2, jet 2 of pass 0 prints the same row as jet 0 of pass 3:

$$\begin{aligned} row(0,\ 2) &= 0 * 4 \ + \ 2 * 6 \ = \ 12 \\ row(3,\ 0) &= 3 * 4 \ + \ 0 * 6 \ = \ 12 \end{aligned}$$

In fact, with this particular weave pattern, jets 0 and 1 of pass p+3 always overprint jets 2 and 3 of pass p. We'll represent overprinting rows by a '^' in our diagrams, and correct rows by '*':

$S = 6$ $J = 4$:

```
0 *-----*-----*-----*
1          *-----*-----*-----*
2                *-----*-----*-----*
3                     ^-----^-----*-----*
4                           ^-----^-----*-----*
5                                 ^-----^-----*-----*
```

B.2.4 What makes a "perfect" weave?

So what causes the perfect weave cases to be perfect, and the other cases not to be? In all the perfect cases above, S and J are relatively prime (i.e. their greatest common divisor (GCD) is 1). As we mentioned above, $S = 6$ and $J = 4$ have a common factor, which causes the overprinting. Where S and J have a GCD of 1, they have no common factor other than 1 and, as a result, no overprinting occurs. If S and J are not relatively prime, their common factor will cause overprinting.

We can work out the greatest common divisor of a pair of natural numbers using Euler's algorithm:

Start with the two numbers: (e.g.) 9, 24

Swap them if necessary so that the larger one comes first: 24, 9

Subtract the second number from the first: 15, 9

Repeat until the first number becomes smaller: 6, 9

Swap the numbers again, so the larger one comes first: 9, 6

Subtract again: 3, 6

Swap: 6, 3

Subtract: 3, 3

And again: 0, 3

When one of the numbers becomes 0, the other number is the GCD of the two numbers you started with.

These repeated subtractions can be done with C's '%' operator, so we can write this in C as follows:

```
unsigned int
gcd(unsigned int x, unsigned int y)
{
    if (y == 0)
        return x;
    while (x != 0) {
        if (y > x)
            swap (&x, &y);
        x %= y;
    }
    return y;
}
```

'`gcd(S,J)`' will feature quite prominently in our weaving algorithm.

If $0< = j<J$, there should only be a single pair (p, j) for any given row number. If S and J are not relatively prime, this assumption breaks down. (For conciseness, let $G = \mathrm{GCD}(S, J)$.)

$S = 8$, $J = 6$, $G = 2$:

```
0 *-------*-------*-------*-------*-------*
1       *-------*-------*-------*-------*-------*
2             *-------*-------*-------*-------*-------*
3                   *-------*-------*-------*-------*-------*
4                         ^-------^-------^-------*-------*-------*
5                               ^-------^-------^-------*-------*-------*
```

In this case, jets 0, 1 and 2 of pass $p+4$ collide with jets 3, 4 and 5 of pass p.

How can we calculate these numbers? Suppose we were to print using fewer jets, say J/G jets. The greatest common divisor of J/G and S is 1, enabling a perfect weave. But to get a perfect weave, we also have to advance the paper by a factor of G less:

```
0 *-------*-------*       -       -       -
1    *-------*-------*       -       -       -
2       *-------*-------*       -       -       -
3          *-------*-------*       -       -       -
4             *-------*-------*       -       -       -
5                *-------*-------*       -       -       -
```

If we left the paper advance alone, we'd get a sparse weave; only one row can be printed every G rows:

```
0 *-------*-------*       -       -       -
1       *-------*-------*       -       -       -
2             *-------*-------*       -       -       -
3                   *-------*-------*       -       -       -
4                         *-------*-------*       -       -       -
5                               *-------*-------*       -       -       -
               ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

              These rows need filling in.
```

The rows that would have been printed by the jets we've now omitted (shown as '`-`') are printed by other jets on later passes.

Let's analyse this. Consider how a pass p could collide with pass 0. Pass p starts at offset $p * J$. Pass 0 prints at rows which are multiples of S. If $p * J$ is exactly divisible by S, a collision has occurred, unless $p * J >= J * S$ (which will happen when we finish a pass block).

So, we want to find p and q such that $p * J = q * S$ and p is minimised. Then p is the number of rows before a collision, and q is the number of jets in pass 0 which are not involved in the collision. To do this, we find the lowest common multiple of J and S, which is $L = J * S/G$. L/J is the number of rows before a collision, and L/S is the number of jets in the first pass not involved in the collision.

Thus, we see that the first J/G rows printed by a given pass are not overprinted by any later pass. However, the rest of the rows printed by pass p are overprinted by the first $J - (J/G)$ jets of pass $p+(S/G)$. We will use C to refer to S/G, the number of rows after which a collision occurs.

Another example:

$S = 6$, $J = 9$, $G = 3$, $C = S/G = 2$:

```
0 *-----*-----*-----*-----*-----*-----*-----*-----*
1          *-----*-----*-----*-----*-----*-----*-----*-----*
2                   ^-----^-----^-----^-----^-----^-----*-----*-----*
3                            ^-----^-----^-----^-----^-----^-----*-----*-----*
4                                     ^-----^-----^-----^-----^-----^-----*-----
5                                              ^-----^-----^-----^-----^-----^--
         ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^ ^^

              These rows need filling in.
```

In this case, the first $J - (J/G) = 9 - 9/3 = 6$ jets of pass $p+(6/3) = p+2$ collide with the last 6 jets of pass p. Only one row in every $G = 2$ rows is printed by this weave.

$S = 9$, $J = 6$, $G = 3$, $C = 3$:

```
0 *--------*--------*--------*--------*--------*
1       *--------*--------*--------*--------*--------*
2             *--------*--------*--------*--------*--------*
3                   ^--------^--------^--------^--------*--------*
4                         ^--------^--------^--------^--------*--------*
5                               ^--------^--------^--------^--------*--------*
```

Here, the first $J - (J/G) = 6 - 6/3 = 4$ jets of pass $p+(9/3) = p+3$ collide with the last 4 jets of pass p.

Note that, in these overprinting cases, only rows divisible by G are ever printed. The other rows, those not divisible by G, are not touched by this weave.

We can modify our weave pattern to avoid overprinting any rows and simultaneously fill in the missing rows. Instead of using J alone to determine the start of each pass from the previous pass, we adjust the starting position of some passes. As mentioned before, we will divide the page into pass blocks, with S passes in each block. This ensures that the first jet of the first pass in a block prints the row which the Jth jet of the first pass of the previous block would have printed, if the print head had one extra jet.

Looking back at an example of a perfect weave, we can divide it into pass blocks:

$S = 7$, $J = 2$, $G = 1$:

```
                imaginary extra jet
                |
0 *------*      *     <--start of pass block 0
1   *------*    |
2     *------*  |
3       *------*|
4         *-----|*
5           *---|--*
6             *-|----*
                |
7               *------*  <--start of pass block 1
8                 *------*
9                   *------*
```

We can now calculate the start of a given pass by reference to its pass block. The first pass of pass block b always starts at row $(b * S * J)$. The start row of each of the other passes in the block are calculated using offsets from this row.

For the example above, there are 7 passes in each pass block, and their offsets are 0, 2, 4, 6, 8, 10 and 12. The next pass block is offset $S * J = 14$ rows from the start of the current pass block.

The simplest way to modify the “perfect” weave pattern to give a correct weave in cases where $G! = 1$ is to simply change any offsets which would result in a collision, until the collision disappears. Every printed row in the weave, as we have shown it up to now, is separated from each of its neighbouring printed rows by G blank rows. We will add an extra offset to each colliding pass in such a way that we push the pass onto these otherwise blank rows.

We have seen that, unless $G = 1$, the plain weave pattern results in each pass colliding with the pass S/G passes before. We will now subdivide our pass block into subblocks, each consisting of $B = S/G$ passes. There are therefore G subblocks in a pass block.

For each subblock, the passes in that subblock have a constant offset added to them. The offset is different for each subblock in a block. There are many ways we can choose the offsets, but the simplest is to make the offset equal to the subblock number (starting from 0).

Thus, the passes in the first subblock in each pass block remain at the offsets we've already calculated from J. The passes in the second subblock each have 1 added to their offset, the passes in the third subblock have 2 added, and so on. Thus, the offset of pass p (numbered relative to the start of its pass block) is $p * J + \text{floor}(p/B)$.

This gives us a weave pattern looking like this:

$S = 6$, $J = 9$, $G = 3$, $B = 2$:

```
0 *-----*-----*-----*-----*-----*-----*-----*-----*
1 ^        *-----*-----*-----*-----*-----*-----*-----*-----*
2 |              +-> *-----*-----*-----*-----*-----*-----*-----*-----*
3 |              |            *-----*-----*-----*-----*-----*-----*-----*-----*
4 |              |                  +-> *-----*-----*-----*-----*-----*-----*---
5 |              |                  |            *-----*-----*-----*-----*-----*
6 |              |                  |               +-> *-----*-----*-----*-----
7 |              |                  |               |            *-----*-----*--
```

```
  |              |                  |             start of pass block 1
  |              |                  |             (offset returns to 0)
  |              |                  start of subblock 2 (offset 2 rows)
  |              start of subblock 1 (following passes offset by 1 row)
  start of passblock 0, subblock 0 (pass start calculated as p*J)
```

$S = 9$, $J = 6$, $G = 3$, $B = 3$:

```
0 *--------*--------*--------*--------*--------*
1       *--------*--------*--------*--------*--------*
2             *--------*--------*--------*--------*--------*
3                    *--------*--------*--------*--------*--------*
4                          *--------*--------*--------*--------*--------*
5                                *--------*--------*--------*--------*--------*
6                                       *--------*--------*--------*--------*---
7                                             *--------*--------*--------*------
8                                                   *--------*--------*--------*
9                                                       *--------*--------*-----
10                                                  \---/     *--------*--------
11                                               small offset       *--------*--
12                                                                         *----
```

This method of choosing offsets for subblocks can result in an occasional small offset (as shown above) between one pass and the next, particularly when G is large compared to J. For example:

$S = 8$, $J = 4$, $G = 4$, $B = 2$:

```
0 *-------*-------*-------*
1     *-------*-------*-------*
2          *-------*-------*-------*
3              *-------*-------*-------*
4                   *-------*-------*-------*
5                       *-------*-------*-------*
6                            *-------*-------*-------*
7                                *-------*-------*-------*
8                                 *-------*-------*-------*
9                                \/   *-------*-------*-------*
                              very small offset!
```

We can plot the offset against the subblock number as follows:

```
subblock number
| offset
| |
| 0123
0 *
1  *
2   *
3    *
0 *
1  *
2   *
3    *
```

The discontinuity in this plot results in the small offset between passes.

As we said at the beginning, we want the offsets from each pass to the next to be as similar as possible. We can fix this by calculating the offset for a given subblock b as follows:

```
offset(b) = 2*b             , if b < ceiling(G/2)
          = 2*(G-b)-1       , otherwise
```

We can visualise this as follows, for $G = 10$:

```
  0123456789
0 *
1   *
2     *
3       *
4         *
5          *
6        *
7      *
8    *
9  *
0 *
1   *
2     *
3       *
4         *
5          *
6        *
7      *
8    *
9  *
```

and for $G = 11$:

```
             1
   01234567890
 0 *
 1   *
 2     *
 3       *
 4         *
 5           *
 6          *
 7        *
 8      *
 9    *
10  *
 0 *
 1   *
 2     *
 3       *
 4         *
 5           *
 6          *
```

```
 7        *
 8      *
 9    *
10  *
```

This gives a weave looking like this:

$S = 12$, $J = 6$, $G = 6$, $B = 2$:

```
0 *-----------*-----------*-----------*-----------*-----------*
1       *-----------*-----------*-----------*-----------*-----------*
2               *-----------*-----------*-----------*-----------*-----------*
3                     *-----------*-----------*-----------*-----------*---------
4                             *-----------*-----------*-----------*-----------*-
5                                   *-----------*-----------*-----------*-------
6                                          *-----------*-----------*-----------*
7                                                *-----------*-----------*------
8                                                    *-----------*-----------*--
9                                                          *-----------*--------
10                                                             *-----------*----
11                                                                   *----------
12                                                                       *-----
```

This method ensures that the offset between passes is always in the range $[J-2, J+2]$.

(This might seem odd, but it occurs to me that a good weave pattern might also make a good score for bell ringers. When church bells are rung, a list of "changes" are used. For example, if 8 bells are being used, they will, at first, be rung in order: 12345678. If the first change is for bells 5 and 6, the bells will then be rung in the order 12346578. If the second change is 1 and 2, the next notes are 21346578. After a long list of changes, the order the bells are rung in can become quite complex.

For a group of bell-ringers to change the order of the notes, they must each either delay their bell's next ring, hasten it, or keep it the same as the time it takes to ring all the bells once. The length of time between each ring of a given bell can only be changed a little each time, though; with an ink-jet weave pattern, we want the same to apply to the distance between passes.)

Finally, knowing the number of jets J and their separation S, we can calculate the starting row of any given pass p as follows:

```
passesperblock = S
passblock = floor(p / passesperblock)
offsetinpassblock = p - passblock * passesperblock
subblocksperblock = gcd(S, J)
passespersubblock = S / subblocksperblock
subpassblock = floor(offsetinpassblock / passespersubblock)
if subpassblock < ceiling(subblocksperblock/2)
    subblockoffset = 2*subpassblock
else
    subblockoffset = 2*(subblocksperblock-subpassblock)-1
startingrow = passblock * S * J + offsetinpassblock * J + subblockoffset
```

We can simplify this down to the following:

```
subblocksperblock = gcd(S, J)
```

```
subpassblock = floor((p % S) * subblocksperblock / S)
if subpassblock * 2 < subblocksperblock
    subblockoffset = 2*subpassblock
else
    subblockoffset = 2*(subblocksperblock-subpassblock)-1
startingrow = p * J + subblockoffset
```

So the row number of jet j of pass p is

```
subblocksperblock = gcd(S, J)

subblockoffset(p)
    = 2*subpassblock       , if subpassblock * 2 < subblocksperblock
    = 2*(subblocksperblock-subpassblock)-1      , otherwise
      where
      subpassblock = floor((p % S) * subblocksperblock / S)

row(j, p) = p * J + subblockoffset(p) + j * S
```

Together with the inequality $0<=j<J$, we can use this definition in reverse to calculate the pass number containing a given row, r. Working out the inverse definition involves a little guesswork, but one possible result is as follows. Given a row, r, which is known to be the first row of a pass, we can calculate the pass number as follows:

```
subblocksperblock = gcd(S, J)
subblockoffset = r % subblocksperblock
pass = (r - subblockoffset) / J
```

If $G == 1$, we can determine the pass number with this algorithm:

```
offset = r % J
pass = (r - offset) / J
while (offset % S != 0)
{
  pass--
  offset += J
}
jet = offset / S
```

Generalising, we come up with this algorithm. Given r, S and J:

```
G = gcd(S, J)
passespersubblock = S/G
subblockoffset = r % G
subpassblock = subblockoffset / 2   , if subblockoffset % 2 == 0
             = G - (subblockoffset+1)/2    , otherwise
baserow = r - subblockoffset - (subpassblock * passespersubblock * J)
offset = baserow % J
pass = (baserow - offset) / J
while (offset % S != 0)
{
  offset += J
  pass -= 1
}
subblockretreat = floor(pass / passespersubblock) % G
```

```
pass -= subblockretreat * passespersubblock
pass += subpassblock * passespersubblock
jet = (r - subblockoffset - pass * J) / S
```

Let's look at some examples of imperfect but correct weave patterns:

$S = 6$, $J = 4$, GCD = 2,
passesperblock=S=6,
passespersubblock=S/G=6/2=3:

```
0 *-----*-----*-----*
1     *-----*-----*-----*
2         *-----*-----*-----*
3              *-----*-----*-----*
4                  *-----*-----*-----*
5                      *-----*-----*-----*
6                         *-----*-----*-----*
7                             *-----*-----*-----*
8                                 *-----*-----*-----*
9                                      *-----*-----*-----*
10                                         *-----*-----*-----*
11                                             *-----*-----*-----*
12                                                *-----*-----*-----*
13                                                    *-----*-----*-----*
14                                                        *-----*-----*-----*
15                                                             *-----*-----*----
16                                                                 *-----*-----*
17                                                                     *-----*--
18                                                                        *-----
19                                                                            *-
```

$S = 8$, $J = 6$, $G = 2$,
passesperblock=S=8,
passespersubblock=S/G=8/2=4:

```
0 *-------*-------*-------*-------*-------*
1       *-------*-------*-------*-------*-------*
2             *-------*-------*-------*-------*-------*
3                   *-------*-------*-------*-------*-------*
4                          *-------*-------*-------*-------*-------*
5                                *-------*-------*-------*-------*-------*
6                                      *-------*-------*-------*-------*-------*
7                                            *-------*-------*-------*-------*--
8                                                 *-------*-------*-------*-----
9                                                       *-------*-------*-------
10                                                            *-------*-------*-
11                                                                  *-------*---
12                                                                         *----
```

$S = 6$, $J = 12$, $G = 6$,
passesperblock=S=6,
passespersubblock=S/G=6/6=1:

```
0 *-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
```

```
1               *-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*---
2                             *-----*-----*-----*-----*-----*-----*-----*-----*-
3                                          *-----*-----*-----*-----*-----*-----*
4                                                    *-----*-----*-----*-----*--
5                                                              *-----*-----*----
6                                                                         *-----
```

We have now solved the basic weaving problem. There are two further refinements we need to consider: oversampling, and filling in the missing rows at the start of the weave.

B.2.5 Oversampling

By oversampling, we mean printing on the same row more than once. There are two reasons for oversampling: to increase the horizontal resolution of the printout and to reduce banding.

Oversampling to increase horizontal resolution is necessary because, although the printer might be able to position an ink drop to, for example, 1/1440" horizontally, it may not be able to lay down two such drops 1/1440" apart. If it can print two drops 1/720" apart, 2x oversampling will be necessary to get a 1/1440" horizontal resolution. If it can only print two drops 1/360" apart, 4x oversampling will be necessary for a 1/1440" horizontal resolution. The printer enforces this “drop spacing” by only accepting raster passes with a horizontal resolution matching the spacing with which it can print dots, so we must print passes at different horizontal positions if we are to obtain a higher horizontal resolution. (Another reason it does this may be to reduce the amount of memory needed in the printer.)

Oversampling can also be done to decrease the banding apparent in an image. By splitting a row into two or more sets of dots (“lines”) and printing each line on the same row, but with a different nozzle for each line, we can get a smoother print.

To quantify these two kinds of oversampling, we'll introduce two new constants: H shows how many different horizontal offsets we want to print at (the “horizontal oversampling”) while O shows how many times we want to print each row, over and above the number of times necessary for horizontal oversampling (the “extra oversampling”).

It is necessary for all the lines printed by a given pass to have the same horizontal offset, but there need not be any relation between them in terms of extra oversampling. For the moment, however, we will treat all oversampling as potentially requiring this alignment; all lines in one pass must be derived from the original row data in the same way. Thus, we'll assume $O = 1$ for now.

So, how do we do this oversampling? In fact, it can be done easily: advance the paper by a factor of H less between each pass. We'll define a new variable, A, to show how much we advance the paper between passes. Previously, we'd have defined $A = J$; we now let $A = J/H$. This also affects our pass blocks. Printing one pass block used to involve advancing the paper $S * J$ rows; it now advances the paper $S * J/H$ rows. We therefore name a group of H pass blocks a “band”. Printing one band involves advancing the paper $S * J$ rows, as a pass block did before.

To keep our weave pattern working correctly, so that overprinting does not occur within a pass block, we also have to redefine G as $\mathrm{GCD}(S, A)$. Here's an example of an oversampled weave pattern:

$S = 4$, $J = 10$, $H = 2$, $A = J/H = 10/2 = 5$, $G = \mathrm{GCD}(4, 5) = 1$,
passesperblock=S=4,
passespersubblock=S/G=4/1=4:

```
0 *---*---*---*---*---*---*---*---*---*
1      *---*---*---*---*---*---*---*---*---*
2           *---*---*---*---*---*---*---*---*---*
3                *---*---*---*---*---*---*---*---*---*
4                     *---*---*---*---*---*---*---*---*---*
5                          *---*---*---*---*---*---*---*---*---*
6                               *---*---*---*---*---*---*---*---*---*
7                                    *---*---*---*---*---*---*---*---*---*
8                                         *---*---*---*---*---*---*---*---*---*
9                                              *---*---*---*---*---*---*---*---*
10                                                  *---*---*---*---*---*---*---
11                                                       *---*---*---*---*---*--
12                                                            *---*---*---*---*-
13                                                                 *---*---*---*
14                                                                      *---*---
15                                                                           *--
```

Now we have to determine which line is printed by each jet on each pass. If we number each line generated as we split up a row, we can use these numbers. We'll number the lines in our diagram by replacing the '*'s with integers in the range $[0 \ldots H - 1]$.

Overprinting occurs once per pass block, so we can simply print pass block 0 with line 0, pass block 1 with line 1, pass block 2 with line 2, etc, wrapping to 0 when we've run out of lines:

```
0 0---0---0---0---0---0---0---0---0---0
1      0---0---0---0---0---0---0---0---0---0
2           0---0---0---0---0---0---0---0---0---0
3                0---0---0---0---0---0---0---0---0---0
4                     1---1---1---1---1---1---1---1---1---1
5                          1---1---1---1---1---1---1---1---1---1
6                               1---1---1---1---1---1---1---1---1---1
7                                    1---1---1---1---1---1---1---1---1---1
8                                         0---0---0---0---0---0---0---0---0---0
9                                              0---0---0---0---0---0---0---0---0
10                                                  0---0---0---0---0---0---0---
11                                                       0---0---0---0---0---0--
12                                                            1---1---1---1---1-
13                                                                 1---1---1---1
14                                                                      1---1---
15                                                                           1--
```

$S = 4$, $J = 12$, $H = 2$, $A = J/H = 12/2 = 6$, $G = \mathrm{GCD}(4, 6) = 2$,
passesperblock=S=4,
passespersubblock=S/G=4/2=2:

```
0 0---0---0---0---0---0---0---0---0---0---0---0
1       0---0---0---0---0---0---0---0---0---0---0---0
2              0---0---0---0---0---0---0---0---0---0---0---0
```

```
3                    0---0---0---0---0---0---0---0---0---0---0---0
4                         1---1---1---1---1---1---1---1---1---1---1---1
5                               1---1---1---1---1---1---1---1---1---1---1---1
6                                      1---1---1---1---1---1---1---1---1---1---1
7                                            1---1---1---1---1---1---1---1---1--
8                                                 0---0---0---0---0---0---0---0-
9                                                       0---0---0---0---0---0---
10                                                             0---0---0---0---0
11                                                                   0---0---0--
12                                                                        1---1-
```

But what do we do if J is not an exact multiple of H? This is a difficult problem, which I struggled with for quite a few days before giving in and taking the easy (but less elegant) way out. The easy solution is to round J/H down, then add on the accumulated error at the end of each band.

$S = 4$, $J = 11$, $H = 2$ $A = \text{floor}(J/H) = \text{floor}(11/2) = 5$, $G = \text{GCD}(4, 5)$,
passesperblock=S=4,
passespersubblock=S/G=4/1=4

```
Band 0:
0 0---0---0---0---0---0---0---0---0---0---0
1      0---0---0---0---0---0---0---0---0---0---0
2           0---0---0---0---0---0---0---0---0---0---0
3                0---0---0---0---0---0---0---0---0---0---0
4                     1---1---1---1---1---1---1---1---1---1---1
5                          1---1---1---1---1---1---1---1---1---1---1
6                               1---1---1---1---1---1---1---1---1---1---1
7                                    1---1---1---1---1---1---1---1---1---1---

Band 1:
8 |                                           0---0---0---0---0---0---0---0---0-
9  \-----------------------------------------/     0---0---0---0---0---0---0---0
10                   S*J rows                           0---0---0---0---0---0---
11                                                           0---0---0---0---0--
12                                                                1---1---1---1-
13                                                                     1---1---1
14                                                                          1---
```

We can calculate the starting row and subpass number of a given pass in this scheme as follows:

```
A = floor(J / H)
subblocksperblock = gcd(S, A)
subpassblock = floor((p % S) * subblocksperblock / S)
if subpassblock * 2 < subblocksperblock
    subblockoffset = 2*subpassblock
else
    subblockoffset = 2*(subblocksperblock-subpassblock)-1
band = floor(P / (S * H))
passinband = P % (S * H)
startingrow = band * S * J + passinband * A + subblockoffset
subpass = passinband / S
```

So the row number of jet j of pass p is

```
A = floor(J / H)
subblocksperblock = gcd(S, A)

subblockoffset(p)
    = 2*subpassblock       , if subpassblock * 2 < subblocksperblock
    = 2*(subblocksperblock-subpassblock)-1      , otherwise
      where
      subpassblock = floor((p % S) * subblocksperblock / S)

band(p) = floor(p / (S * H))
passinband(p) = p % (S * H)

row(j, p) = band(p) * S * J + passinband(p) * A + subblockoffset(p) + j * S
row(j, p) = p * J + subblockoffset(p) + j * S
```

To be continued. . . .

Appendix C ESC/P2

C.1 Introduction to ESC/P2

This is a description of the ESC/P2 raster commands used by the gimp-print plugin and Ghostscript driver, which is a subset of the complete command set. The full documents are found on `http://www.ercipd.com/isv/edr_docs.htm`. Note that these are **not** always correct, and are certainly not complete.

All ESCP/2 raster commands begin with the '`ESC`' character (1b hex), followed by either one or two command characters and arguments where applicable. Older commands generally have one command character. Newer commands usually have a '`(`' (left parenthesis) followed by a command character and a byte count for the arguments that follow. The byte count is a 16-bit (2 byte) binary integer, in little endian order.

All arguments listed here are of the form '`name[bytes]`' where '`[bytes]`' is the number of bytes that comprise the argument. The arguments themselves are usually one, two, or four byte integers, always little endian (the least significant bits come first). Presumably this is to match Intel processors.

In some cases, the same command sequence identifies different versions of the same command, depending upon the number of bytes of arguments.

C.2 Standard commands

`ESC @` ESC/P2 command

Reset the printer. Discards any output, ejects the existing page, returns all settings to their default. Always use this before printing a page.

`ESC (G` *BC*`=1` *ON1* ESC/P2 command

Turn on graphics mode. *ON* should be '`1`' (turn on graphics mode).

`ESC (U` *BC*`=1` *UNIT1* ESC/P2 command

Set basic unit of measurement used by printer. This is expressed in multiples of 1/3600". At 720 DPI, *UNIT* is '`5`'; at 360 DPI, *UNIT* is '`10`'.

`ESC (U` *BC*`=5` *PAGEUNITS1 VUNIT1 HUNIT1 BASEUNIT2* ESC/P2 command

Set basic units of measurement used by the printer. *PAGEUNIT* is the unit of page measurement (for commands that set page dimensions and the like). *VUNIT* is the unit of vertical measurement (for vertical movement commands). *HUNIT* is the unit of horizontal movement (for horizontal positioning commands). All of these units are expressed in *BASEUNIT*, which is in reciprocal inches. Typically, *BASEUNIT* is '`1440`'. In 720 DPI mode, *PAGEUNIT*, *VUNIT*, and *HUNIT* are all '`2`'; in 1440x720 DPI mode, *PAGEUNIT* and *VUNIT* are normally set to '`2`'; *HUNIT* is set to '`1`'.

`ESC (K` *BC*`=2` *ZERO1 GRAYMODE1* ESC/P2 command

Set color or grayscale mode, on printers supporting an explicit grayscale mode. These printers can be identified because they are advertised as having more black nozzles than nozzles of individual colors. Setting grayscale mode allows use of these extra nozzles for faster printing. *GRAYMODE* should be '`0`' or '`2`' for color, '`1`' for grayscale. *ZERO* should always be '`0`'.

`ESC (i` *BC*=`1` *MICROWEAVE1* ESC/P2 command

If *MICROWEAVE* is '`1`', use microweave mode. On older printers, this is used to turn on microweave; on newer printers, it prints one row at a time. All printers support this mode. It should only be used at 720 (or 1440x720) DPI. The Epson Stylus Pro series indicates additional modes:

'`2`' "Full-overlap"

'`3`' "Four-pass"

'`4`' "Full-overlap 2"

Any of these commands can be used with the high four bits set to either '`3`' or '`0`'.

`ESC U` *DIRECTION1* ESC/P2 command

If *DIRECTION* is '`1`', print unidirectionally; if '`0`', print bidirectionally.

`ESC (s` *BC*=`1` *SPEED1* ESC/P2 command

On some older printers, this controls the print head speed. *SPEED* of '2' is 10 inches/sec; SPEED of '`0`' or 1 is 20.

`ESC (e` *BC*=`2` *ZERO1 DOTSIZE1* ESC/P2 command

Choose print dotsize. *DOTSIZE* can take on various values, depending upon the printer. Almost all printers support '`0`' and '`2`'. Variable dot size printers allow a value of *16*. Other than the value of *16*, this appears to be ignored at resolutions of 720 DPI and above.

`ESC (C` *BC*=`2` *PAGELENGTH2* ESC/P2 command
`ESC (C` *BC*=`4` *PAGELENGTH4* ESC/P2 command

Set the length of the page in "pageunits" (see '`ESC (U`' above). The second form of the command allows setting of longer page lengths on new printers (these happen to be the printers that support variable dot size).

`ESC (c` *BC*=`4` *TOP2 LENGTH2* ESC/P2 command
`ESC (c` *BC*=`8` *TOP4 LENGTH4* ESC/P2 command

Set the vertical page margins of the page in "pageunits" (see '`ESC (U`' above). The margins are specified as the top of the page and the length of the page. The second form of the command allows setting of longer page lengths on new printers (these happen to be the printers that support variable dot size).

`ESC (S` *BC*=`8` *WIDTH4 LENGTH4* ESC/P2 command

Set the width and length of the printed page region in "pageunits" (see '`ESC (U`' above).

`ESC (v` *BC*=2 *ADVANCE2* — ESC/P2 command
`ESC (v` *BC*=4 *ADVANCE4* — ESC/P2 command

Feed vertically *ADVANCE* "vertical units" (see '`ESC (U`' above) from the current print head position.

`ESC (V` *BC*=2 *ADVANCE2* — ESC/P2 command
`ESC (V` *BC*=4 *ADVANCE4* — ESC/P2 command

Feed vertically *ADVANCE* "vertical units" (see '`ESC (U`' above) from the top margin.

`ESC ($` *BC*=4 *OFFSET4* — ESC/P2 command

Set horizontal position to *OFFSET* from the left margin. This command operates on printers of the 740 class and newer (all printers with variable dot size).

`ESC $` *OFFSET2* — ESC/P2 command

Set horizontal position to *OFFSET* from the left margin. This command operates on printers of the 740 class and newer (all printers with variable dot size).

`ESC (\` *BC*=4 *UNITS2 OFFSET2* — ESC/P2 command

Set horizontal position to *OFFSET* from the previous print head position, measured in *UNITS*. *UNITS* is measured in inverse inches, and should be set to 1440 in all cases. This operates on all 1440 dpi printers that do not support variable dot size.

`ESC (/` *BC*=4 *OFFSET4* — ESC/P2 command

Set horizontal position to *OFFSET* from the previous print head position, measured in "horizontal units" (see '`ESC (U`' above). This operates on all variable dot size printers.

`ESC \` *OFFSET2* — ESC/P2 command

Set horizontal position to *OFFSET* from the previous print head position, measured in basic unit of measurement (see '`ESC (U`' above). This is used on all 720 dpi printers, and can also be used on 1440 dpi printers in lower resolutions to save a few bytes. Note that *OFFSET* may be negative. The range of values for this command is between '`-16384`' and '`16383`'.

`ESC r` *COLOR1* — ESC/P2 command
`ESC (r` *BC*=2 *DENSITY1 COLOR1* — ESC/P2 command

Set the ink color. The first form is used on four-color printers; the second on six-color printers. *DENSITY* is '`0`' for dark inks, '`1`' for light. *COLOR* is

'`0`' black

'`1`' magenta

'`2`' cyan

'`4`' yellow

This command is not used on variable dot size printers in softweave mode.

`ESC . ` *COMPRESS1 VSEP1 HSEP1 LINES1 WIDTH2 data...* ESC/P2 command

Print data. *COMPRESS* signifies the compression mode:

'`0`' no compression

'`1`' TIFF compression (incorrectly documented as "run length encoded")

'`2`' TIFF compression with a special command set.

VSEP depends upon resolution and printer type. At 360 DPI, it is always '`10`'. At 720 DPI, it is normally '`5`'5. On the ESC 600, it is '`40`' (8∗5). On some other printers, it varies.

HSEP1 is '`10`' at 360 DPI and '`5`' at 720 or 1440 DPI (1440 DPI cannot be printed in one pass; it is printed in two passes, with the dots separated in each pass by 1/720").

LINES is the number of lines to be printed. It should be '`1`' in microweave and 360 DPI. At 720 DPI softweave, it should be the number of lines to be actually printed.

WIDTH is the number of pixels to be printed in each row. Following this command, a carriage return ('`13`' decimal, '`0A`' hex) should be output to return the print head position to the left margin.

The basic data format is a string of bytes, with data from left to right on the page. Within each byte, the highest bit is first.

The TIFF compression is implemented as one count byte followed by one or more data bytes. There are two cases:

1. If the count byte is '`128`' or less, it is followed by [*count*]+1 data bytes. So if the count byte is '`0`', it is followed by 1 data byte; if it is '`128`', it is followed by 129 data bytes.
2. If the count byte is greater than 128, it is followed by one byte. This byte is repeated (257 - [count]) times. So if [count] is 129, the next byte is treated as though it were repeated 128 times; if [count] is 255, it is treated as though it were repeated twice.

`ESC i` *COLOR1 COMPRESS1 BITS1 BYTES2 LINES2 data...* ESC/P2 command

Print data in the newer printers (that support variable dot size), and Stylus Pro models.

COLOR is the color:

'`0`' black

'`1`' magenta

'`2`' cyan

'`4`' yellow

'`17`' light magenta

'`18`' light cyan

COMPRESS signifies the compression mode:

'`0`' no compression

`1` TIFF compression (incorrectly documented as "run length encoded")

`2` TIFF compression with a special command set.

BITS is the number of bits per pixel.

BYTES is the number of bytes wide for each row ($ceiling(BITS * width\_of\_row, 8)$). Note that this is different from the `ESC .` command above.

LINES is the number of lines to be printed. This command is the only way to get variable dot size printing. In variable dot mode, the size of the dots increases as the value (`1`, `2`, or `3`) increases.

`ESC (D` *BC*=4 *BASE2 VERTICAL1 HORIZONTAL1* — ESC/P2 command

Set printer horizontal and vertical spacing. It only applies to variable dot size printers in softweave mode (and possibly other high end printers).

BASE is the base unit for this command; it must be `14400`.

VERTICAL is the distance in these units between printed rows; it should be ($separation\_in\_nozzles * BASE/720$).

HORIZONTAL is the horizontal separation between dots in a row. Depending upon the printer, this should be either (14400/720) or (14400/360). The Stylus Pro 9000 manual suggests that the settings should match the chosen resolution, but that is apparently not the case (or not always the case) on other printers.

`ESC (R` *BC*=8 00 `R E M O T E 1` — ESC/P2 command

Enters "remote mode". This is a special, undocumented command set that is used to set up various printer options, such as paper feed tray, and perform utility functions such as head cleaning and alignment. It does not appear that anything here is actually required to make the printer print. Our best understanding of what is in a remote command sequence is described in a separate section below.

`ESC 01 @EJL [sp] ID\r\n` — ESC/P2 command

Return the printer ID. This is considered a remote mode command, although the syntax is that of a conventional command. This returns the following information:

```
@EJL ID\r
MFG:EPSON;
CMD:ESCPL2,BDC;
MDL:[printer model];
CLS:PRINTER;
DES:EPSON [printer model];
\f
```

After all data has been sent, a form feed byte should be sent.

All newer Epson printers (STC 440, STP 750) require the following command to be sent at least once to enable printing at all. This command specifically takes the printer out of the Epson packet mode communication protocol (whatever that is) and enables normal data transfer. Sending it multiple times is is not harmful, so it is normally sent at the beginning of each job:

```
ESC 01@EJL[space]1284.4[newline]@EJL[space][space][space][space]
[space][newline]ESC@
```

The proper sequence of initialization commands is:

```
magic command
ESC  remote mode if needed
ESC (G
ESC (U
ESC (K (if appropriate)
ESC (i
ESC U (if needed)
ESC (s (if appropriate)
ESC (e
ESC (C
ESC (c
ESC (S
ESC (D (if needed)
ESC (V (optional -- this can be accomplished with ESC (v)
```

For printing, the proper sequence is:

```
ESC (v
```

and repeat for each color:

`ESC ($` or `ESC (\` or `ESC \`
`ESC (r` or `ESC r` (if needed—not used with '`ESC i`' and not needed if the color has not changed from the previous printed line)
`ESC .` or `ESC i  ...data... [return] (0A hex)`

To terminate a page:

```
[formfeed] (0C hex)
ESC @
```

C.3 Remote Mode Commands

The following description of remote commands comes out of an examination of the sequences used by the printer utilities bundled with the Windows drivers for the ESC740, and from other sources (some Epson manuals, experimentation, analysis of print files). It is largely speculative as these commands are not all documented in the Epson documentation we have access to. Generally, newer manuals provide more thorough documentation.

Remote command mode is entered when the printer is sent the following sequence:

```
ESC (R BC=8 00 R E M O T E 1
```

Remote mode commands are then sent, and terminated with the following sequence:

```
ESC 00 00 00
```

All remote mode commands must be sent before the initial '`ESC (G`' command is sent.

This introductory sequence is then followed by a sequence of commands. Each command is constructed as follows:

Two ASCII bytes indicating the function

A byte count (two bytes, little-endian) for the parameters

Binary parameters, if any

This is a list of all remote commands we have seen:

`NC` *BC*`=2 00 00` — ESC/P2 remote command
: Print a nozzle check pattern

`VI` *BC*`=2 00 00` — ESC/P2 remote command
: on my 740, prints the following:

```
W01286 I02382\r\n
```

probably "version information"

`* AI` *BC*`=3 00 00 00` — ESC/P2 remote command
: Prints a "printer ID". On one 870, prints the following:

```
51-51-50-51-49-48\r\n
```

The Windows driver has a text entry field where this number can be entered, but its purpose is unknown.

`* LD` *BC*`=0` — ESC/P2 remote command
: Load printer defaults from NVRAM, DIP switches, and/or ROM. This apparently does not load factory defaults per se, but any settings that are saved. This is commonly used right at the end of each print job after the '`ESC @`' printer reset command.

`* CH` *BC*`=2 00` *xx* — ESC/P2 remote command
: Perform a head cleaning cycle. The heads to clean are determined by parameter *xx*:

'`00`' clean all heads

'`01`' clean black head

'`02`' clean color heads

While *xx* = '`00`' is probably supported by all printers, *xx* = '`01`' and '`02`' may well not be.

`* DT` *BC*`=3 00` *xx* `00` — ESC/P2 remote command
: Print an alignment pattern. There are three patterns, which are picked via the choice of *xx*. Pattern '`0`' is coarse, pattern '`1`' is medium, and pattern '`2`' is fine.

`* DU` *BC*`=6 00` *xx* `00 09 00` *yy* — ESC/P2 remote command
: Print another alignment pattern. It isn't entirely clear what *xx* and *yy* are, but it appears that *xx* takes on the values '`1`' and '`2`' , and *yy* takes on the values '`0`' and *1*. This may only work on the 580 and/or 480.

`* DA` *BC*`=4 00` *xx* `00` *yy* — ESC/P2 remote command
: Set results for the first alignment pattern. *xx* is the pattern ('`1`'–'`3`'); *yy* is the best choice from the set ('`1`'–'`7`' or '`1`'–'`15`'). This does not save to NVRAM, so when the printer is powered off, the setting will be lost.

`* DA` *BC*=`6 00` *xx* `00` *yy* `09 00` ESC/P2 remote command

Set results for alternate alignment pattern. *xx* appears to be the pass, and *yy* appears to be the optimum pattern. This does not save to NVRAM, so when the printer is powered off, the setting will be lost.

`* SV` *BC*=`0` ESC/P2 remote command

Save the current settings to NVRAM.

`* RS` *BC*=`1 01` ESC/P2 remote command

Reset the printer.

`* IQ` *BC*=`1 01` ESC/P2 remote command

Get ink quantity. This requires direct access to the printer port. The return looks like

```
IQ:KKCCMMYY
```

or

```
IQ:KKCCMMYYccmm
```

(for 4-color and 6-color printers respectively), where each pair of digits are hexadecimal representations of percent.

The following two commands have been observed on an STP 870.

`* IR` *BC*=`2 00` *xx* ESC/P2 remote command

Function unknown. This command has been observed on an STP 870 with *xx*='`03`' at the start of a job and *xx*='`02`' at the end of a job (where it is followed by an '`LD`' command). When in roll mode, the values change to *xx*='`01`' at the start of a job and *xx*='`00`' at the end of a job.

`* FP` *BC*=`3 00` *xx yy* ESC/P2 remote command

xx='`00`' and *yy*='`00`' selects the printer's normal left margin (about 3mm). *xx*='`0xb0`' and *yy*='`0xff`' selects zero-margin mode, where the left-most print position is shifted to a point about 0.1" to the left of the left-hand edge of the paper, allowing printing up to (and off) the left-hand edge of the paper.

The commands below are partially documented in the Stylus Pro 9000 manual. Much of this information is interpreted; none is tested.

`* SN` *BC*=`3 00` *xx yy* ESC/P2 remote command

Select Mechanism Sequence. *xx* controls which sub-operation is performed. *xx*='`00`' selects the "Feed paper sequence setting". *yy* can take on the following values (on the STP 870, at any rate):

'`0`' default

'`1`' plain paper

'`2`' postcards

‘3’ film (photo quality glossy film, transparencies)

‘4’ envelopes

‘5’ plain paper (fast load)

‘6’ back light film (although this has been observed with heavyweight matte paper)

‘7’ matte paper (observed with 360 dpi inkjet paper, and photo quality inkjet paper)

‘8’ photo paper

Experimentation suggests that this setting changes details of how the printers’ cut sheet feeder works, presumably to tune it for different types of paper.

xx=‘01’ controls the platen gap setting; *yy*=‘00’ is the default, *yy*=‘1’ or ‘2’ are higher settings.

xx=‘02’ controls paper loading speed (*yy*=‘0’ is normal, ‘1’ is fast, ‘2’ is slow). It appears that ‘1’ is used when printing on “plain paper”, “360dpi ink jet paper” or “ink jet transparencies”, and *yy*=‘00’ for all other paper type settings.

xx=‘07’ controls duplex printing for printers with that capability (*yy=0* is default, for non-duplex printing; ‘1’ is front side of the paper, and *2* is back side).

xx=‘09’ controls zero margin printing on the printers with the capability of printing zero-margin on all sides (Stylus Photo 780/790, 890, and 1280/1290). *yy*=‘0’ is the default; ‘1’ enables zero margin printing.

* PP *BC*=3 00 *xx yy* ESC/P2 remote command
  Set Paper Path. *xx*=‘2’ indicates manual feed, *xx*=‘3’ is for roll paper. *yy* selects “paper path number”.

* AC *BC*=2 00 *xx* ESC/P2 remote command
  Set Auto Cutting State. *xx*=‘0’ selects auto cutting off, *xx*=‘1’ selects auto cutting on, and *xx*=‘2’ indicates horizontal print page line on. It appears that with auto cutting on, roll paper is cut automatically at the point a formfeed character is sent. The formfeed character is normally used to eject a page; with this turned on, it also cuts the roll paper. Horizontal print page line on prints a narrow line of black dots at the position the paper should be cut manually.

* DR *BC*=4 00 xx *DT2* ESC/P2 remote command
  Set Drying Time. *xx*=‘00’ sets the drying time “per scan” (per pass?); *xx*=‘01’ sets the drying time per page. *DT* indicates the drying time, which is in seconds if page mode is used and in milliseconds if scan mode is used. *DT* must not exceed 3600 seconds in per-page mode and 10000 milliseconds in per-scan mode.

* IK *BC*=2 00 *xx* ESC/P2 remote command
  Select Ink Type. *xx*=‘00’ selects dye ink. Pigment ink is apparently selected by *xx*=‘01’. This probably does not apply to the consumer-grade printers.

* `PZ BC=2 00 xx` ESC/P2 remote command
  Set Pause After Printing. *xx*=‘`00`’ selects no pause after printing; *xx*=‘`01`’ selects pause after printing. If turned on, the printer is paused after the page is ejected (by the FF byte). If cutting is turned on, the printer is paused **after** the cutting or printing of the horizontal cut line.

* `EX` *BC*`=6 00 00 00 00 0x14` *xx* ESC/P2 remote command
  Set Vertical Print Page Line Mode. *xx*=‘`00`’ is off, *xx*=‘`01`’ is on. If turned on, this prints vertical trim lines at the left and right margins.

* `EX` *BC*`=6 00 00 00 00 0x05` *xx* ESC/P2 remote command
  Set Roll Paper Mode. If *xx* is ‘`0`’, roll paper mode is off; if *xx* is ‘`1`’, roll paper mode is on.

* `EX` *BC*`=3 00` *xx yy* ESC/P2 remote command
  Appears to be a synonym for the ‘`SN`’ command described above.

* `PH` *BC*`=2 00` *xx* ESC/P2 remote command
  Select Paper Thickness. Set the paper thickness *xx* in .1 mm units. This must not exceed 0x10 (1.6 mm). If the thickness is set "more than" .6 mm (which probably means "at least" .6 mm, since the other case reads "less than 0.5 mm"), the platen gap is set high irrespective of the ‘`SN`’ command.

* `PM` *BC*`=2 00 00` ESC/P2 remote command
  Function unknown. Used on the STC 3000 at least when using roll feed, and on the STP 870 in all print files analysed to date.

* `ST` *BC*`=2 00` *xx* ESC/P2 remote command
  Epson's STP 750/1200 programming guide refers to the ‘`ST`’ command as “Set printer state reply”. If *xx* is ‘`0`’ or ‘`2`’, the printer will not send status replies. If *xx* is ‘`1`’ or ‘`3`’, the printer will send status replies. The status replies consist of state, error codes, ink leve, firmware version, and warning status.

  The actual reply is documented as

  ```
  @BDC ST\r
  ST: xx;
  [ER: yy;]
  IQ: n1n2n3n4;
  [WR: w1,w2...;]
  RV: zz;
  AI:CW:02kkccmmyy, MI:mm
  [TC:tttt;]
  INK:...;
  \f
  ```

  (‘`\r`’ is carriage return; ‘`\n`’ is newline; ‘`\f`’ is formfeed.)

  ‘`ST`’ is the printer status:

  ‘`00`’ Error

'01'	Self-test
'02'	Busy
'03'	Waiting while printing
'04'	Idle
'07'	Cleaning/filling ink heads
'08'	Not yet initialized/filling heads

'ER', if provided, is the error status:

'00'	Fatal error
'01'	Interface not selected
'04'	Paper jam
'05'	Out of ink
'06'	Paper out

'IQ' is the amount of ink left, as a (decimal!) percentage expressed in hexadecimal. The values are black, cyan, magenta, and yellow (presumably 6-color printers supply light cyan and light magenta inks as additional parameters).

'WR', if provided, is the warning status:

'10'	Black ink low
'11'	Cyan
'12'	Magenta
'13'	Yellow
'14'	Light cyan (presumably)
'15'	Light magenta (presumably)

'RV' is the firmware revision (one byte ASCII).

'AI' is actuator information. These are two byte ASCII codes that indicate "ink weight rank ID" of KCMY, respectively.

'TC', if provided, is the total time of cleaning or ink filling (?).

'INK:' and 'MI' are **not documented**.

* SM *BC*=2 00 *xx* ESC/P2 remote command

Set Status Reply Rate. *xx* is the repeat interval in seconds. If *xx* is '0', the status is returned only when the printer's state changes.

* ST *BC*=1 01 ESC/P2 remote command

Reply Printer Status. The reply is formatted as

    @BDC PS\r\nST:xx;\f

('\r' is carriage return; '\n' is newline; '\f' is formfeed). If *xx* (the reply value) is '0' or '2', automatic status update is disabled; if '1' or '3', it is enabled.

`* SM` *BC*`=1 01` ESC/P2 remote command

Reply Printer Status Rate. The reply is formatted as

```
@BDC PS\r\nST:xx;\f
```

('`\r`' is carriage return; '`\n`' is newline; '`\f`' is formfeed). See '`SM` *BC*`=2`' above for the meaning of the return value.

`* ??` *BC*`=xx` *y*`[1] ...` *y*`[xx]` ESC/P2 remote command

Echo Parameters (perhaps better described as Echo Commands). The command string is executed (it would appear from the documentation), and the string sent is returned using a sequence similar to that described in the '`ST` *BC*`=1`' and '`SM` *BC*`=1`' commands. Note that in this case the number of bytes is variable!

`* SM` *BC*`=2 00 02` ESC/P2 remote command

Function unknown. Used on the STC 3000 at least when using roll feed.

C.4 Appropriate Remote Commands

All of the remote commands described above are wrapped up with the usual boilerplate. The files always start with '`00 00 00`' and the "magic" command described above, then two '`ESC @`'s to reset the printer. The remote command sequences come next; if they print anything that is usually followed by a '`FF`' (0C hex) character to feed the page, then the file ends with another two '`ESC @`'s to get back to the ground state.

An alignment sequence goes like this:

1. Host uses '`DT`' to print an alignment sheet
2. User eyeballs the sheet to see which is the best aligned pattern.
3. Host sends a '`DA`' command indicating which pattern the user chose
4. If the user said "realign", meaning he isn't done yet, go to step 1
5. We are done: host sends a '`SV`' command and exits

The sequence used (by the STC 3000, at least) to print from the roll feed is (with byte count omitted)

```
PM 00 00
SN 00 00 00
EX 00 00 00 00 05 01
ST 00 01
SM 00 02
```

The sequence used by the STP 870 to print on plain paper is

```
PM 00 00
IR 00 03
SN 00 00 01
SN 00 01 00
SN 00 02 01
EX 00 00 00 00 05 00
FP 00 00 00
```

and the job finishes with

```
IR 00 02
LD
```

For different paper type settings on the STP 870, the arguments to '`SN`' vary. The arguments to the first and third '`SN`' commands are as outlined in the description of the '`SN`' command above; the arguments to the second ("platen gap") are '`00 01 01`' for thick papers ("matte paper—heavyweight", "photo paper" and "premium glossy photo paper") and '`00 01 00`' for all others.

For roll-mode printing, the STP 870's sequence changes as follows. '`IR`''s arguments become '`00 01`' in the header, and '`00 00`' after the job, and '`EX`''s last argument changes from '`00`' to '`01`'.

For zero-margin printing on the STP 870, the arguments to '`FP`' become '`00 0xb0 0xff`'. This moves the origin about 5.5mm to the left, to a point one tenth of an inch to the left of the left-hand edge of the paper, allowing printing right up to (and beyond) the edge of the paper. Some printers (at least the STP 870) include white absorbent pads at the left margin position and other positions (89mm and 100mm on the STP 870) to soak up ink which misses the edge of the paper. Printing off the edge of paper of a width not aligned with a pad could result in making a mess of the inside of the printer and ink getting on the reverse of the paper.

Appendix D Adding a new printer

This appendix covers adding a new ESCP/2, PCL, or Canon printer. Writing a new driver module is not covered.

The three steps to adding a printer are:

1. Add an entry to 'printers.xml'
2. Add the appropriate code and data to the appropriate driver module
3. Tune the printer

Printer information is stored in two places: in 'printers.xml' (which contains the list of printer models available to the the upper-level application), and in the appropriate driver file ('print-escp2.c', 'print-pcl.c', or 'print-canon.c').

D.1 'printers.xml'

'printers.xml' is an XML-like file (there's no formal DTD) that contains very simple printer definitions. A typical definition follows:

```
<printer name="EPSON Stylus Color 1500" driver="escp2-1500">
<color>
<model value=2>
<gamma value=0.597>
<density value=1.0>
<language value=escp2>
</printer>
```

There are other tags that may be present. The only ones that are mandatory are '<printer>', '<color>', '<model>', and '<language>'. The other optional parameters (gamma and density in this case) can be used to adjust control settings. This is probably not the right place for them; the printer drivers themselves should contain this information. There's probably no good reason for anything but gamma and density to be in here. Gamma refers to the printer's gamma factor; density is the desired base ink density for the printer. The Epson driver contains the density information for each printer at each resolution internally. An even better driver would adjust density and possibly even gamma for paper type. All the more reason not to have that information here.

If you really are curious about what tags are permitted, please see 'printdefl.l'. I deliberately want to make this obscure.

Anyway, here's the definition of the tags that do matter:

<printer name="*longname*" driver="*drivername*"> — 'printers.xml' tag

This starts the definition of a printer. The *longname* should be something human readable; the *drivername* should consist of alphanumerics and hyphens, and be fairly short. The *longname* is what will appear in the GUI listing of printers; the *drivername* is what is actually used to key into the list of printers. It is legal to have multiple printers with the same driver name.

<color> — 'printers.xml' tag
<nocolor> — 'printers.xml' tag

Indicates that this printer is capable of color, or is not capable of color respectively

`<model value=int>` 'printers.xml' tag

This defines a model number. This is passed into the driver, which may do whatever it cares to with it—index into a table, compute on, or whatever. This need not be unique.

`<language value=type>` 'printers.xml' tag

This defines what driver module this printer uses. *type* should be '`escp2`', '`pcl`', '`canon`', or '`ps`'.

`</printer>` 'printers.xml' tag

This, of course, closes off a printer definition.

This is handled very ad-hoc. It's ugly. But it's reasonably easy to extend, and it's buzzword-compliant.

D.2 The driver file

Adding a new printer to a driver module ('`print-escp2.c`', '`print-pcl.c`', or '`print-canon.c`'—'`print-ps.c`' is really ad hoc) requires a bit more planning. Each driver is somewhat different, but they all generally have a vector of printer definitions, and the code does some special casing based on particular printer capabilities. The PCL and Canon drivers are quite similar; the Canon driver was actually cribbed from the PCL driver, but it then returned the favor.

The Epson driver is a little bit different. Canon and PCL printers have some amount of intelligence; a lot of them have specific ink options, and know about specific paper sizes and types, and must be told the right thing. Epson printers have somewhat less intelligence and will more or less do exactly what the host tells it to do in a fairly regular fashion. I actually prefer this; it isn't materially more work for the host to compute things like exact paper sizes and such, it allows a lot more tweaking, and it may be why Epson has been more open with information – the communication protocol doesn't really contain very much IP, so they have less reason to keep it secret.

Someone else will have to fill in the sections about PCL and Canon printers.

D.3 Epson inkjet printers

The '`model_capabilities`' vector in '`print-escp2.c`' contains one entry for each defined printer model. The '`model`' parameter in '`printers.xml`' is an index into this table.

In general, the new printers have fewer eccentricities than the older printers. That doesn't mean they're simpler, just that they're more consistent.

An `escp2_printer_t` is a C struct defined as follows:

escp2_printer_t Data type

```
typedef struct escp2_printer
{
  model_cap_t flags; /* Bitmask of flags, see below */
/*****************************************************************************/
```

```
  int nozzles; /* Number of nozzles per color */
  int min_nozzles; /* Minimum number of nozzles per color */
  int nozzle_separation; /* Separation between rows, in 1/360" */
  int black_nozzles; /* Number of black nozzles (may be extra) */
  int min_black_nozzles; /* # of black nozzles (may be extra) */
  int black_nozzle_separation; /* Separation between rows */
/****************************************************************************/
  int xres; /* Normal distance between dots in */
/* softweave mode (inverse inches) */
  int enhanced_xres; /* Distance between dots in highest */
/* quality modes */
  int base_separation; /* Basic unit of row separation */
  int base_resolution; /* Base hardware spacing (above this */
/* always requires multiple passes) */
  int enhanced_resolution;/* Above this we use the */
    /* enhanced_xres rather than xres */
  int resolution_scale;   /* Scaling factor for ESC(D command */
  int max_black_resolution; /* Above this resolution, we */
      /* must use color parameters */
      /* rather than (faster) black */
      /* only parameters*/
  int max_hres;
  int max_vres;
  int min_hres;
  int min_vres;
/****************************************************************************/
  int max_paper_width; /* Maximum paper width, in points */
  int max_paper_height; /* Maximum paper height, in points */
  int min_paper_width; /* Maximum paper width, in points */
  int min_paper_height; /* Maximum paper height, in points */
/* Softweave: */
  int left_margin; /* Left margin, points */
  int right_margin; /* Right margin, points */
  int top_margin; /* Absolute top margin, points */
  int bottom_margin; /* Absolute bottom margin, points */
/* "Micro"weave: */
  int m_left_margin; /* Left margin, points */
  int m_right_margin; /* Right margin, points */
  int m_top_margin; /* Absolute top margin, points */
  int m_bottom_margin; /* Absolute bottom margin, points */
/****************************************************************************/
  int extra_feed; /* Extra distance the paper can be spaced */
/* beyond the bottom margin, in 1/360". */
/* (maximum useful value is */
/* nozzles * nozzle_separation) */
  int separation_rows; /* Some printers require funky spacing */
/* arguments in microweave mode. */
  int pseudo_separation_rows;/* Some printers require funky */
/* spacing arguments in softweave mode */
```

```
  int            zero_margin_offset;   /* Offset to use to achieve */
      /* zero-margin printing */
/***************************************************************************/
 /* The stylus 480 and 580 have an unusual arrangement of
  color jets that need special handling */
  const int *head_offset;
  int initial_vertical_offset;
  int black_initial_vertical_offset;

/***************************************************************************/
  const int *dot_sizes; /* Vector of dot sizes for resolutions */
  const double *densities; /* List of densities for each printer */
  const escp2_variable_inklist_t *inks; /* Choices of inks for this printer */
/***************************************************************************/
  const double *lum_adjustment;
  const double *hue_adjustment;
  const double *sat_adjustment;
  const paperlist_t *paperlist;
} escp2_printer_t;
```

The printer definition block is divided into 8 sections. The first section is a set of miscellaneous printer options. These are described in the code, and will not be discussed further here.

The second section describes the number of nozzles and the separation between nozzles in base units. The base unit is 1/360" for all currently supported printers, but future printers may support a smaller base unit.

Many printers have more black nozzles than nozzles of other colors, and when used in black and white mode, it's possible to use these extra nozzles, which speeds up printing. As an example, a printer that is specified to have 48 cyan, magenta, and yellow nozzles, and 144 black nozzles, can use all 144 black nozzles when printing black ink only. When printing in color, only 48 nozzles of each color (including black) can be used.

Most printers can print using either the number of nozzles available or any smaller number. Some printers require that all of the nozzles be used. Those printers will set `min_nozzles` and/or `min_black_nozzles` to the same value as `nozzles` and/or `black_nozzles`.

The third section defines basic units of measure for the printer, including the standard separation between dots, the base nozzle separation, and the minimum and maximum printing resolutions the printer supports. Most of these are fairly self-explanatory, but some are not obvious.

Most Epson printers, other than the high-end Stylus Pro models, cannot print dots spaced more closely than 1/360" or 1/720" apart (this is the setting for `xres`. This is true even for printers that support resolutions of 1440 or 2880 DPI. In these cases, the data must be printed in 2, 4, or 8 passes. While the printer can position the head to a resolution of 1/1440" or 1/2880", the head cannot deposit ink that frequently.

Some printers can only print in their very best quality (using the smallest dots available) printing at a lower resolution. For example, the Stylus Photo EX can normally print with a dot spacing of 1/720". The smallest dot size cannot be printed with a dot spacing of less than 1/360", however. In this case, we use `enhanced_xres` to specify the resolution to be used in this enhanced mode, and `enhanced_resolution` to specify the printing resolution above which we use the `enhanced_xres`.

The `resolution_scale` command is used to specify scaling factors for the dot separation on newer printers. It should always be 14400 with current printers.

The fourth section specifies the minimum and maximum paper sizes, and the margins. Some printers allow use of narrower margins when softweave is used; both sets of margins are specified.

There is a convenient '`INCH`' macro defined to make specification of the `max_paper_width` and `max_paper_height` more legible. It multiplies 72 by the provided expression to get the appropriate number of points. For example, to specify 8.5", '`INCH(17/2)`' expands to '`(72 * 17/2)`', which is evaluated left to right, and hence generates the correct value.

The fifth section specifies some miscellaneous values that are required for certain printers. For most printers, the correct values are 1 for `separation_rows` and 0 for the others. Very, very few printers require (or allow) `separation_rows` to be anything but 1 and `pseudo_separation_rows` other than zero. The Stylus Color 1520, Stylus Color 800, Stylus Color 850, and (strangely enough to my mind, since it's a new printer) Stylus Color 660 seem to be the only exceptions.

The `zero_margin_offset` is used to specify an additional negative horizontal offset required to print to the edges of the paper on newer Stylus Photo printers. These must be determined empirically; good starting values are 100 for 1440 DPI and 50 for 2880 DPI printers. The goal is to print to the edge of the page, but not over it.

The sixth section specifies head offsets for printers that do not have the color jets aligned. Certain printers, such as the Stylus Color 480, have an unusual head arrangement whereby instead of all of the colors being aligned vertically, the nozzles are configured in groups. These printers are easy to determine; if the normal head offset of zero for each color is used, the printing will be vertically out of alignment. Most of these printers require specification of a negative offset for printing to the top edge of the paper; typically these printers do not require such an offset when printing black only.

The seventh section specifies the most difficult values to tune, the dot sizes, printing densities, and ink values (for variable dot size enabled printers). These will be described in detail below.

The last section specifies luminosity, hue, and saturation adjustment vectors for the printer, and the paper definitions. These are used to adjust the color in Photograph and Solid Colors output modes. These are each vectors of 48 (actually 49, as the first value must be duplicated) doubles that remap the luminosity, hue, and saturation respectively. The hue is calculated, and the value used to interpolate between the two closest points in each vector.

The paper definitions is a set of paper definitions. The paper definition contains the name of the paper type, special settings that are required for printers to process the paper correctly, and a set of adjustment values. These are not currently discussed here.

The lists of dot sizes and densities contain values for 13 printing modes: 120/180 DPI using printer weaving (single row; incorrectly referred to as "microweave") and "soft" weaving (the driver determines the exact pattern of dot layout), 360 DPI microweave and softweave, 720x360 DPI microweave and softweave, 720 DPI microweave and softweave, 1440x720 microweave and softweave, 2880x720 microweave and softweave, and 2880x1440 softweave only. Printer weaving is referred to as "microweave" for historical reasons.

For the dot sizes, the value for each element in the vector selects the dot size to be used when printing at this (or similar) resolution. The dot sizes are determined by consulting the programming manual for the printer and experimenting as described below. Current Epson printers always use dot sizes less than '`16`', or '`0x10`', to indicate single dot size (each dot is represented by 1 bit, and it's either printed or not), and dot sizes of '`16`' or greater to indicate variable dot size (each dot is represented by 2 bits, and it can either be not printed or take on 2 or 3 values, representing the relative size of the printed dot). Variable dot sizes permit the use of very small dots (which would be too small to fill the page and produce solid black) in light areas, while allowing the page to be filled with larger dots in darker areas.

Even single dot size printers can usually produce dots of different sizes; it's just illegal to actually try to switch dot size during a page. These dots are also much bigger than those used in true variable dot size printing.

A dot size of '`-1`' indicates that this resolution is illegal for the printer in question. Any resolutions that would use this dot size will not be presented to the user. A dot size of '`-2`' indicates that this resolution is legal, but that the driver is not to attempt to set any dot size. Some very old printers do not support the command to set the dot size.

Most printers support a dot size of '`0`' as a mode-specific default, but it's often a bigger dot than necessary. Printers usually also support some dot sizes between '`1`' and '`3`'. Usually '`1`' is the right dot size for 720 and 1440 dpi printing, and '`3`' works best at 360 dpi.

Variable dot size printers usually support 2 or 3 sets of variable dot sizes. Older printers based on a 6 picolitre drop (the 480, 720, 740, 750, 900, and 1200) support two: mode 16 (0x10 in hexadecimal) for normal variable dots at 1440 or 720 dpi, and mode 17 (0x10) for special larger dots at 360 dpi. Newer printers based on 4 picolitre drops normally support three sizes: '`0x10`' for 4 pl base drops, '`0x11`' for 6 pl base drops, and '`0x12`' for special large drops. On these printers, '`0x10`' usually works best at 1440x720 and '`0x11`' works best at 720x720. Unfortunately, '`0x10`' doesn't seem to generate quite enough density at 720x720, because if it did the output would be very smooth. Perhaps it's possible to tweak things. . . .

The list of densities is a list of base density values for all of the above listed modes. "Density" refers to the amount of ink deposited when a solid color (or solid black) is printed. So if the density is '`.5`', solid black actually prints only half the possible dots. "Base density" refers to the fact that the density value can be scaled in the GUI or on the Ghostscript command line. The density value specified (which is not made visible to the user) is multiplied by the base density to obtain the effective density value. All other things (such as ink drop size) remaining the same, doubling the resolution requires halving the base density. The base density in the density vector may exceed '`1`', as many paper types require lower density than the base driver. The driver ensures that the actual density never exceeds 1.

Tuning the density should be done on high quality paper (usually glossy photo paper). The goal is to find the lowest density value that results in solid black (no visible gaps under a fairly high power magnifying glass or loupe). If an appropriate density value is found for 720 DPI, it could be divided by 2 for 1440x720, by 4 for 2880x720, and by 8 for 2880x1440.

However, for printers that offer a choice of dot size, this may not be the best strategy. The best choice for dot size is the smallest dot size that allows choosing a density value not greater than 1 that gives full coverage. This dot size may be different for different resolutions. Tuning variable dot size printers is more complicated; the process is described below.

The last member is a pointer to a structure containing a list of ink values for variable dot size (or 6 color) inks. We model variable dot size inks as producing a certain "value" of ink for each available dot size, where the largest dot size has a value of 1. 6-color inks are handled similarly; the light cyan and light magenta inks are treated as a fractional ink value. The combination of variable dot size and 6 color inks, of course, just creates that many more different ink choices.

This structure is actually rather complicated; it contains entries for each combination of physical printer resolution (180, 360, 720, and 1440 dpi), ink colors (4, 6, and 7), and single and variable dot sizes (since some printer modes can't handle variable dot size inks). Since there's so much data, it's actually a somewhat deeply nested structure:

An `escp2_printer_t` contains a pointer (essentially, a reference rather than a copy) to an `escp2_variable_inklist_t`.

An `escp2_variable_inklist_t` contains pointers to `escp2_variable_inkset_t` structures. There is one such pointer for each combination of resolution, dot type, and ink colors as described above. Yes, this is rather inflexible.

An `escp2_variable_inkset_t` contains pointers to `escp2_variable_ink_t` structures. There is one such pointer for each of the four colors (C, M, Y, and K).

An `escp2_variable_ink_t` contains a pointer to the actual list of ink values (`simple_dither_range_t`), the number of ink values, and a density value to be used for computing the transitions. This density value is actually a scaling value; it is multiplied by the effective density to compute the density to be used for computing the transitions. Normally, this value is '`1`', but in some cases it may be possible to get smoother results with a different value (in particular, the single dot size 6-color inks work best with the effective density scaled to '`.75`' for this purpose). A lower density lowers the transition points, which results in more ink being deposited.

A `simple_dither_range_t` is a structure containing four values:

1. The value of the particular ink
2. The bit pattern used to represent the ink
3. Whether the ink is light (0) or dark (1), for inks with light and dark variants
4. The relative amount of ink actually deposited by this dot (not currently used for much; it can be used for ink reduction purposes, to reduce the amount of ink deposited on the paper).

These things are interesting as arrays. From an array of `simple_dither_range_t`'s, the dither code computes transition values that it looks up at run time to decide what ink to print, as well as whether to print at all.

Really confused now? Yup. You'll probably find it easier to simply read the code.

D.4 Tuning the printer

Now, how do you use all this to tune a printer? There are a number of ways to do it; this one is my personal favorite.

There's a file named '`cyan-sweep.tif`'. This consists of a thin bar of cyan sweeping from white to almost pure cyan, and from pure cyan to black. The first thing to do is to pick the appropriate `simple_dither_range_t` (or create a whole new `escp2_variable_inklist_t`) and comment out all but the darkest ink (this means you'll be using the largest dots of dark ink). At 8.5" width (the width of a letter-size piece of paper), the bar will be 1/8" high. Printing it on wider or narrower paper will change the height accordingly. Print it width-wise across a piece of photo quality paper in line art mode using ordered or adaptive hybrid dither. Do not use photographic mode; the colors in photographic mode vary non-linearly depending upon the presence of the three color components, while in line art mode the colors are much purer. Make sure that all the color adjustments are set to defaults (1.0). Use the highest quality version of the print mode you're testing to reduce banding and other artifacts. This is much easier to do with the Gimp than with Ghostscript.

At this stage, you want to look for four things:

1. The black near the center of the line is solid, but not more so than that.
2. The cyan immediately to the left of the black is *almost* solid.
3. The dark cyan at the far right of the page is solid, but not more so. You can try tuning the density so that it isn't quite solid, then nudging up the density until it is.
4. Both sweeps sweep smoothly from light to dark. In particular, the dark half of the bar shouldn't visibly change color; it should go smoothly from cyan to black.

Repeat this stage until you have everything just right. Use the positioning entry boxes in the dialog to position each bar exactly 1/8" further down the page. Adjacent bars will be touching.

The next step is to uncomment out the second darkest dot size. If you're using variable dots, use the second largest dot size of the dark ink rather than the largest dot size of the light ink. This will give you two inks.

When you recompile the plugin, you simply need to copy the new executable into the correct place. You do not need to exit and restart the Gimp.

Print another bar adjacent to the first one. Your goal is to match the bar using a single dot size as closely as possible. You'll find that the dark region of the bar shouldn't change to any great degree, but the light half probably will. If the lighter part of the light half is too dark, you need to increase the value of the smaller dot; if it's too light, you need to decrease the value. The reasoning is that if the value is too low, the ink isn't being given enough credit for its contribution to the darkness of the ink, and vice versa. Repeat until you have a good match. Make sure you let the ink dry fully, which will take a few minutes. Wet ink will look too dark. Don't look at the paper too closely; hold it at a distance. The extra graininess of the largest dot size will probably make it look lighter than it should; if you hold it far enough away so that you can't see the dots, you'll get a more accurate picture of what's going on.

After you have what looks like a good match, print another bar using only the largest dot size (or dark ink, for single dot size 6-color printers). You want to ensure that the bars touching each other look identical, or as close as possible to it; your eye won't give you a good reading if the bars are separated from each other. You'll probably have to repeat the procedure.

The next step is to comment out all but the largest and third-largest dot size, and repeat the procedure. When they match, use all three dot sizes of dark ink. Again, the goal is to match the single dot size.

You'll probably find the match is imperfect. Now you have to figure out what region isn't right, which takes some experimentation. Even small adjustments can make a noticeable difference in what you see. At this stage, it's very important to hold the page far enough from your eye; when you use all three dot sizes, the texture will be much more even, which sometimes makes it look darker and sometimes lighter.

After this is calibrated, it's time to calibrate the light ink against the dark ink. To do this, comment out all but the large dot version of the two inks, and repeat the procedure. This is trickier, because the hues of the inks might not be quite identical. Look at the dark half of the bar as well as the light half to see that the hue really doesn't change as you sweep from cyan to black. Sometimes it's easier to judge that way. You may find that it looks blotchy, in which case you should switch from ordered dither to adaptive hybrid.

After you have the light and dark inks calibrated against each other, it's time to add everything back in. Usually you don't want to use the largest dot size of light ink. These dots will be much larger than the small dots of dark ink, but they'll still be lighter. This will cause problems when printing mixed colors, since you'll be depositing more ink on lighter regions of the page, and you'll probably get strange color casts that you can't get rid of in neutral tones. I normally use only the smallest one or two dot sizes of light ink.

After you've tweaked everything, print the color bar with saturation set to zero. This will print neutral tones using color inks. Your goal here is to look for neutral tonality. If you're using a 6-color printer and get a yellow cast, it means that the values for your light inks are too high (remember, that means they're getting too much credit, so you're not depositing enough cyan and magenta ink, and the yellow dominates). If you get a bluish or bluish-purple cast, your light inks are too low (you're not giving them enough credit, so too much cyan and magenta is deposited, which overwhelms the yellow). Make sure you do this on very white, very high grade inkjet paper that's designed for 1440x720 dpi or higher; otherwise the ink will spread on contact and you'll get values that aren't really true for high grade paper. You can, of course, calibrate for low grade paper if that's what you're going to use, but that shouldn't be put into the distribution.

You can also fully desaturate this bar inside the Gimp and print it as monochrome (don't print the cyan as monochrome; the driver does funny things with luminance), for comparison. You'll find it very hard to get rid of all color casts.

There are other ways of tuning printers, but this one works pretty well for me.

D.5 Canon inkjet printers

Basically, a new Canon printer can be added to '`print-canon.c`' in a similar way as described above for the epson inkjet printers. The main differences are noted here:

In general, Canon printers have more "built-in intelligence" than Epson printers which results in the fact that the driver only has to tell the printing conditions like resolutions, dot sizes, etc. to the printer and afterwards transfer the raster data line by line for each color used.

`canon_cap_t` is a C struct defined as follows:

canon_cap_t Data type

```
typedef struct canon_caps {
  int model;          /* model number as used in printers.xml */
  int max_width;      /* maximum printable paper size */
  int max_height;
  int base_res;       /* base resolution - shall be 150 or 180 */
  int max_xdpi;       /* maximum horizontal resolution */
  int max_ydpi;       /* maximum vertical resolution */
  int max_quality;
  int border_left;    /* left margin, points */
  int border_right;   /* right margin, points */
  int border_top;     /* absolute top margin, points */
  int border_bottom;  /* absolute bottom margin, points */
  int inks;           /* installable cartridges (CANON_INK_*) */
  int slots;          /* available paperslots */
  int features;       /* special bjl settings */
  canon_dot_size_t dot_sizes; /* Vector of dot sizes for resolutions */
  canon_densities_t densities; /* List of densities for each printer */
  canon_variable_inklist_t *inxs; /* Choices of inks for this printer */
} canon_cap_t;
```

Since there are Canon printers which print in resolutions of $2^n * 150$ dpi (e.g. 300, 600, 1200) and others which support resolutions of $2^n * 180$ dpi (e.g. 360, 720, 1440), there's a base resolution (150 or 180, respectively) given in the `canon_cap_t`. The structs `canon_dot_size_t`, `canon_densities_t` and `canon_variable_inklist_t` refer to resolutions being multiples of the base resolution.

For the Canon driver, the struct `canon_dot_size_t` holds values for a model's capabilities at a given resolution: '`-1`' if the resolution is not supported. '`0`' if it can be used and '`1`' if the resolution can be used for variable dot size printing.

In `canon_densities_t` the base densities for each resolution can be specified like for an epson printer, the same holds true for `canon_variable_inklist_t`. See the descriptions above to learn about how to adjust your model's output to yield nice results.

There's a slight difference though in the way the Canon driver and the escp2 driver define their variable inklists: In the Canon driver, you need to define an inklist like this:

```
static const canon_variable_inklist_t canon_ink_myinks[] =
{
  {
    1,4, /* 1bit/pixel, 4 colors */
    &ci_CMYK_1, &ci_CMYK_1, &ci_CMYK_1,
    &ci_CMYK_1, &ci_CMYK_1, &ci_CMYK_1,
  },
  {
    3,4, /* 3bit/pixel, 4 colors */
    &ci_CMYK_3, &ci_CMYK_3, &ci_CMYK_3,
```

```
    &ci_CMYK_3, &ci_CMYK_3, &ci_CMYK_3,
  },
};
```

where the '&ci_CMYK_1' and '&ci_CMYK_3' entries are references to a previously defined const of type `canon_variable_inkset_t`.

Data Type and Variable Index

Function and Macro Index

Concept Index