GNU a2ps, version 4.15.6

General Purpose PostScript Generating Utility
Edition 4.15.6, 13 March 2024

Akim Demaille
Miguel Santana

1 Introduction

This document describes GNU `a2ps` version 4.15.6. The latest versions may be found on the home page (`https://www.gnu.org/software/a2ps/`). We plan to update the GNU home page (`https://www.gnu.org/software/a2ps/`) in the near future, in which case the latter will be a better source of information.

We tried to make this document informative and pleasant. It tries to be more than a plain reference guide, and intends to offer information about the concepts or tools etc. that are related to printing PostScript. This is why it is now that big: to offer you all the information you might want, **not** because `a2ps` is difficult to use. See Appendix A [Glossary], page 92, for technical words or even general information.

Please, send us emailcards `:)`. Whatever the comment is, or if you just like `a2ps`, write to Miguel Santana and Akim Demaille. But *never* write to either of us for asking questions, or to report bugs. Chances are very high never to receive an answer, as we receive too many messages. See Section 1.3 [a2ps Mailing Lists], page 2, for information on the mailing lists.

1.1 Description

`a2ps` formats files for printing on a PostScript printer.

The format used is nice and compact: normally two pages on each physical page, borders surrounding pages, headers with useful information (page number, printing date, file name or supplied header), line numbering, pretty-printing, symbol substitution etc. This is very useful for making archive listings of programs or just to check your code in the bus. Actually `a2ps` is kind of bootstrapped: its sources are frequently printed with `a2ps :)`.

While at the origin its names was derived from "ASCII to PostScript", today we like to think of it as "Any to PostScript". Indeed, `a2ps` supports *delegations*, i.e., you can safely use `a2ps` to print DVI, PostScript, LaTeX, JPEG etc., even compressed.

A short list of features of `a2ps` might look like this:

- Customizable through various configuration files (see Chapter 4 [Configuration Files], page 30)
- Powerful escapes to define the headers, table of contents etc. the way you want (see Section 3.2 [Escapes], page 24);
- Variables to push even further the customizability in a comfortable manner (see Section 4.9 [Your Variables], page 33);
- Open approach of encodings (see Chapter 6 [Encodings], page 43);
- Excellent support of the Latin 2, 3, 4, 5 and 6 encodings, thanks to `Ogonkify` (see Section "Overview" in *Ogonkify manual*), written by Juliusz Chroboczek.
- Fully customizable output style: fonts, background and foreground colors, line numbering style etc. (see Section 8.6 [Designing PostScript Prologues], page 80).
- Possibility to delegate the processing of some files to other filters (see Section 4.10 [Your Delegations], page 35).
- Many contributions, e.g., pretty-print diffs, print reference cards of programs, sanitize broken PostScript files, print Duplex on Simplex printers etc. (see Chapter 9 [Contributions], page 84).

– And finally, the ability to pretty-print sources written in quite a few various languages (see Chapter 7 [Pretty Printing], page 48).

1.2 Reporting Bugs

We try hard to make `a2ps` portable on any Unix platform, and bug free. But sometimes there can still be bad surprises, even after having compiled and checked `a2ps` on several very different platforms.

You may encounter some of these problems yourself. In any case, please never abandon without giving us a chance. We need information from everybody so that mistakes get fixed as fast as possible.

So, if you have a problem (configuration error, compilation error, runtime error, documentation error or unclear), first check in the FAQ (see Chapter 10 [FAQ], page 88), then on the page Known Bugs (`https://www.gnu.org/software/a2ps//bugs.html`) if the issue has not been addressed yet. If it is not the case, but it appears that the version of `a2ps` you have is old, consider upgrading.

If the problem persists, send us a mail (`bug-a2ps@gnu.org`) which subject is '`a2ps` *`version`*`:` *`short-description`*' and which content mentions the name of your machine and OS, the version of `a2ps`, every detail you have on your compiler, and as much traces as possible (the error messages you get on the screen, or the output of `make` when it fails etc.).

Be sure to get a quick answer.

1.3 a2ps Mailing Lists

There are several mailing lists related to `a2ps`:

`a2ps@gnu.org`
: This list is dedicated to announcements, questions/answers, etc. The alpha versions are announced too. Requests and suggestions can be sent there.

`bug-a2ps@gnu.org`
: Any bug report should be sent to this address. Please, be sure to state the version of `a2ps` in the subject of your message, together with a short description of the problem. In the body of the message, include all the information that might be relevant: the system you run, etc.

`a2ps-patches@gnu.org`
: Send patches, style sheets, new delegations etc. to this list. In other words, any candidate for inclusion into `a2ps` should be sent to this list. It also serves to coordinate the developers. If you are interested in the development of `a2ps`, then visit the Savannah a2ps page[1].

`a2ps-commit@gnu.org`
: Each time a change is made the main `a2ps` repository, a message is sent to this mailing list. For developers only.

[1] `https://savannah.gnu.org/projects/a2ps/`

To subscribe to any of these list, go to their web pages: a2ps[2], bug-a2ps[3], a2ps-patches[4], and a2ps-commit[5].

Be sure *never* to send a private message to one of the authors, as it is approximately the best means never to get an answer. In addition it is counter productive for the community, as the answer to your question might have interested more people.

1.4 Helping the Development

If you like `a2ps` and if you feel like helping, there are several things you can do.

Testing You just can't imagine how hard it is to make sure that the program that works perfectly here will work on your machine. Actually, in general the last weeks before a release are mostly dedicated to (Unix) portability issues.

So we **need** beta-testers! To be one is fairly simple: subscribe to the mailing-list where the betas are announced and distributed.

Translation
The interface of `a2ps` is under `GNU gettext` which means that all the messages can be translated, without having to look at the code of `a2ps`: you don't need to be a programmer at all. All the details are available on the a2ps translation page[6].

Style Sheets
Since `a2ps` is evolving and getting more powerful, the style sheets should be checked and improved. There are too many so that the authors work on them. Therefore if you feel your favorite language is not honored as it should be, improve the style sheet! (see Chapter 7 [Pretty Printing], page 48, for details.)

Encodings `a2ps` is wide open to any 8-bit encoding. If your language is not covered today by `a2ps`, you can easily provide the support yourself. Honestly, the trickiest part is to find correct **free** fonts that support your mother tongue (see Section 6.2 [Encoding Files], page 44, to know more).

Fonts There are still some characters missing in Ogonkify. See the list of missing characters[7] and the Ogonkify home page[8] for details.

Documentation
If you feel something is missing or is unclear, send us your contributions.

Porting Porting a program to special architectures (MS-DOS, OS/2 etc.), or building special packages (e.g., RPM) requires having an access to these architectures. If you feel like maintaining such a port, tell us.

---

[2] `https://mail.gnu.org/mailman/listinfo/a2ps`
[3] `https://mail.gnu.org/mailman/listinfo/bug-a2ps`
[4] `https://mail.gnu.org/mailman/listinfo/a2ps-patches`
[5] `https://mail.gnu.org/mailman/listinfo/a2ps-patches`
[6] `https://www.gnu.org/software/a2ps//po/`
[7] `https://www.irif.fr/~jch//software/ogonkify/missing.html`
[8] `https://www.irif.fr/~jch//software/ogonkify/`

Features Well, if you feel like doing something else, go ahead! But contact us, because we have quite a big stack of things we want to do or have started to do, and synchronizing might be useful.

2 User's Guide

This chapter is devoted to people who don't know a2ps yet: we try to give a soft and smooth introduction to the most useful features. For a reference manual, see Chapter 3 [Invoking a2ps], page 11. For the definition of some words, see Appendix A [Glossary], page 92, for questions you have, see Chapter 10 [FAQ], page 88.

2.1 Purpose

a2ps is a program that takes a text file (i.e., human readable), and makes a PostScript file out of it. Typically output is sent to a printer.

2.2 How to print

To print a file doc.txt, just give it to a2ps: the default setting should be the one you'd like:

```
gargantua ~ $ a2ps doc.txt
[doc.txt (plain): 9 pages on 5 sheets]
[Total: 9 pages on 5 sheets] sent to the default printer
```

a2ps sent the file doc.txt to the default printer, writing two columns of text on a single face of the sheet. Indeed, by default a2ps uses the option '-2', standing for two virtual pages.

2.2.1 Basics for Printing

Say you want to print the C file bar.c, and its header foo.h, on 4 virtual pages, and save it into the file foobar.ps. Just hit:

```
gargantua $ a2ps foo.h bar.c -4 -o foobar.ps
[foo.h (C): 1 page on 1 sheet]
[bar.c (C): 3 pages on 1 sheet]
[Total: 4 pages on 2 sheets] saved into the file 'foobar.ps'
```

The option '-4' tells a2ps to make four virtual pages: two rows by two columns. The option '-o foobar.ps' (which is the short version of '--output=foobar.ps') specifies the output file. Long options must always be separated by spaces, though short options with no arguments may be grouped.

Note too that the options may be specified before or after the files, it does not matter.

If you send foobar.ps to a printer, you'll discover that the keywords were highlighted, that the strings and comments have a different face. Indeed, a2ps is a *pretty-printer*: if it knows the (programming) language in which your file is written, it will try to make it look nice and clear on the paper.

But too bad: foo.h is only one virtual page long, and bar.c takes three. Moreover, the comments are essential in those files. And even worse: the system's default printer is out of ink. Thanks god, precious options may help you:

```
gargantua $ a2ps -4 -Av foo.h bar.c --prologue=gray -P lw
[foo.h (C): 1 page on 1 sheet]
[bar.c (C): 3 pages on 1 sheet]
[Total: 4 pages on 1 sheet] sent to the printer `lw'
```

Here the option '`-A`' is a short cut for the option '`--file-align`' which specifies how different files should be separated. This option allows several symbolic arguments: '`virtual`', '`rank`', '`page`', '`sheet`' (See Section 3.1.3 [Sheet Options], page 15, for more details). The value '`virtual`' means not to start each file on a different virtual pages.

So to fill the page is asked by '`--file-align=virtual`', or '`-A virtual`'. But symbolic arguments can be abbreviated when there are no ambiguity, so here, you can just use '`-Av`'.

The option '`-P lw`' means to print on the printer named '`lw`', and finally, the long option '`--prologue`' requires the use one of the alternative printing styles. There are other prologues (See Section 3.1.6 [Input Options], page 18, option '`--prologue`'), and you can even design yours (see Section 8.6 [Designing PostScript Prologues], page 80).

2.2.2 Special Printers

There are three special printers pre-defined.

The first one, `void`, sends the output to the trash. Its main use is to see how many pages would have been used.

```
gargantua ~ $ a2ps -P void parsessh.c
[parsessh.c (C): 33 pages on 17 sheets]
[Total: 33 pages on 17 sheets] sent to the printer `void'
```

The second, `display` sends the output to `Ghostview`, so that you can check the output without printing. Of course if you don't have `Ghostview`, it won't work... And it is up to you to configure another displaying application (see Section 4.5 [Your Printers], page 31).

The last, `file` saves the output into a file named after the file you printed (e.g., saves into `foo.ps` when you print `foo.c`).

2.2.3 Using Delegations

`a2ps` can decide that `a2ps` itself is not the right tool to do what you want. In that case it delegates the task to other programs. What you should retain from this, is, *forget that there are delegations*. Indeed, the interface with the delegations has been designed so that you don't need to be aware that they exist to use them. Do as usual.

As an example, if you need to print a PostScript file, just hit:

```
gargantua ~ $ a2ps article.ps -d
[article.ps (ps, delegated to PsNup): 7 pages on 4 sheets]
[Total: 8 pages on 4 sheets] sent to the default printer
```

While honoring your defaults settings, `a2ps` delegates the task to put two virtual pages per physical page to `psnup`, a powerful filter part of the famous `psutils` by Angus Duggan.

Suppose now that you want to display a Texinfo file. Then, provided you have all the programs `a2ps` needs, just hit

```
gargantua ~ $ a2ps a2ps.texi -P display
[a2ps.texi (texinfo, delegated to texi2dvi): 75 pages on 38 sheets]
[Total: 76 pages on 38 sheets] sent to the printer 'display'
```

Once the read documentation, you know you want to print just pages 10 to 20, plus the cover. Just hit:

```
gargantua ~ $ a2ps a2ps.texi --pages=1,10-20 -d
[a2ps.texi (texinfo, delegated to texi2dvi): 13 pages on 7 sheets]
[Total: 14 pages on 7 sheets] sent to the default printer
```

A final word: compressed files can be treated in the very same way:

```
gargantua ~ $ a2ps a2ps.texi.gz -a1,10-20 -d
[a2ps.texi (compressed, delegated to Gzip-a2ps): 13 pages on 7 sheets]
[Total: 14 pages on 7 sheets] sent to the default printer
```

You should be aware that:

– the option '`-Z`' enables the delegations if they are not (see '`--list=defaults`' for your settings);

– the set of delegations is customizable, to know the delegations your `a2ps` knows, consult '`a2ps --list=delegations`'.

2.2.4 Printing Duplex

If you still want to save more paper, and you are amongst the set of happy users of Duplex printers, `a2ps` will also be able to help you (See Appendix A [Glossary], page 92, for definitions). The option to specify Duplex printing is '`--sides=`*`mode`*' (see Section 3.1.9 [PostScript Options], page 23).

Here is how to print the documentation in Duplex and send it to the Duplex printer '`margot`':

```
quasimodo ~ a2ps/doc $ a2ps -s2 -Pmargot a2ps.texi
[a2ps.texi (texinfo, delegated to texi2dvi): 109 pages on 28 sheets]
[Total: 110 pages on 28 sheets] sent to the printer 'margot'
```

This is also valid for several files.

Actually, you can do something even more tricky: print a small book! This is much more complicated than printing Duplex, because the pages needs to be completely reorganized another way. This is precisely the job of `psbook`, yet another PsUtil from Angus Duggan. But there is a user option which encapsulates the magic sequence of options: '`book`'. Therefore, just run

```
quasimodo a2ps/doc $ a2ps -=book -Pmargot a2ps.texi
[a2ps.texi (texinfo, delegated to texi2dvi): 109 pages on 109 sheets]
[Total: 109 pages on 109 sheets] sent to the printer 'margot'
```

and *voila' !*, a booklet printed on margot!

We strongly discourage you to try with several files at once, because the tools then easily get lost. And, after all, the result will be exactly the same once you collated all the booklets together.

Another limitation is that this does not work if it is not sent to a printer. This kind of weird limitations will be solved in the future.

2.2.5 Checking the Defaults

If `a2ps` did not have the behavior expected, this may be because of the default settings given by your system administrator. Checking those default values is easy:

```
~ % a2ps --list=defaults
                Configuration status of a2ps 4.12a
                ==================================
Sheets:
-------
  medium          = A4, portrait
  page layout     = 1 x 1, rows first
  borders         = yes
  file alignment  = page
  interior margin = 0
More stuff deleted here
Internals:
----------
  verbosity level     = 2
  file command        = /usr/bin/file -L
  temporary directory = /tmp
  library path        =
        /home/akim/.a2ps
        /usr/share/a2ps/sheets
        /usr/share/a2ps/ps
        /usr/share/a2ps/encoding
        /usr/share/a2ps/afm
        /usr/share/ogonkify/afm
        /usr/share/a2ps/ppd
        /usr/share/a2ps/fonts
        /usr/share/ogonkify/fonts
        /usr/share/a2ps
```

Remember that the on-line help is always available. Moreover, if your screen is small, you may *pipe* it into `more`. Just trust this:

```
a2ps --help | more
```

2.3 Important parameters

Many things are parameterizable in `a2ps`, but two things are just essential to make sure everything goes right:

The paper Make sure that the paper `a2ps` uses is the same as your printer (See Section 3.1.3 [Sheet Options], page 15, option '`--medium`').

The encoding
Make sure that the *encoding* `a2ps` uses is the same as the standard alphabet in your country (See Section 3.1.6 [Input Options], page 18, option '`--encoding`').

Both values may be checked with '`a2ps --list=defaults`'.

2.4 Localizing

`a2ps` provides some Native Language Support, that is speaking your mother tongue. It uses three special features for non-English languages:

the tongue i.e., the language used by the interface,

the date i.e., the format and the words used in the language to specify a date.

To enable these features, properly set your environment variable `LANG` (see the documentation of your system, for instance '`man locale`', '`man environ`' etc.).

The problem with this approach is that a lot more than just messages and time information is affected: especially the way numbers are written changes, what may cause problems with `awk` and such.

So if you just want messages and time format to be localized, then define:

```
set LC_MESSAGES=fr ; export LC_MESSAGES
set LC_TIME=fr     ; export LC_TIME
```

2.5 Interfacing with Other Programs

Here are some tips on how to use `a2ps` with other programs.

2.5.1 Interfacing With a Mailer

When you print from a mailer (or a news reader), your mailer calls a tool, say `a2ps` on a part of the whole mailbox. This makes it difficult for `a2ps` to guess that the file is of the type '`mail`'. Therefore, for better results, make sure to tell `a2ps` the files are mails. The user option '`mail`' (or '`longmail`' for longer inputs) encapsulates most typical tuning users want to print mails (for instance, don't print all the headers).

Most specifically, if your mailer is:

`elm` Once you are in elm, hit *`o`* to enter in the options edition menu, hit *`p`* to edit the printer command, and enter '`a2ps -=mail %s -d`'. The option '`-d`' means to print on the default printer.

`pine` Jan Chrillesen suggests us how to use `a2ps` with the Pine mail-reader. Add the following to `.pinerc` (of course you can put it in `pine.conf` as well):

```
# Your printer selection
```

```
printer=a2ps -=mail -d

# Special print command
personal-print-command=a2ps -=mail -d
```

2.5.2 Processing the output of other programs

Use the following command:

```
a2ps
```

Not too hard, isn't it?

Nevertheless, this setting suppose your world is OK, your `file(1)` detects correctly PostScript files, and your `a2ps` is configured to delegate. In case one one these conditions is not met, use:

```
a2ps -ZEps
```

Do not forget to tell the program whose output you are processing whether your printer supports colors, and the type of paper it uses.

3 Invoking a2ps

Calling `a2ps` is fairly simple:

```
a2ps Options... Files...
```

If no *Files...* are given, `a2ps` prints its standard input. If '`-`' appears in the *Files...*, it designates the standard input too.

3.1 Command line options

To read the options and arguments that you give, `a2ps` uses GNU `getopt`, hence:

- the options (short with arguments or long) must be separated by spaces.
- the order between options and files does not matter: '`a2ps -4m main.c`' and '`a2ps main.c -4m`' are identical.
- the order between options **does matter**, especially between options that influence the same parameters. For instance '`a2ps -1 -l132`' is not the same as '`a2ps -l132 -1`' (the latter being equivalent to '`a2ps -1`').
- short options may be grouped together: '`a2ps -4mg main.c -P printer`'
- when there are no ambiguities, long options can be abbreviated, e.g., '`--pro`' will be understood as '`--prologue`',
- '`--`' ends the options. Anything behind '`--`' is considered to be a file: '`a2ps -- -2`' prints the file `-2`[1].

Here after a *boolean* is considered as true (i.e. setting the option on), if *boolean* is '`yes`', or '`1`'; as false if it equals '`no`' or '`0`'; and raise an error otherwise. The corresponding short option takes no arguments, but corresponds to a positive answer.

When an argument is presented between square brackets, it means that it is optional. Optional arguments to short option must never be separated from the option.

3.1.1 Tasks Options

Task options specify the task `a2ps` will perform. It will not print, it executes the task and exits successfully.

`--version` [Option]
: print version and exit successfully.

`--help` [Option]
: Print a short help, and exit successfully.

`--copyright` [Option]
: Display Copyright and copying conditions, and exit successfully.

`--guess` [Option]
: Act like `file` does: display the (key of the) type of the *Files*.

  For instance, on a `C` file, you expect it to answer '`c`', and upon a PostScript file, '`ps`'.

  This can be very useful on broken systems to understand why a file is printed with a bad style sheet (see Section 5.4 [Style Sheet Files], page 41).

[1] A classical Unix trick to make the difference between the option '`-2`', and the file `-2` is to type `./-2`.

`--which` [Option]

Look in the library for the files which names are given as arguments. For instance:

```
~ % a2ps --which bw.pro gray.pro
/usr/local/share/a2ps/ps/bw.pro
/usr/local/share/a2ps/ps/gray.pro
```

If there are several library files matching the name, only the first one is reported: this allows to check which occurrence of a file is used by `a2ps`.

`--glob` [Option]

Look in the library for the files which names match the patterns given as arguments. For instance:

```
~ % a2ps --glob 'g*.pro'
/usr/local/share/a2ps/ps/gray.pro
/usr/local/share/a2ps/ps/gray2.pro
```

`--list=`*topic* [Option]

Display a report on `a2ps`' status with respect to *topic*, and exit successfully. *topic* can be any non-ambiguous abbreviation of:

'`defaults`'
'`options`' Give an extensive report on `a2ps` configuration and installation.

'`features`'
Known media, encodings, languages, prologues, printers, variables, delegations and user options are reported. In a word, anything that you may define.

'`delegations`'
Detailed list of the delegations. See Section 4.10 [Your Delegations], page 35.

'`encodings`'
Detailed list of known encodings. See Section 6.2.3 [Some Encodings], page 45.

'`media`' Detailed list of known media. See Section 4.4 [Your Media], page 31.

'`prologues`'
Detailed list of PostScript prologues. See Section 8.6 [Designing PostScript Prologues], page 80.

'`printers`'
Detailed list of printers and named outputs. See Section 4.5 [Your Printers], page 31.

'`style-sheets`'
Detailed list of the known style sheets. See Section 7.2 [Known Style Sheets], page 48.

‘user-options’
Detailed list of the user options. See Section 4.6 [Your Shortcuts], page 32.

‘variables’
Detailed list of the variables. See Section 4.9 [Your Variables], page 33.

There are also options meant for the maintainers only, presented for sake of completeness.

‘texinfo-style-sheets’
‘ssh-texi’
Detailed list of known style sheets in Texinfo format. If the sheet verbosity is set, report version numbers, requirements and ancestors.

‘html-style-sheets’
‘ssh-html’
Detailed list of the style sheets in HTML format.

‘texinfo-encodings’
‘edf-texi’
Detailed list of encodings, in Texinfo format.

‘texinfo-prologues’
‘pro-texi’
Detailed list of prologues, in Texinfo format.

3.1.2 Global Options

These options are related to the interface between you and a2ps.

-q [Option]
--quiet [Option]
--silent [Option]
be really quiet

-v[*level*] [Option]
--verbose[=*level*] [Option]
tell what we are doing. At

- *level* = 0, report nothing,
- *level* = 1, a2ps just prints the total number of pages printed,
- *level* = 2 (default), it reports it for each file,
- above, it gives internal details.

There is also an interface made for the maintainer with finer grained selection of the verbosity level. *level* is a list of tokens (non ambiguous abbreviations are valid) separated by either ‘,’ or ‘+’. The tokens may be:

‘configuration’
‘options’ reading the configurations files and the options,

‘encodings’
the encodings,

‘expert’ more detailed information is provided: PPD listings is exhaustive,

‘files’ inputs and outputs,

‘fonts’ the fonts,

‘escapes’
‘variables’
‘meta-sequences’
the expansion of escapes and variables,

‘parsers’ any parsing process (style sheets, PPD files etc.),

‘pathwalk’
‘pw’ the search for files,

‘ppd’ PPD processing,

‘sheets’ the style sheets,

‘stats’ statistics on some internal data structures,

‘tools’ launched programs or shell commands ; triggers the escape ‘?V’ on (see Section 3.2.3 [Available Escapes], page 25),

‘all’ all the messages.

When a2ps is launched it consults the environment variable A2PS_VERBOSITY. If it is set, this defines the verbosity level for the whole session (options ‘--verbose’, and ‘-q’ etc. have then no influence). The valid values for A2PS_VERBOSITY are exactly the valid arguments of the option ‘--verbose’. This helps tracking down configuration problems that occur *before* a2ps had even a chance to read the command line.

`-=`*shortcut* [Option]
`--user-option=`*shortcut* [Option]
use the *shortcut* defined by the user. See Section 4.6 [Your Shortcuts], page 32. Shortcuts may be freely mixed with regular options and arguments.

There are a few predefined user-options:

‘lp’ emulates a line printer, i.e., turn off most ‘pretty’ features.

‘mail’
‘longmail’
preferred options to print a mail or a news. ‘longmail’ prints more text on a single sheet.

‘manual’ make the job be printed on the manually fed tray.

`--debug` [Option]
enable debugging features. They are:

- print the overall BoundingBox in PostScript;
- down load a PostScript debugger which helps understanding why a printer may reject a file.

`-D` *key*[=*value*] [Option]
`--define=`*key*`[=`*value*`]` [Option]
Without *value*, unset the variable *key*. Otherwise, set it to *value*. See Section 4.9 [Your Variables], page 33, for more details. Note that '`-Dfoo=`' gives *foo* an empty value, though '`-Dfoo`' unsets foo.

3.1.3 Sheet Options

This options specify the general layout, how the sheet should be used.

`-M` *medium* [Option]
`--medium=`*medium* [Option]
use output medium *medium*. See the output of '`a2ps --list=media`' for the list of supported media. Typical values are '`A3`', '`A4`', '`A5`', '`B4`', '`B5`', '`Letter`', '`Legal`'. The default is the user's preferred paper size, or the system default; see the man page of `paper` for how this is configured. The default paper size may also be requested explicitly with the name '`libpaper`'.

`-r` [Option]
`--landscape` [Option]
print in landscape mode

`-R` [Option]
`--portrait` [Option]
print in portrait mode

`--columns=`*num* [Option]
specify the number of columns of virtual pages per physical page.

`--rows=`*num* [Option]
specify the number of rows of virtual pages per physical page.

`--major=`*direction* [Option]
specify whether the virtual pages should be first filled in rows (*direction* = '`rows`') or in columns (*direction* = '`columns`').

`-1` [Option]
1 x 1 portrait, 80 chars/line, major rows (i.e. alias for '`--columns=1 --rows=1 --portrait --chars-per-line=80 --major=rows`').

`-2` [Option]
2 x 1 landscape, 80 chars/line, major rows.

`-3` [Option]
3 x 1 landscape, 80 chars/line, major rows.

`-4` [Option]
2 x 2 portrait, 80 chars/line, major rows.

`-5` [Option]
5 x 1 landscape, 80 chars/line, major rows.

`-6` [Option]
3 x 2 landscape, 80 chars/line, major rows.

`-7` [Option]
7 x 1 landscape, 80 chars/line, major rows.

`-8` [Option]
4 x 2 landscape, 80 chars/line, major rows.

`-9` [Option]
3 x 3 portrait, 80 chars/line, major rows.

`-j` [Option]
`--borders=`*boolean* [Option]
print borders around virtual pages.

`-A` *mode* [Option]
`--file-align=`*mode* [Option]
Align separate files according to *mode*. This option allows the printing of more than one file on the same page. *mode* can be any one of:

'`virtual`' Each file starts on the next available virtual page (i.e., leave no empty virtuals).

'`rank`' Each file starts at the beginning of the next row or column depending on the '`--major`' setting.

'`page`' Each file starts on a new page.

'`sheet`' Each file starts on a new sheet. In Simplex mode, this is the same as '`page`', in Duplex mode, files always start on a front side.

an integer *num*
Each file starts on a page which is a multiple of *num* plus 1. For instance, for '`2`', the files must start on odd pages.

`--margin[=`*num*`]` [Option]
Specify the size of the margin (*num* PostScript points, or 12 points without arguments) to leave in the inside (i.e. left for the front side page, and right for the back side). This is intended to ease the binding.

3.1.4 Page Options

This options are related to the content of the virtual pages.

Please note that the options '`-f`', '`-L`', '`-l`', '`-m`', and '`-1`' .. '`-9`' all have an influence on the font size. Only the last one will win (i.e., '`a2ps -L66 -l80`' is the same as '`a2ps -l80`').

`--line-numbers[=`*number*`]` [Option]
print the line numbers from *number* lines to *number* lines. Default is '`1`'.

`-C` [Option]
Alias for '`--line-numbers=5`'.

`-f` *size*[*unit*] [Option]
`--font-size=`*size*[*unit*] [Option]
scale font to *size* for body text. *size* is a float number, and *unit* can be '`cm`' for centimeters, '`points`' for PostScript points, and '`in`' for inches. Default unit in '`points`'.

To change the fonts used, change the current prologue (see Section 8.6 [Designing PostScript Prologues], page 80.

`-l` *num* [Option]
`--chars-per-line=`*num* [Option]
Set the font size so that *num* columns appear per virtual pages. *num* is the real number of columns devoted to the body of the text, i.e., no matter whether lines are numbered or not.

`-L` *num* [Option]
`--lines-per-page=`*num* [Option]
Set the font size so that *num* lines appear per virtual pages. This is useful for printing preformatted documents which have a fixed number of lines per page. The minimum number of lines per page is set at 40 and maximum is at 160. If a number less than 40 is supplied, scaling will be turned off.

`-m` [Option]
`--catman` [Option]
Understand UNIX manual **output** ie: 66 lines per page and possible bolding and underlining sequences. The understanding of bolding and underlining is there by default even if '`--catman`' is not specified. You may want to use the '`ul`' prologue (See Section 3.1.6 [Input Options], page 18, option '`--prologue`') if you prefer underlining over italics.

If your file is actually a UNIX manual *input*, i.e., a roff file, then depending whether you left `a2ps` delegate or not, you will get a readable version of the text described, or a pretty-printed version of the describing file (see Section 4.10 [Your Delegations], page 35).

`-T` *num* [Option]
`--tabsize=`*num* [Option]
set tabulator size to *num*. This option is ignored if `--interpret=no` is given.

`--non-printable-format=`*format* [Option]
specify how non-printable chars are printed. *format* can be

'`caret`' Use classical Unix representation: '`^A`', '`M-^B`' etc.

'`space`' A space is written instead of the non-printable character.

'`question-mark`'
A '`?`' is written instead of the non-printable character.

'`octal`' For instance '`\001`', '`177`' etc.

'`hexa`' For instance '`\x01`', '`\xfe`' etc.

'`emacs`' For instance '`C-h`', '`M-C-c`' etc.

3.1.5 Headings Options

These are the options through which you may define the information you want to see all around the pages.

All these options support *text* as an argument, which is composed of plain strings and escapes. See Section 3.2 [Escapes], page 24, for details.

`-B` [Option]
`--no-header` [Option]
no page headers at all.

`-b[`*text*`]` [Option]
`--header[=`*text*`]` [Option]
set the page header

`--center-title[=`*text*`]` [Option]
`--left-title[=`*text*`]` [Option]
`--right-title[=`*text*`]` [Option]
Set virtual page center, left and right titles to *text*.

`-u[`*text*`]` [Option]
`--underlay[=`*text*`]` [Option]
use *text* as *under lay* (or *water mark*), i.e., in a light gray, and under every page.

`--left-footer[=`*text*`]` [Option]
`--footer[=`*text*`]` [Option]
`--right-footer[=`*text*`]` [Option]
Set sheet footers to *text*.

3.1.6 Input Options

`-a[`*Page range*`]` [Option]
`--pages[=`*Page range*`]` [Option]
With no argument, print all the page, otherwise select the pages to print. *Page range* is a list of interval, such as '`-a1`': print only the first page, '`-a-3,4,6,10-`': print the first 3 pages, page 4 and 6, and all the page after 10 (included). Giving '`toc`' prints the table of content whatever its page number is.

The pages referred to are the *input* pages, not the output pages, that is, in '`-2`', printing with '`-a1`' will print the first virtual page, i.e., you will get half the page filled.

Note that page selection does work with the delegations (see Section 4.10 [Your Delegations], page 35).

`-c` [Option]
`--truncate-lines=`*boolean* [Option]
Cut lines too large to be printed inside the borders. The maximum line size depends on format and font size used and whether line numbering is enabled.

`-i` [Option]
`--interpret=`*boolean* [Option]
interpret tab and ff chars. This means that '`^L`' jumps to a new (virtual) pages, '`tab`' advances to the next tabulation.

`--end-of-line=`*type* [Option]
Specify what sequence of characters denotes the end of line. *type* can be:

`n`
`unix` '`\n`'.

`r`
`mac` '`\r`'.

`nr` '`\n\r`'. As far as we know, this type of end-of-line is not used.

`pc`
`rn` '`\r\n`'. This is the type of end-of-line on MS-DOS.

`any`
`auto` Any of the previous cases. This last case prevents the bad surprises with files from PC (trailing '`^M`').

`-X` *key* [Option]
`--encoding=`*key* [Option]
Use the input encoding identified by *key*. See Section 6.2.3 [Some Encodings], page 45, and the result of '`a2ps --list=encodings`' to know what encodings are supported. Typical values are '`ASCII`', '`latin1`'... '`latin6`', '`ison`' etc.

`--stdin=`*filename* [Option]
Give the name *filename* to the files read through the standard input.

`-t` *name* [Option]
`--title=`*name* [Option]
Give the name *name* to the document. Escapes can be used (see Section 3.2 [Escapes], page 24).

This is used for instance in the name given to the document from within the PostScript code (so that `Ghostview` and others can display a file with its real title, instead of just the PostScript file name).

It is **not** the name of the output. It is just a logical title.

`--prologue=`*prologue* [Option]
Use *prologue* as the PostScript prologue for `a2ps`. *prologue* must be in a file named *prologue*`.pro`, which must be in a directory of your library path (see Chapter 5 [Library Files], page 38). Available prologues are:

'`bold`' This style is meant to replace the old option `-b` of a2ps 4.3. It is a copy of the black and white prologue, but in which all the fonts are in Bold.

'`bw`' Style is plain: pure black and white, with standard fonts.

'`color`' Colors are used to highlight the keywords.

'diff' This style is meant to be used with the udiff, wdiff style sheets, to underline the differences. New things are in bold on a diff background, while removed sequences are in italic.

'diffcolor'
Colors are used to highlight the keywords (for diffs).

'fixed' This style uses exclusively fixed size fonts. You should use this style if you want the tabulations to be properly printed.

There are no means to use a fixed size Symbol font, therefore you should not use the heavy highlighting style.

'gray' Gray background is used for comments and labels.

'gray2' Black background is used for comments and labels.

'matrix' The layout is the same as 'bw', but alternating gray and white lines. There are two macros defining the behavior: 'pro.matrix.cycle' defines the length of the cycle (number of white and gray lines). It defaults to 6. 'pro.matrix.gray' defines the number of gray lines. Default is 3.

'ul' This style uses bold faces and underlines, but never italics. This is particularly meant for printing formatted man pages.

`--print-anyway=`*boolean* [Option]
force binary printing. By default, the whole print job is stopped as soon as a binary file is detected. To detect such a file we make use of a very simple heuristic: if the first sheet of the file contains more than 40% of non-printing characters, it's a binary file. a2ps also asks file(1) what it thinks of the type of the file. If file(1) answers 'data', the file will also be considered as binary, hence not printed.

`-Z` [Option]
`--delegate=`*boolean* [Option]
Enable delegation of some files to delegated applications. If delegating is on, then a2ps will *not* process the file by itself, but will call an application which handles the file in another way. If delegation is off, then a2ps will process *every* file itself.

Typically most people don't want to pretty-print a PostScript source file, but want to print what describes that file. Then set the delegations on.

See Section 4.10 [Your Delegations], page 35, for information on delegating, and option '--list=delegations' for the applications your a2ps knows.

`--toc[=`*format*`]` [Option]
Generate a Table of Contents, which *format* is an escape (see Section 3.2 [Escapes], page 24) processed as a PreScript file (see Section 7.3.2 [PreScript], page 60). If no *format* is given (i.e., you wrote '--toc'), use the default table of contents shape (#{toc}). If the given format is empty (i.e., you wrote '--toc='), don't issue the table of contents.

Note that it is most useful to define a variable (see Section 4.9 [Your Variables], page 33), for instance, in a configuration file:

```
Variable: toc.mine \
```

```
\\Keyword{Table of Content}\n\
#-1!f\
|$2# \\keyword{$-.20n} sheets $3s< to $3s> ($2s#) \
pages $3p<-$3p> $4l# lines\n||\
\\Keyword{End of toc}\n
```

and to give that variable as argument to '`--toc`': '`a2ps *.c --toc=#{toc.mine}`'.

Note too that you can generate only the table of content using '`--pages`':

```
a2ps *.c --toc -atoc
```

3.1.7 Pretty Printing Options

These options are related to the pretty printing features of `a2ps`.

`--highlight-level=`*level* [Option]

Specify the *level* of highlighting. *level* can be

'`none`' no highlighting

'`normal`' regular highlighting

'`heavy`' even more highlighting.

See the documentation of the style sheets ('`--list=style-sheets`') for a description of '`heavy`' highlighting.

`-g` [Option]

Alias for '`--highlight-level=heavy`'.

`-E[`*language*`]` [Option]
`--pretty-print[=`*language*`]` [Option]

With no arguments, set automatic style selection on. Otherwise, set style to *language*. Note that setting *language* to '`plain`' turns off pretty-printing. See Section 7.2 [Known Style Sheets], page 48, and the output of '`--list=style-sheets`' for the available style sheets.

If *language* is '*`key`*`.ssh`', then don't look in the library path, but use the file *`key.ssh`*. This is to ease debugging non installed style sheets.

`--strip-level=`*num* [Option]

Depending on the value of *num*:

'`0`' everything is printed;

'`1`' regular comments are not printed

'`2`' strong comments are not printed

'`3`' no comment is printed.

This option is valuable for instance in `java` in which case strong comments are the so called documentation comments, or in `SDL` for which some graphical editors pollutes the specification with internal data as comments.

Note that the current implementation is not satisfactory: some undesired blank lines remain. This is planed to be fixed.

3.1.8 Output Options

These are the options to specify what you want to do out of what a2ps produces. Only a single destination is possible at a time, i.e., if ever there are several options '-o', '-P' or '-d', the last one is honored.

`-o` *file* [Option]
`--output=`*file* [Option]
leave output to file *file*. If *file* is '-', leave output to the standard output.

`--version-control=`*type* [Option]
to avoid loosing a file, a2ps offers backup services. This is enabled when the output file already exists, is regular (that is, no backup is done on special files such as /dev/null), and is writable (in this case, disabling version control makes a2ps fail the very same way as if version control was disabled: permission denied).

The type of backups made can be set with the VERSION_CONTROL environment variable, which can be overridden by this option. If VERSION_CONTROL is not set and this option is not given, the default backup type is 'existing'. The value of the VERSION_CONTROL environment variable and the argument to this option are like the GNU Emacs 'version-control' variable; they also recognize synonyms that are more descriptive. The valid values are (unique abbreviations are accepted):

'none'
'off' Never make backups (override existing files).

't'
'numbered'
Always make numbered backups.

'nil'
'existing'
Make numbered backups of files that already have them, simple backups of the others.

'never'
'simple' Always make simple backups.

`--suffix=`*suffix* [Option]
The suffix used for making simple backup files can be set with the SIMPLE_BACKUP_SUFFIX environment variable, which can be overridden by this option. If neither of those is given, the default is '~', as it is in Emacs.

`-P` *name* [Option]
`--printer=`*name* [Option]
send output to printer *name*. See item 'Printer:' and 'Unknown printer:' in Section 4.5 [Your Printers], page 31, and results of option '--list=defaults' to see the bindings between printer names and commands.

It is possible to pass additional options to lpr or lp via the variable 'lp.options', for more information see Section 10.2.5 [Pass Options to lpr], page 91.

`-d` [Option]

send output to the default printer. See item '`DefaultPrinter:`' in Section 4.5 [Your Printers], page 31.

3.1.9 PostScript Options

The following options are related only to variations you want to produce onto a PostScript output.

`--ppd[=`*key*`]` [Option]

With no argument, set automatic PPD selection, otherwise set the PPD to *key*. FIXME: what to read.

`-n` *num* [Option]
`--copies=`*num* [Option]

print *num* copies of each page

`-s` *duplex-mode* [Option]
`--sides=`*duplex-mode* [Option]

Specify the number of sheet sides, or, more generally, the Duplex mode (see Appendix A [Glossary], page 92). The valid values for *duplex-mode* are:

'`1`'
'`simplex`' One page per sheet.

'`2`'
'`duplex`' Two pages per sheet, DuplexNoTumble mode.

'`tumble`' Two pages per sheet, DuplexTumble mode.

Not only does this option require Duplex from the printer, but it also enables duplex features from `a2ps` (e.g., the margin changes from front pages to back pages etc.).

`-S` *key*`[:`*value*`]` [Option]
`--setpagedevice=`*key*`[:`*value*`]` [Option]

Pass a page device definition to the generated PostScript output. If no *value* is given, *key* is removed from the definitions. Note that several '`--setpagedevice`' can be accumulated.

For example, command

```
ubu $ a2ps -SDuplex:true -STumble:true NEWS
[NEWS (plain): 15 pages on 8 sheets]
[Total: 15 pages on 8 sheets] sent to the default printer
```

prints file `report.pre` in duplex (two sides) tumble (suitable for landscape documents). This is also valid for delegated files:

```
a2ps -SDuplex:true -STumble:true a2ps.texi
```

Page device operators are implementation dependent but they are standardized. See Section 8.2 [Page Device Options], page 79, for details.

`--statusdict=`*key*`[:`*value*`]` [Option]
`--statusdict=`*key*`[::`*value*`]` [Option]

Pass a statusdict definition to the generated PostScript output. `statusdict` operators and variables are implementation dependent; see the documentation of your

printer for details. See Section 8.3 [Statusdict Options], page 79, for details. Several '`--statusdict`' can be accumulated.

If no *value* is given, *key* is removed from the definitions.

With a single colon, pass a call to an operator, for instance:

```
a2ps --statusdict=setpapertray:1 quicksort.c
```

prints file `quicksort.c` by using paper from the paper tray 1 (assuming that printer supports paper tray selection).

With two colons, define variable *key* to equal *value*. For instance:

```
a2ps --statusdict=papertray::1 quicksort.c
```

produces

```
/papertray 1 def
```

in the PostScript.

`-k` [Option]
`--page-prefeed` [Option]

enable page prefeeding. It consists in positioning the sheet in the printing area while the PostScript is interpreted (instead of waiting the end of the interpretation of the page before pushing the sheet). It can lead to an significant speed up of the printing.

`a2ps` quotes the access to that feature, so that non supporting printers won't fail.

`-K` [Option]
`--no-page-prefeed` [Option]

disable page prefeeding.

3.2 Escapes

The escapes are some sequences of characters that will be replaced by their values. They are very much like variables.

3.2.1 Use of Escapes

They are used in several places in `a2ps`:

Page markers

Headers, footers, titles and the water mark (see Section 3.1.5 [Headings Options], page 18), in general to print the name of file, page number etc. On a new sheet `a2ps` first draws the water mark, then the content of the first page, then the frame of the first page, (ditto with the others), and finally the sheet header and footers. This order must be taken into account for some escapes (e.g., '`$l.`', '`$l^`').

Named output

To specify the generic name of the file to produce, or how to access a printer (see Section 4.5 [Your Printers], page 31).

Delegation

To specify the command associated to a delegation (see Section 4.10 [Your Delegations], page 35).

Table of Content
: To specify an index/table of content printed at the end of the job.

Variables in PostScript prologue
: To allow the user to change some parameters to your prologues (see Section 8.6 [Designing PostScript Prologues], page 80).

3.2.2 General Structure of the Escapes

All format directives can also be given in format

escape width directive

where

escape In general

- '`%`' escapes are related to general information (e.g., the current date, the user's name etc.),
- '`#`' escapes are related to the output (e.g., the output file name) or to the options you gave (e.g., the number of virtual pages etc.), or to special constructions (e.g., enumerations of the files, or tests etc.),
- '`$`' escapes are related to the current input file (e.g., its name, its current page number etc.),
- '`\`' introduces classical escaping, or quoting, sequences (e.g., '`\n`', '`\f`' etc.).

width Specifies the width of the column to which the escape is printed. There are three forms for *width*

'`+`*`paddinginteger`*'
: the result of the expansion is prefixed by the character *padding* so that the whole result is as long as *integer*. For instance '`$+.10n`' with a file name '`$n`'=`foo.c` gives '`.....foo.c`'.

  If no *padding* is given, ' ' (white space) is used.

'`-`*`paddinginteger`*'
: Idem as above, except that completion is done on the left: '`$+.10n`' gives '`foo.c.....`'.

'*`integer`*' which is a short cut for '`+`*`integer`*'. For example, escape '`$5P`' will expand to something like ' `12`'.

directive See Section 3.2.3 [Available Escapes], page 25.

3.2.3 Available Escapes

Supported escapes are:

'`\\`' character '`\`'

'`\%`' character '`%`'

'`\$`' character '`$`'

‘\#’ character ‘#’

‘#?*cond*|*if_true*|*if_false*|’
this may be used for conditional assignment. The separator (presented here as ‘|’) may be any character. *if_true* and *if_false* may be defined exactly the same way as regular headers, included escapes and the ‘#?’ construct.

The available tests are:

‘#?1’
‘#?2’
‘#?3’ true if tag 1, 2 or 3 is not empty. See item ‘$t1’ for explanation.

‘#?d’ true if Duplex printing is requested (‘-s2’).

‘#?j’ true if bordering is asked (‘-j’).

‘#?l’ true if printing in landscape mode.

‘#?o’ true if only one virtual page per page (i.e., ‘#v’ is 1).

‘#?p’ a page range has been specified (i.e., ‘#p’ is not empty).

‘#?q’ true if a2ps is in quiet mode.

‘#?r’ true if major is rows (‘--major=rows’).

‘#?v’ true if printing on the back side of the sheet (verso).

‘#?V’ true if verbosity level includes the ‘tools’ flag (See Section 3.1.2 [Global Options], page 13. option ‘--verbosity’).

‘#!*key*|*in*|*between*|’
Used for enumerations. The separator (presented here as ‘|’) may be any character. *in* and *between* are escapes.

The enumerations may be:

‘#!$’ enumeration of the command line options. In this case *in* in never used, but is replaced by the arguments.

‘#!f’ enumeration of the input files in the other they were given.

‘#!F’ enumeration of the input files in the alphabetical order of their names.

‘#!s’ enumeration of the files appearing in the current sheet.

For instance, the escapes ‘The files printed were: #!f|$n|, |.’ evaluated with input ‘a2ps NEWS main.c -o foo.ps’, gives ‘The files printed were: NEWS, main.c.’.

As an exception, ‘#!’ escapes use the *width* as the maximum number of objects to enumerate if it is positive, e.g., ‘#10!f|$n|, |’ lists only the ten first file names. If *width* is negative, then it does not enumerate the -*width* last objects (e.g., ‘#-1!f|$n|, |’ lists all the files but the last).

‘${*var*}’ value of the environment variable *var* if defined, nothing otherwise.

‘${*var*:-*word*}’
: if the environment variable *var* is defined, then its value, otherwise *word*.

‘${*var*:+*word*}’
: if the environment variable *var* is defined, then *word*, otherwise nothing.

‘$[*num*]’
: value of the *num*th argument given on the command line. Note that $[0] is the name under which a2ps has been called.

‘#{*key*}’
: expansion of the value of the variable *key* if defined, nothing otherwise (see Section 4.9 [Your Variables], page 33)

‘#{*key*:-*word*}’
: if the variable *var* is defined, then the expansion of its, otherwise *word*.

‘#{*key*:+*word*}’
: if the variable *var* is defined, then *word*, otherwise nothing.

‘#.’
: the extension corresponding to the current output language (e.g. ‘ps’).

‘%*’
: current time in 24-hour format with seconds ‘hh:mm:ss’

‘$*’
: file modification time in 24-hour format with seconds ‘hh:mm:ss’

‘$#’
: the sequence number of the current input file

‘%#’
: the total number of files

‘%a’
: the localized equivalent for ‘Printed by *User Name*’. *User Name* is obtained from the variable ‘user.name’ (see Section 4.9.2 [Predefined Variables], page 34).

‘%A’
: the localized equivalent for ‘Printed by *User Name* from *Host Name*’. The variables ‘user.name’ and ‘user.host’ are used (see Section 4.9.2 [Predefined Variables], page 34).

‘%c’
: trailing component of the current working directory

‘%C’
: current time in ‘hh:mm:ss’ format

‘$C’
: file modification time in ‘hh:mm:ss’ format

‘%d’
: current working directory

‘$d’
: directory part of the current file (‘.’ if the directory part is empty).

‘%D’
: current date in ‘yy-mm-dd’ format

‘$D’
: file modification date in ‘yy-mm-dd’ format

‘%D{*string*}’
: format current date according to *string* with the strftime(3) function.

‘$D{*string*}’
: format file's last modification date according to *string* with the strftime(3) function.

‘%e’
: current date in localized short format (e.g., ‘Jul 4, 76’ in English, or ‘14 Juil 89’ in French).

‘$e’ file modification date in localized short format.

‘%E’ current date in localized long format (e.g., ‘July 4, 76’ in English, or ‘Samedi 14 Juillet 89’ in French).

‘$E’ file modification date in localized long format.

‘$f’ full file name (with directory and suffix).

‘\f’ character ‘\f’ (form feed).

‘#f0’
‘#f9’ ten temporary file names. You can do anything you want with them, a2ps removes them at the end of the job. It is useful for the delegations (see Section 4.10 [Your Delegations], page 35) and for the printer commands (see Section 4.5 [Your Printers], page 31).

‘%F’ current date in ‘dd.mm.yyyy’ format.

‘$F’ file modification date in ‘dd.mm.yyyy’ format.

‘#h’ medium height in PostScript points

‘$l^’ top most line number of the current page

‘$l.’ current line number. To print the page number and the line interval in the right title, use ‘--right-title="$q:$l^-$l."’.

‘$l#’ number of lines in the current file.

‘%m’ the host name up to the first ‘.’ character

‘%M’ the full host name

‘\n’ the character ‘\n’ (new line).

‘%n’ shortcut for the value of the variable ‘user.login’ (see Section 4.9.2 [Predefined Variables], page 34).

‘$n’ input file name without the directory part.

‘%N’ shortcut for the value of the variable ‘user.name’ (see Section 4.9.2 [Predefined Variables], page 34).

‘$N’ input file name without the directory, and without its suffix (e.g., on foo.c, it will produce ‘foo’).

‘#o’ name of the output, before substitution (i.e., argument of ‘-P’, or of ‘-o’).

‘#O’ name of the output, after substitution. If output goes to a file, then the name of the file. If the output is a symbolic printer (see Section 4.5 [Your Printers], page 31), the result of the evaluation. For instance, if the symbolic printer ‘file’ is defined as ‘> $n.%.’, then ‘#O’ returns ‘foo.c.ps’ when printing foo.c to PostScript. ‘#o’ would have returned ‘file’.

‘#p’ the range of the page to print from this page. For instance if the user asked ‘--pages=1-10,15’, and the current page is 8, then ‘#p’ evaluates to ‘1-3,8’.

‘$p^’ number of the first page of this file appearing on the current sheet. Note that ‘$p.’, evaluated at the end of sheet, is also the number of the last page of this file appearing on this sheet.

‘$p-’ interval of the page number of the current file appearing on the current sheet. It is the same as ‘p^-p.’, if ‘$p^’ and ‘$p.’ are different, otherwise it is equal to ‘$p.’.

‘%p.’ current page number

‘$p.’ page number for this file

‘%p#’ total number of pages printed

‘$p#’ number of pages of the current file

‘$p<’ number of the first page of the current file

‘$p>’ number of the last page of the current file

‘%q’ localized equivalent for ‘Page %p.’

‘$q’ localized equivalent for ‘Page $p.’

‘%Q’ localized equivalent for ‘Page %p./%p#’

‘$Q’ localized equivalent for ‘Page $p./$p#’

‘$s<’ number of the first sheet of the current file

‘%s.’ current sheet number

‘$s.’ sheet number for the current file

‘$s>’ number of the last sheet of the current file

‘%s#’ total number of sheets

‘$s#’ number of sheets of the current file

‘%t’ current time in 12-hour am/pm format

‘$t’ file modification time in 12-hour am/pm format

‘$t1’
‘$t2’
‘$t3’ Content of tag 1, 2 and 3. Tags are pieces of text a2ps fetches in the files, according to the style. For instance, in mail-folder style, tag 1 is the title of the mail, and tag 2 its author.

‘%T’ current time in 24-hour format ‘hh:mm’

‘$T’ file modification time in 24-hour format ‘hh:mm’

‘#v’ number of virtual sheets

‘%V’ the version string of a2ps.

‘#w’ medium width in PostScript points

‘%W’ current date in ‘mm/dd/yy’ format

‘$W’ file modification date in ‘mm/dd/yy’ format

4 Configuration Files

`a2ps` reads several files before the command line options. In order, they are:

1. the system configuration file (typically `/usr/local/etc/a2ps.cfg`) unless you have defined the environment variable '`A2PS_CONFIG`', in which case `a2ps` reads the file it points to;
2. the user's home configuration file (`$HOME/.a2ps/a2psrc`)
3. the local file (`./.a2psrc`)

Because `a2ps` needs architecture dependent information (such as the local `lpr` command) and architecture independent information (such as the type of your printers), users have found useful that `a2ps.cfg` be dedicated to architecture dependent information. A sub configuration file, `a2ps-site.cfg` (see Section 4.1 [Including Configuration Files], page 30) is included from `a2ps.cfg`.

The file `a2ps.cfg` is updated when you update `a2ps`, while `a2ps-site.cfg` is not, to preserve local definitions.

In the configuration files, empty lines and lines starting with '`#`' are comments.

The other lines have all the following form:

`Topic: Arguments`

where *Topic:* is a keyword related to what you are customizing, and *Arguments* the customization. *Arguments* may be spread on several lines, provided that the last character of a line to continue is a '\'.

In the following sections, each *Topic:* is detailed.

4.1 Including Configuration Files

`Include: file` [Configuration Setting]

Include (read) the configuration *file*. if *file* is a relative path (i.e., it does not start with '`/`'), then it is relatively to the current configuration file.

This is especially useful for the site specific configuration file `etc/a2ps.cfg`: you may tune your printers etc. in a separate file for easy upgrade of `a2ps` (and hence of its configuration files).

4.2 Your Library Path

To define the default library path, you can use:

`LibraryPath: path` [Configuration Setting]

Set the library path the *path*.

`AppendLibraryPath: path` [Configuration Setting]

Add *path* at the end of the current library path.

`PrependLibraryPath: path` [Configuration Setting]

Add *path* at the beginning of the current library path.

Note that for users configuration files, it is better not to set the library path, because the system's configuration has certainly been built to cope with your system's peculiarities. Use '`AppendLibraryPath:`' and '`PrependLibraryPath:`'.

4.3 Your Default Options

`Options:` *`options+`* [Configuration Setting]

Give `a2ps` a list of command line options. *options+* is any sequence of regular command line options (see Chapter 3 [Invoking a2ps], page 11).

It is the correct way to define the default behavior you expect from `a2ps`. If for instance you want to use `Letter` as medium, then use:

```
Options: --medium=Letter
```

It is exactly the same as always giving `a2ps` the option '`--medium=Letter`' at run time.

The quoting mechanism is the same as that of a shell. For instance

```
Options: --right-title="Page $p" --center-title="Hello World!"
Options: --title="arg 'Jack said \\\"hi\\\"' has double quotes"
```

4.4 Your Media

`Medium:` *`name dimensions`* [Configuration Setting]

Define the medium *name* to have the *dimensions* (in PostScript points, i.e., 1/72 of inch).

There are two formats supported:

long in which you must give both the size of the whole sheet, and the size of the printable area:

```
# A4 for HP DeskJets
#      name     w      h     llx   lly   urx    ury
Medium: A4dj    595    842    24    50    571    818
```

where *w*x*h* are the dimension of the sheet, and the four other stand for lower left x and y, upper right x and y.

short in which a surrounding margin of 24 points is used

```
# A4
#      name     w      h
Medium: A4      595    842
```

is the same as

```
# A4
#      name     w      h
Medium: A4      595    842    24    24    571    818
```

4.5 Your Printers

A general scheme is used, so that whatever the way you should address the printers on your system, the interface is still the same. Actually, the interface is so flexible, that you should understand 'named destination' when we write 'printer'.

`Printer:` *`name PPD-key destination`* [Configuration Setting]
`Printer:` *`name destination`* [Configuration Setting]
`Printer:` *`name PPD-key`* [Configuration Setting]
Specify the destination of the output when the option '`-P` *`name`*' is given. If *PPD-key* is given, declare the printer *name* to be described by the PPD file *`PPD-key`*`.ppd`. If *destination* is not given, used that of the '`UnknownPrinter:`'.

The *destination* must be of one of the following forms:

'`|` *`command`*'
in which case the output is piped into *command*.

'`>` *`file`*' in which case the output is saved into *file*.

`UnknownPrinter:` [*`PPD-key`*] *`destination`* [Configuration Setting]
Specify the destination of the output when when the option '`-P` *`name`*' is given, but there is no '`Printer:`' entry for *name*.

`DefaultPrinter:` [*`PPD-key`*] *`destination`* [Configuration Setting]
Specify the destination of the output when when the option '`-d`' (send to default output) is given.

Escapes expansion is performed on *destination* (see Section 3.2 [Escapes], page 24). Recall that '`#o`' is evaluated to the destination name, i.e., the argument given to '`-P`'.

For instance

```
# My Default Printer is called dominique
DefaultPrinter: | lp -d dominique

# 'a2ps foo.c -P bar' will pipe into 'lp -d bar'
UnknownPrinter: | lp -d #o

# 'a2ps -P foo' saves into the file 'foo'
Printer: foo > foo.ps
Printer: wc | wc
Printer: lw | lp -d printer-with-a-rather-big-name

# E.g. 'a2ps foo.c bar.h -P file' will save into 'foo.c.ps'
Printer: file > $n.#.

# E.g. 'a2ps foo.c bar.h -P home' will save into 'foo.ps'
# in user's home
Printer: home > ${HOME}/$N.#.
```

4.6 Your Shortcuts

You can define some kind of 'Macro Options' which stand for a set of options.

`UserOption:` *`shortcut options...`* [Configuration Setting]
Define the *shortcut* to be the list of *options...*. When `a2ps` is called with '`-=`*`shortcut`*' (or '`--user-option=`*`shortcut`*'), consider the list of *options...*.

Examples are

```
# This emulates a line printer: no features at all
# call a2ps -=lp to use it
UserOption: lp -1m --pretty-print=plain -B --borders=no

# When printing mail, I want to use the right style sheet with strong
# highlight level, and stripping 'useless' headers.
UserOption: mail -Email -g --strip=1
```

4.7 Your PostScript magic number

`a2ps` produces full DSC conformant PostScript (see Appendix A [Glossary], page 92). Adobe said

> Thou shalt start your PostScript DSC conformant files with
>
> `%!PS-Adobe-3.0`

The bad news is that some printers will reject this header. Then you may change this header without any worry since the PostScript produced by `a2ps` is also 100% PostScript level 1[1].

`OutputFirstLine:` *`magic-number`* [Configuration Setting]
: Specify the header of the produced PostScript file to be *magic-number*. Typical values include '`%!PS-Adobe-2.0`', or just '`%!`'.

4.8 Your Page Labels

In the PostScript file is dropped information on where sheets begin and end, so that post processing tools know where is the physical page 1, 2 etc. With this information can be also stored a label, i.e., a human readable text (typically the logical page numbers), which is for instance what `Ghostview` shows as the list of page numbers.

`a2ps` lets you define what you want in this field.

`PageLabelFormat:` *`format`* [Configuration Setting]
: Specify the *format* to use to label the PostScript pages. *format* can use Escapes (see Section 3.2 [Escapes], page 24). Two variables are predefined for this: '`#{pl.short}`' and '`#{pl.long}`'.

4.9 Your Variables

There are many places in `a2ps` where one would like to have uniform way of extending things. It once became clear that *variables* where needed in `a2ps`.

4.9.1 Defining Variables

`Variable:` *`key value`* [Configuration Setting]
: Define the escape '`#{`*`key`*`}`' to be a short cut for *value*. *key* must not have any character from '`:(){}`'.

[1] That is to say, there are no PostScript printers that don't understand these files.

As as example, here is a variable for `psnup`, which encloses all the option passing one would like. Delegations are then easier to write:

```
Variable: psnup psnup -#v -q #?j|-d|| #?r||-c| -w#w -h#h
```

It is strongly suggested to follow a '`.`' (dot) separated hierarchy, starting with:

'`del`' for variables that are related to delegations.

'`pro`' for variables used in prologues (see Section 8.6 [Designing PostScript Prologues], page 80). Please, specify the name of the prologue (e.g., '`pro.matrix.gray`').

'`ps`' for variables related to PostScript matters, such as the page label (which is associated to `ps.page_label`), the header etc.

'`pl`' for page label formats. See Section 4.8 [Your Page Labels], page 33, the option '`--page-label`' in Section 3.1.6 [Input Options], page 18.

'`toc`' for toc formats. See the option '`--toc`' in Section 3.1.6 [Input Options], page 18.

'`user`' for user related information. See Section 4.9.2 [Predefined Variables], page 34.

This naming convention has not fully stabilized. We apologize for the inconvenience this might cause to users.

4.9.2 Predefined Variables

There are a few predefined variables. The fact that `a2ps` builds them at startup changes nothing to their status: they can be modified like any other variable using `--define` (see Section 3.1.2 [Global Options], page 13).

In what follows, there are numbers (i) like this, or (ii) this. It means that `a2ps` first tries the solution (i), if a result is obtained (non empty value), this is the value given to the variable. Otherwise it tries solution (ii), etc. The rationale behind the order is usually from user modifiable values (e.g. environment variables) through system's hard coded values (e.g., calls to `getpwuid`) and finally arbitrary values.

'`user.comments`'
: Comments on the user. Computed by (i) the system's database (the part of `pw_gecos` after the first '`,`'), (ii) not defined.

'`user.home`'
: The user's home directory. Determined by (i) the environment variable `HOME`, (ii) the system's database (using `getpwuid`), (iii) the empty string.

'`user.host`'
: The user's host name. Assigned from (i) the system (`gethostname` or `uname`), (ii) the empty string.

'`user.login`'
: The user's login (e.g. '`bgates`'). Computed by (i) the environment variable `LOGNAME`, (ii) the environment variable `USERNAME`, (iii) the system's database (using `getpwuid`), (iv) the translated string '`user`'.

'`user.name`'
: The user's name (e.g. '`William Gates`'). Computed by (i) the system's database (`pw_gecos` up to the first '`,`'), (ii) capitalized value of the variable

'`user.login`' unless it was the translated string '`user`', (iii) the translated string '`Unknown User`'.

4.10 Your Delegations

There are some files you don't really want `a2ps` to pretty-print, typically page description files (e.g., PostScript files, `roff` files, etc.). You can let `a2ps` delegate the treatment of these files to other applications. The behavior at run time depends upon the option '`--delegate`' (see Section 3.1.6 [Input Options], page 18).

4.10.1 Defining a Delegation

`Delegation:` *name in:out command* [Configuration Setting]

Define the delegation *name*. It is to be applied upon files of type *in* when output type is *out*[2] thanks to *command*. Both *in* and *out* are `a2ps` type keys such as defined in `sheets.map` (see Section 7.7.3 [The Entry in sheets.map], page 76).

command should produce the file on its standard output. Of course escapes substitution is performed on *command* (see Section 3.2 [Escapes], page 24). In particular, *command* should use the input file '`$f`'.

```
# In general, people don't want to pretty-print PostScript files.
# Pass the PostScript files to psnup
Delegation: PsNup ps:ps \
        psselect #?V||-q| -p#?p|#p|-| $f | \
        psnup -#v -q #?j|-d|| #?r||-c| -w#w -h#h
```

Advantage should be taken from the variables, to encapsulate the peculiarities of the various programs.

```
# Passes the options to psnup.
# The files (in and out) are to be given
Variable: psnup psnup -#v #?V||-q| #?j|-d|| #?r||-c| -w#w -h#h

# Passes to psselect for PS page selection
Variable: psselect psselect #?V||-q| -p#?p|#p|-|

# In general, people don't want to pretty-print PostScript files.
# Pass the PostScript files to psnup
Delegation: PsNup ps:ps     #{psselect} $f | #{psnup}
```

Temporary file names ('`#f0`' to '`#f9`') are available for complex commands.

```
# Pass DVI files to dvips.
# A problem with dvips is that even on failure it dumps its prologue,
# hence it looks like a success (output is produced).
# To avoid that, we use an auxiliary file and a conditional call to
# psnup instead of piping.
Delegation: dvips dvi:ps    #{dvips} $f -o #f0 && #{psnup} #f0
```

[2] Current `a2ps` only handles PostScript output, i.e. *out*='`ps`'

4.10.2 Guide Line for Delegations

First of all, select carefully the applications you will use for the delegations. If a filter is known to cause problems, try to avoid it in delegations[3]. As a thumb rule, you should check that the PostScript generating applications produce files that start by:

```
%!PS-Adobe-3.0
```

`a2ps` **needs** the '`%%BeginSetup`'-'`%%EndSetup`' section in order to output correctly the page device definitions. It can happen that your filters don't output this section. In that case, you should insert a call to `fixps` right after the PostScript generation:

```
########## ROFF files
# Pass the roff files to groff.  Ask grog how groff should be called.
# Use fixps to ensure there is a %%BeginSetup/%%EndSetup section.
Delegation: Groff roff:ps \
   eval `grog -Tps '$f'` | fixps #?V!!-q! | #{d.psselect} | #{d.psnup}
```

There are some services expected from the delegations. The delegations you may write should honor:

the input file
: available via the escape '`$f`'. You should be aware that there are people who have great fun having spaces or dollars in their file names, so you probably should always use '`'$f'`'. Some other variables are affected. Yes, I know, we need a special mechanism for '`'`' itself. Well, we'll see that later '`;-)`'.

the medium
: the dimension of the medium selected by the user are available through '`#w`' and '`#h`'.

the page layout
: the number of virtual pages is '`#v`'.

the page range
: the page range (in a form '`1-2,4-6,10-`' for instance) is available by '`#p`'.

the verbosity level
: please, do not make your delegations verbose by default. The silent mode should always be requested, unless '`#?V`' is set (see the above example with `groff`).

If ever you need several commands, do not use '`;`' to separate them, since it may prevent detection of failure. Use '`&&`' instead.

The slogan "*the sooner, the better*" should be applied here: in the processing chain, it is better to ask a service to the first application that supports it. An example will make it clear: when processing a `DVI` file, `dvips` knows better the page numbers than `psselect` would. So a `DVI` to PostScript delegation should ask the page selection ('`#p`') to `dvips`, instead of using `psselect` later in the chain. An other obvious reason here is plain efficiency (globally, less data is processed).

[3] Because hiding its use into `a2ps` just makes it even more difficult to the users to know why it failed. Let them use it by hand.

4.10.3 Predefined Delegations

The purpose of this section is not to document all the predefined delegations, for this you should read the comments in the system configuration file `a2ps.cfg`. We just want to explain some choices, and give hints on how to make the best use of these delegations.

`dvips` (*DVI to PostScript*) [Delegation]

There is a problem when you use a naive implementation of this delegation: landscape jobs are not recognized, and therefore n-upping generally fails miserably. Therefore, `a2ps` tries to guess if the file is landscape by looking for the keyword '`landscape`' in it, using `strings(1)`:

```
Delegation: dvips dvi:ps\
 if strings $f | sed 3q | grep -F landscape > /dev/null 2>&1; then \
   #{d.dvips} -T#hpt,#wpt $f -o #f0 && #?o|cat|#{d.psnup} -r| #f0;\
 else \
   #{d.dvips} $f -o #f0 && #{d.psnup} #f0; \
 fi
```

In order to have that rule work correctly, it is expected from the TEX, or LATEX file to include something like:

```
\renewcommand{\printlandscape}{\special{landscape}}
\printlandscape
```

in the preamble.

We don't use a pipe because dvips always outputs data (its prologue) even if it fails, what prevents error detection.

`LaTeX` (*LATEX to DVI*) [Delegation]

We use a modern version of the shell script `texi2dvi`, from the package `Texinfo`, which runs `makeindex`, `bibtex` and `latex` as many times as needed. You should be aware that if the file includes files from **other** directories, it may miss some compilation steps. Other cases (most typical) are well handled.

4.11 Your Internal Details

There are settings that only meant for `a2ps` that you can tune by yourself.

`FileCommand:` *`command`* [Configuration Setting]

The command to run to call `file(1)` on a file. If possible, make it follow the symbolic links.

5 Library Files

To be general and to allow as much customization as possible, `a2ps` avoids to hard code its knowledge (encodings, PostScript routines, etc.), and tries to split it in various files. Hence it needs a path, i.e., a list of directories, in which it may find the files it needs.

The exact value of this library path is available by '`a2ps --list=defaults`'. Typically its value is:

```
gargantua ~ $ a2ps --list=defaults
Configuration status of a2ps 4.15.6
More stuff deleted here
Internals:
  verbosity level     = 2
  file command        = /usr/ucb/file -L
  temporary directory =
  library path        =
        /inf/soft/infthes/demaille/.a2ps
        /usr/local/share/a2ps/sheets
        /usr/local/share/a2ps/ps
        /usr/local/share/a2ps/encoding
        /usr/local/share/a2ps/afm
        /usr/local/share/a2ps/printers
        /usr/local/share/a2ps
```

You may change this default path through the configuration files (see Section 4.2 [Your Library Path], page 30).

If you plan to define yourself some files for `a2ps`, they should be in one of those directories.

5.1 Documentation Format

In various places a documentation can be given. Since some parts of this document and of web pages are extracted from documentations, some tags are needed to provide a better layout. The format is a mixture made out of Texinfo like commands, but built so that quick and easy processing can be made.

These tags are:

'`code('`*text*'`)code`'
: Typeset *text* like a piece of code. This should be used for keys, variables, options etc. For instance the documentation of the `bold` prologue mentions the `bw` prologue:

```
Documentation
This style is meant to replace the old option
code(-b)code of a2ps 4.3.  It is a copy of the
black and white prologue, but in which all the
fonts are in Bold.
EndDocumentation
```

‘`href('`*link*‘`)href('`*text*‘`)href`’
: Specifies a hyper text *link* displayed as *text*.

‘`@example`’
‘`@end example`’
: They must be alone on the line. The text between these tags is displayed in a code-like fonts. This should be used for including a piece of code. For instance, in the documentation of the `gnuc` style sheet:

```
documentation is
 "Declaration of functions are highlighted"
 "emph(only)emph if you start the function name"
 "in the first column, and it is followed by an"
 "opening parenthesis.  In other words, if you"
 "write"
 "@example"
 "int main (void)"
 "@end example"
 "it won't work.  Write:"
 "@example"
 "int"
 "main (void)"
 "@end example"
end documentation
```

‘`@itemize`’
‘`@item` *text*
‘`@end itemize`’
: Typeset a list of items. The opening and closing tags must be alone on the line.

5.2 Map Files

Many things are defined through files. There is a general scheme to associate an object to the files to use: map files. They are typically used to:

- resolve aliases. For instance the ISO-8859-1 encoding is also called ISO Latin 1, or Latin 1 for short. The `encoding.map` file will map these three names to the same Encoding Description File.
- cope with broken files systems. For instance, the-one-you-know-I-don't-need-to-name cannot handle files named `Courier-BoldOblique.afm`: it is the same as `Courier-Bold.afm`. The `fonts.map` file is here to associate a font file name to a font name.

The syntax of these files is:

- any empty line, or any line starting by a ‘`#`’ is a comment.
- a line with the format

  `***` *path*

  requests that the file designated by *path* be included at this point.

– any other line has the format

```
key                 value
```

meaning that when looking for *key* (e.g., name of a font, an encoding etc.), `a2ps` should use *value* (e.g., font file name, encoding description file name etc.).

The map files used in `a2ps` are:

`encoding.map`
: Resolving encodings aliases.

`fonts.map`
: Mapping font names to font file names.

`sheets.map`
: Rules to decide what style sheet to use.

5.3 Font Files

Even when a PostScript printer knows the fonts you want to use, using these fonts requires some description files.

5.3.1 Fonts Map File

See Section 5.2 [Map Files], page 39, for a description of the map files. This file associates the *font-key* to a *font* name. For instance:

```
Courier                 pcrr
Courier-Bold            pcrb
Courier-BoldOblique     pcrbo
Courier-Oblique         pcrro
```

associates to font named `Courier`, the key `pcrr`. To be recognized, the font name must be exact: `courier` and `COURIER` are not admitted.

5.3.2 Fonts Description Files

There are two kinds of data `a2ps` needs to use a font:

– the AFM file (*`font-key`*`.afm`), which describes the metrics information corresponding to *font*;

– in the case *font* is not known from the printer, the PFA or PFB file which is down loaded to the printer. These files are actually the PostScript programs which execution produces the characters to be drawn on the page, in this *font*.

5.3.3 Adding More Font Support

`a2ps` can use as many fonts as you want, provided that you teach it the name of the files in which are stored the fonts (see Section 5.3.1 [Fonts Map File], page 40). To this end, a very primitive but still useful shell script is provided: `make_fonts_map.sh`.

First, you need to find the directories which store the fonts you want to use, and extend the library path so that `a2ps` sees those directories. For instance, add:

```
AppendLibraryPath: /usr/local/share/ghostscript/fonts
```

Then run `make_fonts_map.sh`. It should be located in the `afm/` directory of the system's `a2ps` hierarchy. Typically `/usr/local/share/a2ps/afm/make_fonts_map.sh`.

This script asks `a2ps` for the library path, wanders in this path collecting `AFM` files, and digging information in them.

Once the script has finished, a file `fonts.map.new` was created. Check its integrity, and if it's correct, either replace the old `fonts.map` with it, or rename `fonts.map.new` as `fonts.map` and place it higher in the library path (for instance in your `~/.a2ps/` directory).

5.4 Style Sheet Files

The style sheets are defined in various files (see Chapter 7 [Pretty Printing], page 48, for the structure of these files). As for most other features, there is main file, a road map, which defines in which condition a style sheet should be used (see Section 5.2 [Map Files], page 39). This file is `sheets.map`.

Its format is simple:

style-key`:` *patterns*

or

`include(`*file*`)`

The *patterns* need not be on separate lines. There are two kinds of patterns:

/pattern/flags
: if the current file name matches *pattern*, then select style *style-key* (i.e. file *style-key*`.ssh`).

<pattern>flags
: if the result of a call to `file(1)` matches *pattern*, then select style *style-key*.

Currently *flags* can only be '`i`', standing for an insentive match. Please note that the matching is not truly case insensitive: rather, a lower case version of the string is compared to the *pattern* as is, i.e., the *pattern* should itself be lower case.

The special *style-key* '`binary`' tells `a2ps` to consider that the file should not be printed, and will be ignored, unless option '`--print-anyway`' is given.

If a style name can't be found, the plain style is used.

The map file is read bottom up, so that the "last" match is honored.

Two things are to retain from this:

1. if the file is presented through `stdin`, then `a2ps` will run `file(1)`. However, unless you specify a fake file name with '`--stdin`', pattern matching upon the name is turn off. In general you can expect correct delegations, but almost never pretty printing.
2. if `file` is wrong on some files, `a2ps` may use bad style sheets. In this case, do try option '`--guess`', compare it with the output of `file`, and if the culprit is `file`, go and complain to your system administrator :-), or fix it by defining your own filename pattern matching rules.

Consider the case of Texinfo files as an example (the language in which this documentation is written). Files are usually named `foo.texi`, `bar.txi`, or even `baz.texinfo`. `file(1)` is able to recognize Texinfo files:

```
doc % file a2ps.texi
a2ps.texi: Texinfo source text
```

Therefore the sheets.map would look like:

```
# Texinfo files
texinfo:  /*.txi/  /*.texi/  /*.texinfo/
          <Texinfo source*>
```

6 Encodings

`a2ps` is trying to support the various usual encodings that its users use. This chapter presents what an encoding is, how the encodings support is handled within `a2ps`, and some encodings it supports.

6.1 What is an Encoding

This section was taken from the web pages of Alis Technologies, Inc., now Open Text Corporation[1].

Document encoding is the most important but also the most sensitive and explosive topic in Internet internationalization. It is an essential factor since most of the information distributed over the Internet is in text format. But the history of the Internet is such that the predominant - and in some cases the only possible - encoding is the very limited ASCII, which can represent only a handful of languages, only three of which are used to any great extent: English, Indonesian and Swahili.

All the other languages, spoken by more than 90% of the world's population, must fall back on other character sets. And there is a plethora of them, created over the years to satisfy writing constraints and constantly changing technological limitations. The ISO international character set registry contains only a small fraction; IBM's character registry is over three centimeters thick; Microsoft and Apple each have a bunch of their own, as do other software manufacturers and editors.

The problem is not that there are too few but rather too many choices, at least whenever Internet standards allow them. And the surplus is a real problem; if every Arabic user made his own choice among the three dozen or so codes available for this language, there is little likelihood that his "neighbor" would do the same and that they would thus be able to understand each other. This example is rather extreme, but it does illustrate the importance of standards in the area of internationalization. For a group of users sharing the same language to be able to communicate,

1. the code used in the shared document must always be identified (labeling)
2. they must agree on a small number of codes - only one, if possible (standards);
3. their software must recognize and process all codes (versatility)

Certain character sets stand out either because of their status as an official national or international standard, or simply because of their widespread use.

First off, there is the ISO 8859 standards series that standardize a dozen character sets that are useful for a large number of languages using the Latin, Cyrillic, Arabic, Greek and Hebrew alphabets. These standards have a limited range of application (8 bits per character, a maximum of 190 characters, no combining) but where they suffice (as they do for 10 of the 20 most widely used languages), they should be used on the Internet in preference to other codes. For all other languages, national standards should preferably be chosen or, if none are available, a well-known and widely-used code should be the second choice.

Even when we limit ourselves to the most widely used standards, the overabundance remains considerable, and this significantly complicates life for truly international software

[1] `https://www.opentext.com/`

developers and users of several languages, especially when such languages can only be represented by a single code. It was to resolve this problem that both Unicode and the ISO 10646 International standard were created. Two standards? Oh no! Their designers soon realized the problem and were able to cooperate to the extent of making the character set *repertoires* and coding identical.

ISO 10646 (and Unicode) contain over 30,000 characters capable of representing most of the living languages within a single code. All of these characters, except for the *Han* (Chinese characters also used in Japanese and Korean), have a name. And there is still room to encode the missing languages as soon as enough of the necessary research is done. Unicode can be used to represent several languages, using different alphabets, within the same electronic document.

6.2 Encoding Files

The support of the encodings in `a2ps` is completely taken out of the code. That is to say, adding, removing or changing anything in its support for an encoding does not require programming, nor even being a programmer.

See Section 6.1 [What is an Encoding], page 43, if you want to know more about this.

6.2.1 Encoding Map File

See Section 5.2 [Map Files], page 39, for a description of the map files.

The meaningful lines of the `encoding.map` file have the form:

```
alias      key
iso-8859-1 latin1
latin1     latin1
l1         latin1
```

where

alias specifies any name under which the encoding may be used. It influences the option '`--encoding`', but also the encodings dynamically required, as for instance in the `mail` style sheet (support for MIME).

When *encoding* is asked, the lower case version of *encoding* must be equal to *alias*.

key specifies the prefix of the file describing the encoding (`key.edf`, Section 6.2.2 [Encoding Description Files], page 44).

6.2.2 Encoding Description Files

The encoding description file describing the encoding *key* is named `key.edf`. It is subject to the same rules as any other `a2ps` file:

- please make the name portable: alpha-numerical, at most 8 characters,
- empty lines and lines starting by '`#`' are ignored.

The entries are

'`Name:`' Specifies the full name of the encoding. Please, try to use the official name if there is one.

```
Name: ISO-8859-1
```

'`Documentation/EndDocumentation`'
: Introduces the documentation on the encoding (see Section 5.1 [Documentation Format], page 38). Typical informations expected are the other important names this encoding has, and the languages it covers.

```
Documentation
Also known as ISO Latin 1, or Latin 1.  It is a superset
of ASCII, and covers most West-European languages.
EndDocumentation
```

'`Substitute:`'
: Introduces a font substitution. The most common fonts (e.g., `Courier`, `Times-Roman`...) do not support many encodings (for instance it does not support Latin 2). To avoid that Latin 2 users have to replace everywhere calls to `Courier`, `a2ps` allows to specify that whenever a font is called in an encoding, then another font should be used.

  For instance in `iso2.edf` one can read:

```
# Fonts from Ogonkify offer full support of ISO Latin 2
Substitute: Courier              Courier-Ogonki
Substitute: Courier-Bold         Courier-Bold-Ogonki
Substitute: Courier-BoldOblique  Courier-BoldOblique-Ogonki
Substitute: Courier-Oblique      Courier-Oblique-Ogonki
```

'`Default:`'
: Introduces the name of the font that should be used when a font (not substituted as per the previous item) is called but provides to poor a support of the encoding. The `Courier` equivalent is the best choice.

```
Default: Courier-Ogonki
```

'`Vector:`' Introduces the PostScript encoding vector, that is a list of the 256 PostScript names of the characters. Note that only the printable characters are named in PostScript (e.g., '`bell`' in ASCII (`^G`) should not be named). The special name '`.notdef`' is to be used when the character is not printable.

  Warning. Make sure to use real, official, PostScript names. Using names such as '`c123`' may be the sign you use unusual names. On the other hand PostScript names such as '`afii8879`' are common.

6.2.3 Some Encodings

Most of the following information is a courtesy of Alis Technologies, Inc. and of Roman Czyborra's page about The ISO 8859 Alphabet Soup[2]. See Section 6.1 [What is an Encoding], page 43, is an instructive presentation of the encodings.

The known encodings are:

`ASCII (ascii.edf)` [Encoding]
: US-ASCII.

`EUC-JP (euc-jp.edf)` [Encoding]
: The EUC-JP encoding is a 8-bit character set widely used in Japan.

[2] `http://czyborra.com/charsets/`

`HPRoman (hp.edf)` [Encoding]
The 8 bits Roman encoding for HP.

`IBM-CP437 (ibm-cp437.edf)` [Encoding]
This encoding is meant to be used for PC files with drawing lines.

`IBM-CP850 (ibm-cp850.edf)` [Encoding]
Several characters may be missing, especially Greek letters and some mathematical symbols.

`ISO-8859-1 (iso1.edf)` [Encoding]
The ISO-8859-1 character set, often simply referred to as Latin 1, covers most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Rhaeto-Romanic, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English, incidentally also Afrikaans and Swahili, thus in effect also the entire American continent, Australia and the southern two-thirds of Africa. The lack of the ligatures Dutch IJ, French OE and ,,German“ quotation marks is considered tolerable.

The lack of the new C=-resembling Euro currency symbol U+20AC has opened the discussion of a new Latin0.

`ISO-8859-2 (iso2.edf)` [Encoding]
The Latin 2 character set supports the Slavic languages of Central Europe which use the Latin alphabet. The ISO-8859-2 set is used for the following languages: Czech, Croat, German, Hungarian, Polish, Romanian, Slovak and Slovenian.

Support is provided thanks to Ogonkify.

`ISO-8859-3 (iso3.edf)` [Encoding]
This character set is used for Esperanto, Galician, Maltese and Turkish.

Support is provided thanks to Ogonkify.

`ISO-8859-4 (iso4.edf)` [Encoding]
Some letters were added to the ISO-8859-4 to support languages such as Estonian, Latvian and Lithuanian. It is an incomplete precursor of the Latin 6 set.

Support is provided thanks to Ogonkify.

`ISO-8859-5 (iso5.edf)` [Encoding]
The ISO-8859-5 set is used for various forms of the Cyrillic alphabet. It supports Bulgarian, Byelorussian, Macedonian, Serbian and Ukrainian.

The Cyrillic alphabet was created by St. Cyril in the 9th century from the upper case letters of the Greek alphabet. The more ancient Glagolithic (from the ancient Slav glagol, which means "word"), was created for certain dialects from the lower case Greek letters. These characters are still used by Dalmatian Catholics in their liturgical books. The kings of France were sworn in at Reims using a Gospel in Glagolithic characters attributed to St. Jerome.

Note that Russians seem to prefer the KOI8-R character set to the ISO set for computer purposes. KOI8-R is composed using the lower half (the first 128 characters) of the corresponding American ASCII character set.

`ISO-8859-7 (iso7.edf)` [Encoding]
ISO-8859-7 was formerly known as ELOT-928 or ECMA-118:1986. It is meant for modern Greek.

`ISO-8859-9 (iso9.edf)` [Encoding]
The ISO 8859-9 set, or Latin 5, replaces the rarely used Icelandic letters from Latin 1 with Turkish letters.

Support is provided thanks to Ogonkify.

`ISO-8859-10 (iso10.edf)` [Encoding]
Latin 6 (or ISO-8859-10) adds the last letters from Greenlandic and Lapp which were missing in Latin 4, and thereby covers all Scandinavia.

Support is provided thanks to Ogonkify.

`ISO-8859-13 (iso13.edf)` [Encoding]
Latin7 (ISO-8859-13) is going to cover the Baltic Rim and re-establish the Latvian (lv) support lost in Latin6 and may introduce the local quotation marks.

Support is provided thanks to Ogonkify.

`ISO-8859-15 (iso15.edf)` [Encoding]
The new Latin9 nicknamed Latin0 aims to update Latin1 by replacing some less needed symbols (some fractions and accents) with forgotten French and Finnish letters and placing the U+20AC Euro sign in the cell of the former international currency sign.

Support of the Euro symbol is provided thanks to Ogonkify.

`KOI8 (koi8.edf)` [Encoding]
KOI-8 is a subset of ISO-IR-111 that can be used in Serbia, Belarus etc.

`MS-CP1250 (ms-cp1250.edf)` [Encoding]
Microsoft's CP-1250 encoding (aka CeP).

`MS-CP1251 (ms-cp1251.edf)` [Encoding]
Microsoft CP1251 is encoding used in Microsoft Windows for Cyrillic languages

`Macintosh (mac.edf)` [Encoding]
For the Macintosh encoding. The support is not sufficient, and a lot of characters may be missing at the end of the job (especially Greek letters).

7 Pretty Printing

The main feature of `a2ps` is its pretty-printing capabilities. Two different levels of pretty printing can be reached:

- basic (normal highlight level) in which what you print is what you wrote.
- string (heavy highlight level), in which in general, some keywords are replaced by a Symbol character which best represents them. For instance, in most languages '`<=`' and '`>=`' will be replaced by the corresponding single character from the font Symbol.

Note that the difference is up to the author of the style sheet.

7.1 Syntactic limits

`a2ps` is *not* a powerful syntactic pretty-printer: it just handles lexical structures, i.e., if in your favorite language

```
IF IF == THEN THEN THEN := ELSE ELSE ELSE := IF
```

is legal, then `a2ps` is not the tool you need. Indeed `a2ps` just looks for some keywords, or some *sequences*.

7.2 Known Style Sheets

`68000 (68000.ssh)` [Style Sheet]
Written by Akim Demaille. Althought designed at the origin for the 68k's assembler, this style sheet seems to handle rather well other dialects.

`a2ps configuration file (a2psrc.ssh)` [Style Sheet]
Written by Akim Demaille. Meant to print files such as '`a2ps.cfg`', or '`.a2ps/a2psrc`', etc.

`a2ps style sheet (ssh.ssh)` [Style Sheet]
Written by Akim Demaille. Second level of highligthing (option '`-g`')) substitutes the LaTeX symbols.

`Ada (ada.ssh)` [Style Sheet]
Written by Akim Demaille. This style sheets cover Ada 95. If you feel the need for Ada 83, you'll have to design another style sheet.

`ASN.1 (asn1.ssh)` [Style Sheet]
Written by Philippe Coucaud. ASN.1 (Abstract Syntax Notation One) is used to define the protocol data units (PDUs) of all application layer protocols to date.

`Autoconf (autoconf.ssh)` [Style Sheet]
Written by Akim Demaille. Suitable for both `configure.ac` and library `m4` files.

`AWK (awk.ssh)` [Style Sheet]
Written by Edward Arthur. This style is devoted to the AWK pattern scanning and processing language. It is supposed to support classic awk, nawk and gawk.

`B (b.ssh)` [Style Sheet]
Written by Philippe Coucaud. B is a formal specification method mostly used to describe critical systems. It is based on the mathematical sets theory.

`BC (bc.ssh)` [Style Sheet]
Written by Akim Demaille. bc is an arbitrary precision calculator language.

`Bourne Shell (sh.ssh)` [Style Sheet]
Written by Akim Demaille. Some classical program names, or builtin, are highlighted in the second level of pretty-printing.

`C (c.ssh)` [Style Sheet]
Written by Akim Demaille. This style does not highlight the function definitions. Another style which highlights them, GNUish C, is provided (gnuc.ssh). It works only if you respect some syntactic conventions.

`C Shell (csh.ssh)` [Style Sheet]
Written by Jim Diamond. Some classical program names, and/or builtins, are highlighted in the second level of pretty-printing.

`C# (csharp.ssh)` [Style Sheet]
Written by Karen Christenson. This style is for the .NET object-oriented language C#, and is based on the C# Language Specification published in 2002 by Microsoft in the MSDN library. XML comments are mapped to strong comments, and any other comment is a plain comment. The C style-sheet was not selected as an ancestor in order to treat a struct the same as a class or an interface. The CPP style-sheet was not selected as an ancestor because C# set of preprocessor directives is much smaller. Keywords, XML comments, preprocessor directives, label statements, and [] style attributes are high-lighted.

`C++ (cxx.ssh)` [Style Sheet]
Written by Akim Demaille. Should handle all known variations of C++. Most declarations (classes etc.) are not highlighted as they should be. Please, step forward!

`CAML (caml.ssh)` [Style Sheet]
This style is obsolete: use OCaml instead.

`ChangeLog (chlog.ssh)` [Style Sheet]
Written by Akim Demaille. This style covers the usual ChangeLog files.

`Claire (claire.ssh)` [Style Sheet]
Written by Akim Demaille. Claire is a high-level functional and object-oriented language with advanced rule processing capabilities. It is intended to allow the programmer to express complex algorithms with fewer lines and in an elegant and readable manner.

To provide a high degree of expressivity, Claire uses:

- A very rich type system including type intervals and second-order types (with dual static/dynamic typing),
- Parametric classes and methods,

- An object-oriented logic with set extensions,
- Dynamic versioning that supports easy exploration of search spaces.

To achieve its goal of readability, Claire uses

- set-based programming with an intuitive syntax,
- simple-minded object-oriented programming,
- truly polymorphic and parametric functional programming,
- a powerful-yet-readable extension of DATALOG to express logical conditions,
- an entity-relation approach with explicit relations, inverses, unknown values and relational
- operations.

More information on claire can be found on Wikipedia[1].

`Common Lisp (clisp.ssh)` [Style Sheet]
Written by Juliusz Chroboczek. It is not very clear what should be considered as a 'keyword' in Common Lisp. I like binders, control structures and declarations to be highlighted, but not assignments.

Names of defstructs are not highlighted because this would not work with defstruct options.

`Coq Vernacular (coqv.ssh)` [Style Sheet]
Written by Akim Demaille. This style is devoted to the Coq v 5.10 vernacular language.

`CORBA IDL (cidl.ssh)` [Style Sheet]
Written by Bob Phillips. A first attempt at a style sheet for OMG CORBA IDL. I believe I captured all the keywords for CORBA 2.2 IDL. I also stole code from gnuc.ssh to print the method names in bold face. I'm not sure I quite like my own choices for Keyword_strong and Keyword, so I'm looking for feedback. Note that, as with gnuc.ssh, for a method name to be noted as such, the left parenthesis associated with the argument list for the method must appear on the same line as the method name.

`CPP (cpp.ssh)` [Style Sheet]
Written by Akim Demaille. C traditional preprocessor handling, mostly meant to be inherited.

`dc_shell (dc_shell.ssh)` [Style Sheet]
Written by Philippe Le Van. Synopsys Design Compiler is a synthesis tool used by electronic companies for the design of their chips. This sheet is very incomplete, we have a lot of keywords to add, eventually options to highlight... The Label_strong style is used for commands which change the design.

[1] `https://en.wikipedia.org/wiki/Claire_(programming_language)`

`Eiffel (eiffel.ssh)` [Style Sheet]
Written by Akim Demaille. Eiffel is an object oriented language that also includes a comprehensive approach to software construction: a method.

The language itself is not just a programming language but also covers analysis, design and implementation.

Heavy highlight uses symbols to represent common math operators.

`Emacs Lisp (elisp.ssh)` [Style Sheet]
Written by Didier Verna. This style sheet includes support for some extensions dumped with XEmacs.

`Encapsulated PostScript (eps.ssh)` [Style Sheet]
Written by Akim Demaille. Illegal PostScript operators are highlighted as Errors.

`Extended Tcl (tclx.ssh)` [Style Sheet]
Written by Phil Hollenback. Extensions to plain Tcl.

`Fortran (fortran.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. There are several Fortran dialects, depending whether, on the one hand, you use Fortran 77 or Fortran 90/95, and, on the other hand, Fixed form comments, or Free form comments.

The style sheets `for77kwds` and `for90kwds` implements keywords only, while the style sheets `for-fixed` and `for-free` implements comments only.

This style sheet tries to support any of the various flavors (Fortran 77/90/95, fixed or free form). For more specific uses, you should use either:

- for77-fixed, for Fortran 77 fixed form,
- for77-free, for Fortran 77 free form,
- for90-fixed, for Fortran 90/95 fixed form,
- for90-free, for Fortran 90/95 free form.

`Fortran 77 Fixed (for77-fixed.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. Dedicated to Fortran 77 in fixed form, i.e., comments are lines starting with c, C, or *, and only those lines are comments.

`Fortran 77 Free (for77-free.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. Dedicated to Fortran 77 in free form, i.e., comments are introduced by ! anywhere on the line, and nothing else is a comment.

`Fortran 77 Keywords (for77kwds.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. This sheet implements only Fortran 77 keywords, and avoids implementing comments support. This is to allow for implementation of either fixed or free source form.

See the documentation of the style sheet `fortran` for more details.

`Fortran 90 Fixed (for90-fixed.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. Dedicated to Fortran 90/95 in fixed form, i.e., comments are lines starting with c, C, or *, and only those lines are comments.

`Fortran 90 Free (for90-free.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. Dedicated to Fortran 90/95 in free form, i.e., comments are introduced by ! anywhere on the line, and nothing else is a comment.

`Fortran 90 Keywords (for90kwds.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. This sheet implements the superset which Fortran 90 and Fortran 95 provide over Fortran 77.

See the documentation of the style sheet `fortran` for more details.

`Fortran Fixed (for-fixed.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. Implements comments of Fortran in fixed form, i.e., comments are lines starting with c, C, or *, and only those lines are comments. No other highlighting is done.

See the documentation of the style sheet `fortran` for more details.

`Fortran Free (for-free.ssh)` [Style Sheet]
Written by Denis Girou, Alexander Mai. Dedicated to Fortran in free form, i.e., comments are introduced by ! anywhere on the line, and nothing else is a comment.

`GNUish C (gnuc.ssh)` [Style Sheet]
Written by Akim Demaille. Declaration of functions are highlighted *only* if you start the function name in the first column, and it is followed by an opening parenthesis. In other words, if you write

```
int main (void)
```

it won't work. Write:

```
int
main (void)
```

`GNUMakefile (gmake.ssh)` [Style Sheet]
Written by Alexander Mai. Special tokens of GNUmakefiles and non terminal declarations are highlighted.

`Haskell (haskell.ssh)` [Style Sheet]
Written by Ilya Beylin. Haskell: non-strict functional programming language https://www.haskell.org/

`HTML (html.ssh)` [Style Sheet]
Written by Akim Demaille, Wesley J. Chun. This style is meant to pretty print HTML source files, not to simulate its interpretation (i.e., '`<bold>foo</bold>`' does not print '`foo`' in bold). If you really meant to print the result of the HTML file *interpreted*, then you should turn the delegations on, and make sure '`a2ps`' has HTML delegations.

`IDL (idl.ssh)` [Style Sheet]
Written by Robert S. Mallozzi, Manfred Schwarb. Style sheet for IDL 5.2 (Interactive Data Language). Obsolete routines are not supported. https://www.rsinc.com.

`InstallShield 5 (is5rul.ssh)` [Style Sheet]
Written by Alex. InstallShield5 _TM_ RUL script.

`Java (java.ssh)` [Style Sheet]
Written by Steve Alexander. Documentation comments are mapped to strong comments, and any other comment is plain comment.

`JavaScript (js.ssh)` [Style Sheet]
Written by Scott Pakin. Keywords used are everything listed in the Client-Side JavaScript Reference 1.3, plus `"undefined"` (why isn't that listed?) and `"prototype"`. I omitted the semi-standard a2ps optional operators for equality, because JavaScript's use of both strict- and non-strict equality might ambiguate the output. Finally, regular expressions are formatted like strings.

`LACE (lace.ssh)` [Style Sheet]
Written by Akim Demaille. This is meant for the Eiffel equivalent of the Makefiles.

`Lex (lex.ssh)` [Style Sheet]
Written by Akim Demaille. In addition to the C constructs, it highlights the declaration of states, and some special '`%`' commands.

`Lout (lout.ssh)` [Style Sheet]
Written by Jean-Baptiste Nivoit. This is the style for Lout files.

`Mail Folder (mail.ssh)` [Style Sheet]
Written by Akim Demaille. To use from elm and others, it is better to specify '`-g -Email`', since the file sent to printer is no longer truly a mail folder. This style also suits to news. '`--strip`' options are also useful (they strip `"useless"` headers).

Whenever the changes of encoding are clear, a2ps sets itself the encoding for the parts concerned.

Tag 1 is the subject, and Tag 2 the author of the mail/news.

Note: This style sheet is _very_ difficult to write. Please don't report behavior you don't like. Just send me improvements, or write a Bison parser for mails.

`Makefile (make.ssh)` [Style Sheet]
Written by Akim Demaille. Special tokens, and non terminal declarations are highlighted.

`Management Information Base (mib.ssh)` [Style Sheet]
Written by Kelly Wiles. The MIB file is of ASN.1 syntax.

`Maple (maple.ssh)` [Style Sheet]
Written by Richard J Mathar. Some classical program names, and/or builtins, are highlighted in the second level of pretty-printing.

`masm (nasm.ssh)` [Style Sheet]
Written by Aleksandar Veselinovic. This style highlights MASM ASM code.

`Matlab (matlab.ssh)` [Style Sheet]
Written by Joakim Lübeck. This style highlights function definitions and a limited number of keywords, mostly control constructs, and is therefore usable for many Matlab versions. Special care have been taken to distinguish string delimiters from the transpose operator (which is the same symbol) and to recognize comments.

`MATLAB 4 (matlab4.ssh)` [Style Sheet]
Written by Marco De la Cruz. Note that comments in the code should have a space after the %.

`Modula 2 (modula2.ssh)` [Style Sheet]
Written by Peter Bartke.

`Modula 3 (modula3.ssh)` [Style Sheet]
Written by Akim Demaille. Modula-3 is a member of the Pascal family of languages. Designed in the late 1980s at Digital Equipment Corporation and Olivetti, Modula-3 corrects many of the deficiencies of Pascal and Modula-2 for practical software engineering. In particular, Modula-3 keeps the simplicity of type safety of the earlier languages, while providing new facilities for exception handling, concurrency, object-oriented programming, and automatic garbage collection. Modula-3 is both a practical implementation language for large software projects and an excellent teaching language.

This sheet was designed based on Modula 3 home page[2].

`o2c (o2c.ssh)` [Style Sheet]

`Oberon (oberon.ssh)` [Style Sheet]
Written by Akim Demaille. Created by N. Wirth, Oberon is the successor of the Pascal and Modula-2 family of programming languages. It was specifically designed for systems programming, and was used to create the Oberon system in cooperation with J. Gutknecht. A few years later, the Oberon language was extended with additional object-oriented features to result in the programming language Oberon-2.

Implementation of the sheet based on The Project Oberon Site[3].

`Objective C (objc.ssh)` [Style Sheet]
Written by Paul Shum.

`OCaml (ocaml.ssh)` [Style Sheet]
This style should also suit other versions of ML (caml light, SML etc.).

`OCaml Yacc (mly.ssh)` [Style Sheet]
Written by Jean-Baptiste Nivoit. Should handle CAML Special Light parser files.

`Octave (octave.ssh)` [Style Sheet]
Written by C.P. Earls.

`Oracle parameter file (initora.ssh)` [Style Sheet]
Written by Pierre Mareschal. For init.ora parameter files.

`Oracle PL/SQL (plsql.ssh)` [Style Sheet]
Written by Pierre Mareschal. This style is to be checked.

`Oracle SQL (sql.ssh)` [Style Sheet]
Written by Pierre Mareschal. a2ps-sql Pretty Printer Version 1.0.0 beta - 18-MAR-97 For comments, support for – /*..*/ and //. This style is to be checked.

[2] `http://www.modula3.org/`

[3] `http://www.projectoberon.com/`

`Oracle SQL-PL/SQL-SQL*Plus (oracle.ssh)` [Style Sheet]
Written by Pierre Mareschal. 18-MAR-97 For comments, support for – /*..*/ and //. This style is to be checked.

`Pascal (pascal.ssh)` [Style Sheet]
Written by Akim Demaille. The standard Pascal is covered by this style. But some extension have been added too, hence modern Pascal programs should be correctly handled. Heavy highlighting maps mathematical symbols to their typographic equivalents.

`Perl (perl.ssh)` [Style Sheet]
Written by Denis Girou. As most interpreted languages, Perl is very free on its syntax, what leads to significant problems for a pretty printer. Please, be kind with our try. Any improvement is most welcome.

`PHP (php.ssh)` [Style Sheet]
Written by Hartmut Holzgraefe. This is a a2ps stylesheet for PHP syntax highlighting (just the PHP part, HTML is left 'as is'). This is my first try on a2ps stylesheets. It works OK for me. If it doesn't come up to your expectatios, then please tell me.

`pic16f84 (pic16f84.ssh)` [Style Sheet]
Written by Aleksandar Veselinovic. This style highlights PIC16F84 ASM code.

`PostScript (ps.ssh)` [Style Sheet]
Written by Akim Demaille. Only some keywords are highlighted, because otherwise listings are quickly becoming a big bold spot.

`PostScript Printer Description (ppd.ssh)` [Style Sheet]
Written by Akim Demaille. Support for Adobe's PPD files.

`Pov-Ray (pov.ssh)` [Style Sheet]
Written by Jean-Baptiste Nivoit. Should handle Persistence Of Vision input files.

`PreScript (pre.ssh)` [Style Sheet]
Written by Akim Demaille. This style defines commands in the canonic syntax of a2ps. It is meant to be used either as an input language, and to highlight the table of contents etc.

It can be a good choice of destination language for people who want to produce text to print (e.g. pretty-printing, automated documentation etc.) but who definitely do not want to learn PostScript, nor to require the use of LaTeX.

`PreTeX (pretex.ssh)` [Style Sheet]
Written by Akim Demaille. This style sheets provides LaTeX-like commands to format text. It is an alternative to the PreScript style sheet, in which formating commands are specified in a more a2ps related syntax.

It provides by the use of LaTeX like commands, a way to describe the pages that this program should produce.

`Prolog (prolog.ssh)` [Style Sheet]
Written by Akim Demaille. Help is needed on this sheet.

`Promela (promela.ssh)` [Style Sheet]
Written by Akim Demaille. There is no way for this program to highlight send and receive primitives.

`Python (python.ssh)` [Style Sheet]
Written by Akim Demaille. Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python web site[4], and can be freely distributed.

The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

`Rd -- Documentation for GNU R (rd.ssh)` [Style Sheet]
Written by Torsten Hothorn, Kurt Hornik, Dirk Eddelbuettel. R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.

R has a home page at 'https://www.r-project.org/'. It is free software distributed under a GNU-style copyleft, and an official part of the GNU project (GNU S).

`Reference Card (card.ssh)` [Style Sheet]
Written by Akim Demaille. This style sheet is meant to process help messages generated by Unix applications. It highlights the options (-short or –long), and their arguments. Normal use of this style sheet is through the shell script card (part of the a2ps package), but a typical hand-driven use is:

```
program --help | a2ps -Ecard
```

`REXX (rexx.ssh)` [Style Sheet]
Written by Alexander Mai. This style sheet supports REXX. You can get information about REXX from the REXX Language Association[5].

`Ruby (ruby.ssh)` [Style Sheet]
Written by Noritsugu Nakamura.

`S language (s.ssh)` [Style Sheet]
Written by Torsten Hothorn, Kurt Hornik, Dirk Eddelbuettel. Should handle code for interpreters of S, a language for statistical computating and graphics, such as R.

[4] `https://www.python.org`
[5] `https://www.rexxla.org`

R consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.

R has a home page at 'https://www.r-project.org/'. It is free software distributed under a GNU-style copyleft, and an official part of the GNU project ('GNU S').

`S transscript (st.ssh)` [Style Sheet]
Written by Torsten Hothorn, Kurt Hornik, Dirk Eddelbuettel. Should handle transscripts from interpreters of S, a language for statistical computing and graphics, such as R.

R consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.

R has a home page at 'https://www.r-project.org/'. It is free software distributed under a GNU-style copyleft, and an official part of the GNU project ('GNU S').

`Sather (sather.ssh)` [Style Sheet]
Written by Akim Demaille. Sather is an object oriented language designed to be simple, efficient, safe, flexible and non-proprietary. One way of placing it in the 'space of languages' is to say that it aims to be as efficient as C, C++, or Fortran, as elegant as and safer than Eiffel, and support higher-order functions and iteration abstraction as well as Common Lisp, CLU or Scheme.

Implementation of the sheet based on the Sather home page[6].

Heavy highlighting uses symbols for common mathematical operators.

`Scheme (scheme.ssh)` [Style Sheet]
Written by Akim Demaille. This style sheet is looking for a maintainer and/or comments.

`SDL-88 (sdl88.ssh)` [Style Sheet]
Written by Jean-Philippe Cottin. –strip-level=2 is very useful: it cancels the graphical information left by graphic editors. Only the pure specification is then printed.

`Sed (sed.ssh)` [Style Sheet]
Written by Akim Demaille. Comments and labels are highlighted. Other ideas are welcome! A lot of work is still needed.

`Shell (shell.ssh)` [Style Sheet]
Written by Akim Demaille. This style sheet is not meant to be used directly, but rather an as ancestor for shell style sheets.

`Small (small.ssh)` [Style Sheet]
Written by Christophe Continente. This style does not highlight the function definitions.

`SpecC (specc.ssh)` [Style Sheet]
Written by Hideaki Yokota. Non-textual operators are not highlighted. Some logical operators are printed as graphical symbols in the second level of pretty-printing.

[6] `https://www.gnu.org/software/sather/`

`SQL 92 (sql92.ssh)` [Style Sheet]
Written by Pierre Mareschal. 18-MAR-97 This style is to be checked.

`Standard ML (sml.ssh)` [Style Sheet]
Written by Franklin Chen, Daniel Wang. This style sheet takes advantage of the Symbol font to replace many ASCII operators with their natural graphical representation. This is enabled only at heavy highlighting.

`stratego (stratego.ssh)` [Style Sheet]
Written by Nicolas Tisserand. Highlights stratego source code

`Symbols (symbols.ssh)` [Style Sheet]
Written by Akim Demaille. This style sheet should be a precursor for any style sheet which uses LaTeX like symbols.

`TC Shell (tcsh.ssh)` [Style Sheet]
Written by Jim Diamond. C shell with file name completion and command line editing.

`TeX (tex.ssh)` [Style Sheet]
Written by Denis Girou. This is the style for (La)TeX files. It's mainly useful for people who develop (La)TeX packages. With '`-g`', common mathematical symbols are represented graphically.

`Texinfo (texinfo.ssh)` [Style Sheet]
Written by Akim Demaille. Heavy highlighting prints the nodes on separate pages which title is the name of the node.

`TeXScript (texscript.ssh)` [Style Sheet]
Written by Akim Demaille. TeXScript is the new name of what used to be called PreScript. New PreScript has pure a2ps names, PreTeX has pure TeX names, and TeXScript mixes both.

`Tiger (tiger.ssh)` [Style Sheet]
Written by Akim Demaille. Tiger is a toy language that serves as example of the book Modern Compiler Implementation[7] by Andrew W. Appel.

`tk (tk.ssh)` [Style Sheet]
Written by Akim Demaille, Larry W. Virden. Since everything, or almost, is a string, what is printed is not always what you would like.

`Tool Command Language (tcl.ssh)` [Style Sheet]
Written by Akim Demaille, Larry W. Virden. Since everything, or almost, is a string, what is printed is not always what you would like.

`Unified Diff (udiff.ssh)` [Style Sheet]
Written by Akim Demaille. This style is meant to be used onto the output unidiffs, that is to say output from '`diff -u`'.

[7] `https://www.cs.princeton.edu/~appel/modern/`

Typical use of this style is:

```
diff -u old new | a2ps -Eudiff
```

The prologue `diff` helps to highlight the differences (‘`a2ps -Ewdiff --prologue=diff`’).

`Unity (unity.ssh)` [Style Sheet]
Written by Jean-Philippe Cottin. The graphic conversion of the symbols (option ‘`-g`’) is nice.

`VERILOG (verilog.ssh)` [Style Sheet]
Written by Edward Arthur. This style is devoted to the VERILOG hardware description language.

`VHDL (vhdl.ssh)` [Style Sheet]
Written by Thomas Parmelan. Non-textual operators are not highlighted. Some logical operators are printed as graphical symbols in the second level of pretty-printing.

`Visual Basic for Applications (vba.ssh)` [Style Sheet]
Written by Dirk Eddelbuettel.

`Visual Tcl (vtcl.ssh)` [Style Sheet]
Written by Phil Hollenback. All the Vtcl keywords that aren't in Tcl or TclX.

`VRML (vrml.ssh)` [Style Sheet]
Written by Nadine Richard. According to Grammar Definition Version 2.0 ISO/IEC CD 14772[8].

`wdiff (wdiff.ssh)` [Style Sheet]
Written by Akim Demaille. This style is meant to be used onto the output of Franc,ois Pinard's program `wdiff`. `wdiff` is a utility that underlines the differences of words between to files. Where `diff` make only the difference between lines that have changed, `wdiff` reports words that have changed inside the lines.

Typical use of this style is:

```
wdiff old new | a2ps -Ewdiff
```

`wdiff` can be found in usual GNU repositories. The prologue `diff` helps to highlight the differences (‘`a2ps -Ewdiff --prologue=diff`’).

`XS (xs.ssh)` [Style Sheet]
Written by Kestutis Kupciunas. This style covers Perl XS language.

`Yacc (yacc.ssh)` [Style Sheet]
Written by Akim Demaille. Special tokens, and non terminal declarations are highlighted.

`Z Shell (zsh.ssh)` [Style Sheet]
Zsh is a UNIX command interpreter (shell) usable as an interactive login shell and as a shell script command processor. Of the standard shells, zsh most closely resembles

[8] `https://www.web3d.org/documents/specifications/14772/V2.0/part1/grammar.html`

ksh but includes many enhancements. Zsh has comand line editing, builtin spelling correction, programmable command completion, shell functions (with autoloading), a history mechanism, and a host of other features.

This style sheet highlights some classical program names and builtins in the second level of pretty-printing.

7.3 Type Setting Style Sheets

This section presents a few style sheets that define page description languages (compared to most other style sheet meant to pretty print source files).

7.3.1 Symbol

The style sheet `Symbol` introduces easy to type keywords to obtain the special characters of the PostScript font `Symbol`. The keywords are named to provide a LaTeX taste. These keywords are also the names used when designing a style sheet, hence to get the full list, see Section 7.6.1 [A Bit of Syntax], page 66.

If you want to know the correspondence, it is suggested to print the style sheet file of `Symbol`:

```
a2ps -g symbol.ssh
```

7.3.2 PreScript

`PreScript` has been designed in conjunction with `a2ps`. Since bold sequences, special characters etc. were implemented in `a2ps`, we thought it would be good to allow direct access to those features: `PreScript` became an input language for `a2ps`, where special font treatments are specified in an `ssh` syntax (see Section 7.6 [Style Sheets Implementation], page 66).

The main advantages for using `PreScript` are:

- it is fairly simple,
- `a2ps` is small and easy to install, hence it is available on every UNIX platform.

It can be a good candidate for generation of PostScript output (syntactic pretty-printers, generation of various reports etc.).

7.3.2.1 Syntax

Every command name begins with a backslash ('\'). If the command uses an argument, it is given between curly braces with no spaces between the command name and the argument.

The main limit on `PreScript` is that no command can be used inside another command. For instance the following line will be badly interpreted by `a2ps`:

```
\Keyword{Problems using \keyword{recursive \copyright} calls}
```

The correct way to write this in `PreScript` is

```
\Keyword{Problems using} \keyword{recursive} \copyright \Keyword{calls}.
```

Everything from an unquoted % to the end of line is ignored (comments).

7.3.2.2 PreScript Commands

These commands required arguments.

'`\keyword{`*text*`}`'
'`\Keyword{`*text*`}`'
: Highlight lightly/strongly the given *text*. Should be used only for a couple of adjacent words.

'`\comment{`*text*`}`'
'`\Comment{`*text*`}`'
: The *text* is given a special face. The *text* may be removed if option '`--strip`' is used.

'`\label{`*text*`}`'
'`\Label{`*text*`}`'
: *text* should be considered as a definition, or an important point in the structure of the whole text.

'`\string{`*text*`}`'
: Write *text* with string's face (e.g., in font Times).

'`\error{`*text*`}`'
: Write *text* with error's face (generally a very different face, so that you see immediately).

'`\symbol{`*text*`}`'
: *text* is written in the PostScript symbol font. This feature is not compatible with LaTeX. It is recommended, when possible, to use the special keywords denoting symbols, which are compatible with LaTeX (see Section 7.3.1 [Symbol], page 60).

'`\header{`*text*`}`'
'`\footer{`*text*`}`'
: Use *text* as header (footer) for the current page. If several headers or footers are defined on the same page, the last one is taken into account.

'`\encoding{`*key*`}`'
: Change dynamically the current encoding. After this command, the text is printed using the encoding corresponding to *key*.

7.3.2.3 Examples

`PreScript` and `a2ps` can be used for one-the-fly formating. For instance, on the `passwd` file:

```
ypcat passwd |
 awk -F: \
   '{print "\Keyword{" $5 "} (" $1 ") \rightarrow\keyword{" $7 "}"}'\
 | a2ps -Epre -P
```

7.3.3 PreTeX

The aim of the PreTeX style sheet is to provide something similar to `PreScript`, but with a more LaTeX like syntax.

7.3.3.1 Special characters

'$' is ignored in PreTEX for compatibility with LATEX, and '%' introduces a comment. Hence they are the only symbols which have to be quoted by a '\'. The following characters should also be quoted to produce good LATEX files, but are accepted by PreScript: '_', '&', '#'.

Note that *inside a command*, like \textbf, the quotation mechanism does not work in PreScript (\textrm{#$%} writes '#$%') though LATEX still requires quotation. Hence whenever special characters or symbols are introduced, they should be at the outer most level.

7.3.3.2 PreTEX Commands

These commands required arguments.

'\section{*Title*}'
'\subsection{*Title*}'
'\subsubsection{*Title*}.'
: Used to specify the title of a section, subsection or subsubsection.

'\textbf{*text*}'
'\textit{*text*}'
'\textbi{*text*}'
'\textrm{*text*}'
: write *text* in bold, italic, bold-italic, Times. Default font is Courier.

'\textsy{*text*}'
: *text* is written in the PostScript symbol font. This feature is not compatible with LATEX. It is recommended, when possible, to use the special keywords denoting symbols, which are compatible with LATEX (See the style sheet Symbol).

'\header{*text*}'
'\footer{*text*}'
: Use *text* as header (footer) for the current page. If several headers or footers are defined on the same page, the last one is taken into account.

'\verb+*text*+'
: Quote *text* so that no special sequence will be interpreted. In '\verb+*quoted string*+' '+' can be any symbol in '+', '!', '|', '#', '='.

'\begin{document}'
'\end{document}'
'\begin{itemize}'
'\end{itemize}'
'\begin{enumerate}'
'\end{enumerate}'
'\begin{description}'
'\end{description}'
: These commands are legal in LATEX but have no sense in PreTEX. Hence there are simply ignored and not printed (if immediately followed by an end-of-line).

7.3.3.3 Differences with LATEX

The following symbols, inherited from the style sheet Symbol, are not supported by LATEX:

‘\Alpha’, ‘\apple’, ‘\Beta’, ‘\carriagereturn’, ‘\Chi’, ‘\Epsilon’, ‘\Eta’, ‘\florin’, ‘\Iota’, ‘\Kappa’, ‘\Mu’, ‘\Nu’, ‘\Omicron’, ‘\omicron’, ‘\radicalex’, ‘\register’, ‘\Rho’, ‘\suchthat’, ‘\Tau’, ‘\therefore’, ‘\trademark’, ‘\varUpsilon’, ‘\Zeta’.

LaTeX is more demanding about special symbols. Most of them must be in so-called math mode, which means that the command must be inside ‘$’ signs. For instance, though

```
If \forall x \in E, x \in F then E \subseteq F.
```

is perfectly legal in PreTeX, it should be written

```
If $\forall x \in E, x \in F$ then $E \subseteq F$.
```

for LaTeX. Since in PreTeX every ‘$’ is discarded (unless quoted by a ‘\’), the second form is also admitted.

7.3.4 TeXScript

TeXScript is a replacement of the old version of PreScript: it combines both the a2ps-like and the LaTeX-like syntaxes through inheritance of both PreScript and PreTeX.

In addition it provides commands meant to ease processing of file for a2ps by LaTeX.

Everything between ‘%%TeXScript:skip’ and ‘%%TeXScript:piks’ will be ignored in TeXScript, so that there can be inserted command definitions for LaTeX exclusively.

The commands ‘\textbi’ (for bold-italic) and ‘\textsy’ (for symbol) do not exist in LaTeX. They should be defined in the preamble:

```
%%TeXScript:skip
\newcommand{\textbi}[1]{\textbf{\textit{#1}}}
\newcommand{\textsy}[1]{#1}
%%TeXScript:piks
```

There is no way in TeXScript to get an automatic numbering. There is no equivalent to the LaTeX environment enumerate. But every command beginning by \text is doubled by a command beginning by ‘\magic’. a2ps behaves the same way on both families of commands. Hence, if one specifies that arguments of those functions should be ignored in the preamble of the LaTeX document, the numbering is emulated. For instance

```
\begin{enumerate}
\magicbf{1.}\item First line
\magicbf{2.}\item Second line
\end{enumerate}
```

will be treated the same way both in TeXScript and LaTeX.

‘\header’ and ‘\footer’, are not understood by LaTeX.

7.4 Faces

A *face* is an attribute given to a piece of text, which specifies how it should look like. Since a2ps is devoted to pretty-printing source files, the faces it uses are related to the syntactic entities that can be encountered in a file.

The faces a2ps uses are:

‘Plain’ This corresponds to the text body.

'`Keyword`'
'`Keyword_strong`'
: These are related to the keywords that may appear in a text.

'`Comment`'
'`Comment_strong`'
: These are related to comments in the text. Remember that comments should be considered as non essential ("*Aaaeaaarg*" says the programmer); indeed, the user might suppress the comments thanks (?) to the option '`--strip-level`'. Hence, **never** use these faces just because you think they look better on, say, strings.

'`Label`'
'`Label_strong`'
: These are used when a point of extreme importance, or a sectioning point, is met. Typically, functions declarations etc.

'`String`' Used mainly for string and character literals.

'`Error`' Used to underline the presence of an error. For instance in Encapsulated PostScript, some PostScript operators are forbidden: they are underlined as errors.

Actually, there is also the face '`Symbol`', but this one is particular: it is not legal changing its font.

7.5 Style Sheets Semantics

`a2ps` pretty prints a source file thanks to *style sheets*, one per language. In the following is described how the style sheets are defined. You may skip this section if you don't care how `a2ps` does this, and if you don't expect to implement new styles.

7.5.1 Name and key

Every style sheet has both a key, and a name. The name can be clean and beautiful, with any character you might want. The key is in fact the prefix part of the file name, and is alpha-numerical, lower case, and less than 8 characters long.

Anywhere `a2ps` needs to recognize a style sheet by a name, **it uses the key** (in the `sheets.map` file, with the option '`-E`', etc.).

As an example, C++ is implemented in a file called `cxx.ssh`, in which the name is declared to be '`C++`'.

The rationale is that not every system accepts any character in the file name (e.g., no '`+`' in MS-DOS). Moreover, it allows to make symbolic links on the ssh files (e.g., '`ln -s cxx.ssh c++.ssh`' let's you use '`-E c++`').

7.5.2 Comments

ssh files can include the name of its author, a version number, a documentation note and a requirement on the version of `a2ps`. For instance, if a style sheet requires `a2ps` version 4.9.6, then `a2ps` version 4.9.5 will reject it.

7.5.3 Alphabets

`a2ps` needs to know the beginning and the end of a word, especially keywords. Hence it needs two alphabets: the first one specifying by which letters an identifier can begin, and the second one for the rest of the word. If you prefer, a keyword starts with a character belonging to the first alphabet, and a character not pertaining to the second is a separator.

7.5.4 Case sensitivity

If the style is case insensitive, then matching is case insensitive (keywords, operators and sequences).

7.5.5 P-Rules

A *P-rule* (Pretty printing rule), or *rule* for short, is a structure which consists of two items:

lhs
left-hand side
: its source string, with which the source file is compared;

rhs
right hand side
: a list of faced strings which will replace the text matched in the pretty-printed output. A faced string is composed of
  - a string, or a reference to a part of the source string (see Section "Back-reference Operator" in *Regex manual*)
  - the face to use to print it

Just a short example: '`(foo, bar, Keyword_strong)`' as a rule means that every input occurrence of '`foo`' will be replaced by '`bar`', written with the `Keyword_strong` face.

If the destination string is empty, then `a2ps` will use the source string. This is different from giving the source string as a destination string if the case is different. An example will make it fairly clear.

Let `foobar` be a case insensitive style sheet including the rules '`(foo, "", Keyword)`' and '`(bar, bar, Keyword)`'. Then, on the input '`FOO BAR`', `a2ps` will produce '`FOO bar`' in `Keyword`.

`a2ps` implements two different ways to match a string. The difference comes from that some keywords are sensitive to the delimiters around them (such as '`unsigned`' and '`int`' in `C`, which are definitely not the same thing as '`unsignedint`'), and others not (in `C`, '`!=`' is "different from" both in '`a != b`' and '`a!=b`').

The first ones are called *keywords* in `a2ps` jargon, and the seconds are *operators*. Operators are matched anywhere they appear, while keywords need to have separators around them (see Section 7.5.3 [Alphabets], page 65).

Let us give a more complicated example: that of the `Yacc` rules. A rule in `Yacc` is of the form:

```
a_rule : part1 part2 ;
```

Suppose you want to highlight these rules. To recognize them, you will write a regular expression specifying that:

1. it must start at the beginning of the line,

2. then there is string composed of symbols, which is what you want to highlight,
3. and a colon, which can be preceded by blank characters.

The regexp you want is: '`/^[a-zA-Z0-9_]*[\t ]*:/`'. But with the rule

```
/^[a-zA-Z0-9_]*[\t ]*:/, "", Label_strong
```

the blanks and the colon are highlighted too. Hence you need to specify some parts in the regexp (see Section "Back-reference Operator" in *Regex manual*), and use a longer list of destination strings. The correct rule is

```
(/^([a-zA-Z0-9_]*)([\t ]*:)/, \1 Label_strong, \2 Plain)
```

Since it is a bit painful to read, regexps can be spread upon several lines. It is strongly suggested to break them by groups, and to document the group:

```
(/^([a-zA-Z0-9_]*)/    # \1. Name of the rule
 /([\t ]*:)/           # \2. Trailing space and colon
 \1 Label_strong, \2 Plain)
```

7.5.6 Sequences

A *sequence* is a string between two *markers*, along with a list of exceptions. A marker is a fixed string. Typical examples are comments, string (with usually '`"`' as opening and closing markers, and '`\\`' and '`\"`' as exceptions) etc. Three faces are used: one for the initial marker, one for the core of the sequence, and a last one for the final maker.

7.5.7 Optional entries

There are two levels of pretty-printing encoded in the style sheets. By default, `a2ps` uses the first level, called *normal*, unless the option '`-g`' is specified, in which case, *heavy* highlighting is invoked, i.e., optional keywords, operators and sequences are considered.

7.6 Style Sheets Implementation

In the previous section (see Section 7.5 [Style sheets semantics], page 64) were explained the various items needed to understand the machinery involved in pretty printing. Here, their implementation, i.e., how to write a style sheet file, is explained. The next section (see Section 7.7 [A tutorial on style sheets], page 73), exposes a step by step simple example.

7.6.1 A Bit of Syntax

Here are the lexical rules underlying the style sheet language:

- the separators are white space, form feed, new line, and tab.
- '`#`' introduces a comment, ended at the end of the line.
- special characters are the separators, plus '`#`', '`"`', '`,`', '`(`', '`)`', '`+`' and '`/`'. Any other character is a regular character.
- the list of the structuring keywords is

  `alphabet`, `alphabets`, `are`, `case`, `documentation`, `end`, `exceptions`, `first`, `in`, `insensitive`, `is`, `keywords`, `operators`, `optional`, `second`, `sensitive`, `sequences`, `style`

– the list of the keywords designating faces is

  `Comment`, `Comment_strong`, `Encoding`, `Error`, `Index1`, `Index2`, `Index3`, `Index4`, `Invisible`, `Keyword`, `Keyword_strong`, `Label`, `Label_strong`, `Plain`, `String`, `Symbol`, `Tag1`, `Tag2`, `Tag3`, `Tag4`

– the list of keywords designating special sequences is

  `C-char`, `C-string`

– the list of keywords representing special characters is

  `---`, `\Alpha`, `\Beta`, `\Chi`, `\Delta`, `\Downarrow`, `\Epsilon`, `\Eta`, `\Gamma`, `\Im`, `\Iota`, `\Kappa`, `\Lambda`, `\Leftarrow`, `\Leftrightarrow`, `\Mu`, `\Nu`, `\Omega`, `\Omicron`, `\Phi`, `\Pi`, `\Psi`, `\Re`, `\Rho`, `\Rightarrow`, `\Sigma`, `\Tau`, `\Theta`, `\Uparrow`, `\Upsilon`, `\Xi`, `\Zeta`, `\aleph`, `\alpha`, `\angle`, `\approx`, `\beta`, `\bullet`, `\cap`, `\carriagereturn`, `\cdot`, `\chi`, `\circ`, `\clubsuit`, `\cong`, `\copyright`, `\cup`, `\delta`, `\diamondsuit`, `\div`, `\downarrow`, `\emptyset`, `\epsilon`, `\equiv`, `\eta`, `\exists`, `\florin`, `\forall`, `\gamma`, `\geq`, `\heartsuit`, `\in`, `\infty`, `\int`, `\iota`, `\kappa`, `\lambda`, `\langle`, `\lceil`, `\ldots`, `\leftarrow`, `\leftrightarrow`, `\leq`, `\lfloor`, `\mu`, `\nabla`, `\neq`, `\not`, `\not\in`, `\not\subset`, `\nu`, `\omega`, `\omicron`, `\oplus`, `\otimes`, `\partial`, `\perp`, `\phi`, `\pi`, `\pm`, `\prime`, `\prod`, `\propto`, `\psi`, `\radicalex`, `\rangle`, `\rceil`, `\register`, `\rfloor`, `\rho`, `\rightarrow`, `\sigma`, `\sim`, `\spadesuit`, `\subset`, `\subseteq`, `\suchthat`, `\sum`, `\supset`, `\supseteq`, `\surd`, `\tau`, `\theta`, `\therefore`, `\times`, `\trademark`, `\uparrow`, `\upsilon`, `\varUpsilon`, `\varcopyright`, `\vardiamondsuit`, `\varphi`, `\varpi`, `\varregister`, `\varsigma`, `\vartheta`, `\vartrademark`, `\vee`, `\wedge`, `\wp`, `\xi`, `\zeta`

  It is a good idea to print the style sheet '`symbols.ssh`' to see them:

  ```
  a2ps symbols.ssh
  ```

– a string starts and finishes with '`"`', and may contain anything. Regular `C` escaping mechanism is used.

– a regular expression starts and finishes with '`/`', and may contain anything. Regular `C` escaping mechanism is used. Regexps can be split in several parts, *a' la* C strings (i.e., '`/part 1/ /part 2/`').

– any sequence of regular characters which is not a keyword, is a string (consider this as a shortcut, avoiding extraneous '`"`').

7.6.2 Style Sheet Header

The definition of the name of the style sheet is:

```
style name is
  # body of the style sheet
end style
```

The following constructions are optional:

`version` To define the version number of the style sheet

```
version is version-number
```

`written` To define the author(s).

```
written by authors
```

Giving your email is useful for bug reports about style sheets.

```
written by "Some Body <Some.Body@some.whe.re>"
```

`requires` To specify the version of `a2ps` it requires. `a2ps` won't accept a file which requires a higher version number than its own.

```
requires a2ps a2ps-version-number
```

`documentation`

To leave extra comments people should read.

```
documentation is
   strings
end documentation
```

strings may be a list of strings, without comas, in which case new lines are automatically inserted between each item. See Section 5.1 [Documentation Format], page 38, for details on the format.

Please, write useful comments, not '`This style is devoted to C files`', since the name is here for that, nor '`Report errors to mail@me.somewhere`', since `written by` is there for that.

```
documentation is
    "Not all the keywords are used, to avoid too much"
    "bolding. Heavy highlighting (code(-g)code), covers"
    "the whole language."
end documentation
```

7.6.3 Syntax of the Words

There are two things `a2ps` needs to know: what is symbol consistent, and whether the style is case insensitive.

`alphabet` To define two different alphabets, use

```
first alphabet is string
second alphabet is string
```

If both are identical, you may use the shortcut

```
alphabets are string
```

The default alphabets are

```
first alphabet is
  "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_"
second alphabet is
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_\
0123456789"
```

Note that it is on purpose that no characters interval are used.

`case`

```
case insensitive        # e.g., C, C++ etc.
case sensitive          # e.g., Perl, Sather, Java etc.
```

The default is `case insensitive`.

7.6.4 Inheriting from Other Style Sheets

It is possible to extend an existing style. The syntax is:

```
ancestors are
   ancestor_1[, ancestor_2...]
end ancestors
```

where *ancestor1* etc. are style sheet keys.

For semantics, the rules are the following:

- the ancestors are read in order;
- the definition of the current style is read last;
- it is always the last item read which wins (last defined alphabets, case sensitivity, keywords, operators and sequences).

As an example, both `C++` and `Objective C` style sheets extend the `C` style sheet:

```
style "Objective C" is
#[...]
ancestors are
   c
end ancestors
#[...]
end style
```

To the biggest surprise of the author, mutually dependent style sheets do work!

7.6.5 Syntax for the P-Rules

See Section 7.5.5 [P-Rules], page 65, for the definition of *P-rule*.

Because of various short cuts, there are many ways to declare a rule:

```
rules     ::= rule_1 ‘,’ rule_2...
rule      ::= ‘(’ lhs rhs ‘)’
            | lhs srhs ;
lhs       ::= string | regex ;
rhs       ::= srhs ‘,’ ...
srhs      ::= latex-keyword | expansion face
expansion ::= string | ‘\’num | <nothing>;
face      ::= face-keyword | <nothing>;
```

The rules are the following:

- If the left-hand side (lhs) is a regular expression, then it is compiled with the following syntax bits:

```
#define RE_SYNTAX_A2PS \
  (/* Allow char classes. */ \
    RE_CHAR_CLASSES \
  /* Be picky. */ \
  | RE_CONTEXT_INVALID_OPS \
  /* Allow intervals with ‘{’ and ‘}’, forbid invalid ranges. */\
  | RE_INTERVALS | RE_NO_BK_BRACES | RE_NO_EMPTY_RANGES \
  /* ‘(’ and ‘)’ are the grouping operators. */ \
```

```
| RE_NO_BK_PARENS \
/* '|' is the alternation. */ \
| RE_NO_BK_VBAR)
```

Basically it means that all of the possible operators are used, and that they are in non-backslashed form. For instance '`(`' and '`)`' stand for the group operator, while '`\\(`' stands for the character '`(`'. See Section "Regular Expression Syntax" in *Regex manual*, for a detailed description of the regular expressions.

- If no *expansion* is specified, then the matched string is used. For instance '`(/fo*/, NULL, Keyword)`' applied on the source '`fooooo`' produces '`fooooo`' in `Keyword`.
- If no *face* is given, then
  - if the context defines the default face, then this face is used;
  - if no default face is given, `PLAIN` is used.

7.6.6 Declaring the keywords and the operators

Basically, keywords and operators are lists of rules. The syntax is:

```
keywords are
  rules
end keywords
```

or

```
keywords in face-keyword are
  rules
end keywords
```

in which case the default face is set to *face-keyword*.

As an example:

```
keywords in Keyword_strong are
  /foo*/,
  "bar" "BAR" Keyword,
  -> \rightarrow
end keywords
```

is valid.

The syntax for the operators is the same, and both constructs can be qualified with an `optional` flag, in which case they are taken into account in the heavy highlighting mode (see Section 3.1.7 [Pretty Print Options], page 21).

This is an extract of the `C` style sheet:

```
optional operators are
   -> \rightarrow,
   && \wedge,
   || \vee,
   != \neq,
   == \equiv,
   # We need to protect these, so that <= is not replaced in <<=
   <<=,
   >>=,
```

```
   <= \leq,
   >= \geq,
   ! \not
end operators
```

Note how '`<<=`' and '`>>=`' are protected (there are defined to be written as is when met in the source). This is to prevent the two last characters of '`<<=`' from being converted into a 'less or equal' sign.

The order in which you define the elements of a category (but the sequences) does not matter. But since `a2ps` sorts them at run time, it may save time if the alphabetical `C`-order is more or less followed.

You should be aware that when declaring a keyword with a regular expression as lhs, then `a2ps` automatically makes this expression matching only if there are no character of the first alphabet both just before, and just after the string.

In term of implementation, it means that

```
keywords are
  /foo|bar/
end keywords
```

is exactly the same as

```
operators are
  /\\b(foo|bar)\\b/
end operators
```

This can cause problems if you use anchors (e.g. `$`, or `^`) in keywords: the matcher will be broken. In this particular case, define your keywords as operators, taking care of the '`\\b`' by yourself.

See Section "Match-word-boundary Operator" in *Regex manual*, for details on '`\b`'.

7.6.7 Declaring the sequences

Sequences admit several declarations too:

```
sequences      ::= sequences are
                     sequence_1 ',' sequence_2...
                   end sequences
sequence       ::= rule in_face close_opt exceptions_opt
                 | C-string
                 | C-char
                 ;
close_opt      ::= rule
                 | closers are
                     rules
                   end closers
                 | <nothing>
                 ;
exceptions_opt ::= exceptions are
                     rules
                   end exceptions
```

```
                    | <nothing>
                    ;
```

The rules are:

– The default face is then *in_face*.
– If no closing rule is given, '`"\n"`' (i.e., end-of-line) is used.

As a first example, here is the correct definition for a `C` string:

```
sequences are
 "\"" Plain String "\"" Plain
     exceptions are
        "\\\\", "\\\""
     end exceptions
end sequences
```

Since a great deal of languages uses this kind of constructs, you may use `C-string` to mean exactly this, and `C-char` for manifest characters defined the `C` way.

The following example comes from `ssh.ssh`, the style sheet for style sheet files, in which there are two kinds of pseudo-strings: the strings ('`"example"`'), and the regular expressions ('`/example/`'). We do not want the content of the pseudo-strings in the face `String`.

```
sequences are
  # The comments
  "#" Comment,

  # The name of the style sheet
  "style " Keyword_strong (Label + Index1) " is" Keyword_strong,

  # Strings are exactly the C-strings, though we don't want to
  # have them in the "string" face
  "\"" Plain "\""
     exceptions are
        "\\\\", "\\\""
     end exceptions,

  # Regexps
  "/" Plain "/"
     exceptions are
        "\\\\", "\\\/"
     end exceptions

end sequences
```

The order between sequences does matter. For instance in Java, '`/**`' introduces strong comments, and '`/*`' comments. '`/**`' *must* be declared before '`/*`', or it will be hidden.

There are actually some sequences that could have been implemented as operators with a specific regular expression (that goes up to the closer). Nevertheless be aware of a big difference: regular expression are applied to a single line of the source file, hence, they cannot match on several lines. For instance, the `C` comments,

```
/*
```

```
 * a comment
 */
```

cannot be implemented with operators, though C++ comments can:

```
//
// a comment
//
```

7.6.8 Checking a Style Sheet

Once your style sheet is written, you may want to let a2ps perform simple tests on it (e.g., checking there are no rules involving upper case characters in a case insensitive style sheet, etc.). These tests are performed when verbosity includes the style sheets.

you may also want to use the special convention that when a style sheet is required with a suffix, then a2ps will not look at it in its library path, but precisely from when you are.

Suppose for instance you extended the c.ssh style sheet, which is in the current directory, and is said case insensitive. Run

```
ubu $ a2ps foo.c -Ec.ssh -P void -v sheets
# Long output deleted
Checking coherence of "C" (c.ssh)
a2ps: c.ssh:‘FILE’ uses upper case characters
a2ps: c.ssh:‘NULL’ uses upper case characters
"C" (c.ssh) is corrupted.
---------- End of Finalization of c.ssh
```

Here, it is clear that C is not case insensitive.

7.7 A Tutorial on Style Sheets

In this section a simple example of style sheet is entirely covered: that of ChangeLog files.

ChangeLog files are some kind of memory of changes done to files, so that various programmers can understand what happened to the sources. This helps a lot, for instance, in guessing what recent changes may have introduced new bugs.

7.7.1 Example and syntax

First of all, here is a sample of a ChangeLog file, taken from the misc/ directory of the original a2ps package:

```
Sun Apr 27 14:29:22 1997  Akim Demaille  <demaille@inf.enst.fr>

        * base.ps: Merged in color.ps, since now a lot is
          common [added box and underline features].

Fri Apr 25 14:05:20 1997  Akim Demaille  <demaille@inf.enst.fr>

        * color.ps: Added box and underline routines.

Mon Mar 17 20:39:11 1997  Akim Demaille  <demaille@gargantua.enst.fr>
```

```
        * base.ps: Got rid of CourierBack and reencoded_backspace_font.
          Now the C has to handle this by itself.

Sat Mar  1 19:12:22 1997  Akim Demaille  <demaille@gargantua.enst.fr>

        * *.enc: they build their own dictionaries, to ease multi
          lingual documents.
```

The syntax is really simple: A line specifying the author and the date of the changes, then a list of changes, all of them starting with an star followed by the name of the files concerned, then optionally between parentheses the functions affected, and then some comments.

7.7.2 Implementation

Quite naturally the style will be called `ChangeLog`, hence:

```
style ChangeLog is
written by "Akim Demaille <demaille@inf.enst.fr>"
version is 1.0
requires a2ps 4.9.5

documentation is
   "This is a tutorial style sheet.\n"
end documentation
  ...
end style
```

A first interesting and easy entry is that of function names, between '`(`' and '`)`':

```
sequences are
  "(" Plain Label ")" Plain
end sequences
```

A small problem that may occur is that there can be several functions mentioned separated by commas, that we don't want to highlight this way. Commas, here, are exceptions. Since regular expressions are not yet implemented in `a2ps`, there is a simple but stupid way to avoid that white spaces are all considered as part of a function name, namely defining two exceptions: one which captures a single comma, and a second, capturing a comma and its trailing space.

For the file names, the problem is a bit more delicate, since they may end with '`:`', or when starts the list of functions. Then, we define two sequences, each one with one of the possible closers, the exceptions being attached to the first one:

```
sequences are
  "* " Plain Label_strong ":" Plain
     exceptions are
        ", " Plain, "," Plain
     end exceptions,
  "* " Plain Label_strong " " Plain
end sequences
```

Finally, let us say that some words have a higher importance in the core of text: those about removing or adding something.

```
keywords in Keyword_strong are
  add, added, remove, removed
end keywords
```

Since they may appear in lower or upper, of mixed case, the style will be defined as case insensitive.

Finally, we end up with this style sheet file, in which an optional highlighting of the mail address of the author is done. Saving the file is last step. But do not forget that a style sheet has both a name as nice as you may want (such as '`Common Lisp`'), and a key on which there are strict rules: the prefix must be alpha-numerical, lower case, with no more than 8 characters. Let's chose `chlog.ssh`.

```
# This is a tutorial on a2ps' style sheets
style ChangeLog is
written by "Akim Demaille <demaille@inf.enst.fr>"
version is 1.0
requires a2ps 4.9.5

documentation is
   "Second level of high lighting covers emails."
end documentation

sequences are
  "(" Plain Label ")" Plain
     exceptions are
        ", " Plain, "," Plain
     end exceptions,
  "* " Plain Label_strong ":" Plain
     exceptions are
        ", " Plain, "," Plain
     end exceptions,
  "* " Plain Label_strong " " Plain
end sequences

keywords in Keyword_strong are
  add, added, remove, removed
end keywords

optional sequences are
   < Plain Keyword > Plain
end sequences
end style
```

As a last step, you may which to let `a2ps` check your style sheet, both its syntax, and common errors:

```
ubu $ a2ps -vsheet -E/tmp/chlog.ssh ChangeLog -P void
Long output deleted
Checking coherence of "ChangeLog" (/tmp/chlog.ssh)
"ChangeLog" (/tmp/chlog.ssh) is sane.
```

```
---------- End of Finalization of /tmp/chlog.ssh
```

It's all set, your style sheet is ready!

7.7.3 The Entry in `sheets.map`

The last touch is to include the pattern rules about `ChangeLog` files (which could appear as `ChangeLog.old` etc.) in `sheets.map`:

```
# ChangeLog files
chlog:  /ChangeLog*/
```

This won't work... Well, not always. Not for instance if you print `misc/ChangeLog`. This is not a bug, but truly a feature, since sometimes one gets more information about the type of a file from its path, than from the file name.

Here, to match the preceding path that may appear, just use '`*`':

```
# ChangeLog files
chlog:  /*ChangeLog*/
```

If you want to be more specific (`FooChangeLog` should not match), use:

```
# ChangeLog files
chlog:  /ChangeLog*/ /*\/ChangeLog*/
```

7.7.4 More Sophisticated Rules

The example we have presented until now uses only basic features, and does not take advantage of the regexp. In this section we should how to write more evolved pretty printing rules.

The target will be the lines like:

```
Sun Apr 27 14:29:22 1997  Akim Demaille  <demaille@inf.enst.fr>

Fri Apr 25 14:05:20 1997  Akim Demaille  <demaille@inf.enst.fr>
```

There are three fields: the date, the name, the mail. These lines all start at the beginning of line. The last field is the easier to recognize: is starts with a '`<`', and finishes with a '`>`'. Its rule is then '`/<[^>]+>/`'. It is now easier to specify the second: it is composed only of words, at least one, separated by blanks, and is followed by the mail: '`/[[:alpha:]]+([ \t]+[[:alpha:]]+)*/`'. To concatenate the two, we introduce optional blanks, and we put each one into a pair of '`(`'-'`)`' to make each one a recognizable part:

```
([[:alpha:]]+([ \t]+[[:alpha:]]+)*)
(.+)
(<[^>]+>)
```

Now the first part is rather easy: it starts at the beginning of the line, finishes with a digit. Once again, it is separated from the following field by blanks. Split by groups (see Section "Grouping Operators" in *Regex manual*), we have:

```
^
([^\t ].*[0-9])
([ \t]+)
([[:alpha:]]+([ \t]+[[:alpha:]]+)*)
(.+)
(<[^>]+>)
```

Now the destination is composed of back references to those groups, together with a face:

```
# We want to highlight the date and the maintainer name
optional operators are
  (/^([^\t ].*[0-9])/                          # \1. The date
   /([ \t]+)/                                  # \2. Spaces
   /([[:alpha:]]+([ \t]+[[:alpha:]]+)*)/       # \3. Name
   /(.+)/                                      # \5. space and <
   /(<[^>]+)>/                                 # \6. email
   \1 Keyword, \2 Plain, \3 Keyword_strong,
   \5 Plain, \6 Keyword, > Plain)
end operators
```

Notice the way regexps are split, to ease reading.

7.7.5 Guide Line for Distributed Style Sheets

This section is meant for people who wish to contribute style sheets. There is a couple of additional constraints, explained here.

The Copyright
: Please, do put a copyright in your file, the same as all other distributed files have: it should include your name, but also the three paragraphs stating the sheet is covered by the GPL. I won't distribute files without these paragraphs.

The Version
: Do put a version number, so that people can track evolutions.

The Requirements
: Make sure to include a requirement on the needed version of `a2ps`. If you don't know what to put, just put the version of the `a2ps` you run.

The Documentation
: The documentation string is mandatory. Unless the language your style sheet covers is widely known, please document a bit what the style sheet is meant for. If there were choices you made, if there are special behaviors, document them.

The `sheets.map` *Entries*
: Put in a comment on the `sheets.map` lines that correspond to your style sheet.

A Test File
: It is better to give a test file, as small as possible, that contains the most specific and/or most difficult contructs that your style sheet supports. I need to be able to distribute this file, therefore, do not put anything that is copyrighted.

Finally, make sure your style sheet behaves well! (see Section 7.6.8 [Checking a Style Sheet], page 73)

8 PostScript

This chapter is devoted to the information which is only relevant to PostScript.

8.1 Foreword: Good and Bad PostScript

To read this section, the reader must understand what DSC are (see Appendix A [Glossary], page 92).

> *Why are there good PostScript files, easy to post-process, and bad files that none of my tools seem to understand? They print fine though!*

Once you understood that PostScript is not a page description format (like PDF is), you'll have understood most of the problem. Let's imagine for a second that you are a word processor.

The user asks you to print his/her 100 page document in PostScript. Up to page 50, there are few different fonts used. Then, on pages 51 to 80, there are now many different heavy fonts.

When/where will you download the fonts?

The most typical choice, sometimes called *Optimize for Speed*, is, once you arrived to page 51, to download those fonts **once** for the rest of the document. The global processing chain will have worked quite quickly: little effort from the software, same from the printer; better yet: you can start sending the file to the printer even before it is finished! The problem is that this is not DSC conformant, and it is easy to understand why: if somebody wants to print only the page 60, then s/he will lack the three fonts which were defined in page 51... This document is not page independent.

Another choice is to download the three fonts in **each** page ranging from 51 to 80, that is the PostScript file contains 30 times the definition of each font. It is easy for the application to do that, but the file is getting real big, and the printer will have to interpret 30 times the same definitions of fonts. But it is DSC conformant! And you can still send the file while you make it.

Now you understand why

> **Non DSC conformant files are not necessarily badly designed files from broken applications.**

They are files meant to be sent directly to the printer (they are still perfect PostScript files after all!), they are not meant to be post-processed. And the example clearly shows why they are **right**.

There is a third possibility, sometimes called *Optimize for Portability*: downloading the three fonts in the prologue of the document, i.e., the section before the first page where are given all the common definitions of the whole file. This is a bit more complicated to implement (the prologue, which is issued first though, grows at the same time as you process the file), and cannot be sent concurrently with the processing (you have to process the whole file to design the prologue). This file is small (the fonts are downloaded once only), and DSC conformant. Well, there are problems, of course... You need to wait before sending the output, it can be costly for the computer (which cannot transfer as it produces), and for the printer (you've burnt quite a lot of RAM right since the beginning just to hold fonts that won't be used before page 51... This can be a real problem for small printers).

This is what `a2ps` does.

If should be clear that documents optimized for speed should never escape the way between the computer and the printer: no post-processing is possible.

What you should remember is that some applications offer the possibility to tune the PostScript output, and they can be praised for that. Unfortunately, when these very same applications don't automatically switch to "Optimize for Portability" when you save the PostScript file, and they can be criticized for that.

So please, think of the people after you: if you create a PostScript file meant to be exchanged, read, printed, etc; by other people: give sane DSC conformant, optimized for portability files.

8.2 Page Device Options

Page device is a PostScript level 2 feature that offers an uniform interface to control the printer's output device. `a2ps` protects all page device options inside an if block so they have no effect in level 1 interpreters. Although all level 2 interpreters support page device, they do not have to support all page device options. For example some printers can print in duplex mode and some cannot. Refer to the documentation of your printer for supported options.

Here are some usable page device options which can be selected with the '`-S`' option ('`--setpagedevice`'). For a complete listing, see *PostScript Language Reference Manual* (section 4.11 Device Setup in the second edition, or section 6, Device Control in the third edition).

`Collate` *`boolean`*
: how output is organized when printing multiple copies

`Duplex` *`boolean`*
: duplex (two side) printing

`ManualFeed` *`boolean`*
: manual feed paper tray

`OutputFaceUp` *`boolean`*
: print output 'face up' or 'face down'

`Tumble` *`boolean`*
: how opposite sides are positioned in duplex printing

8.3 Statusdict Options

The `statusdict` is a special storage entity in PostScript (called a *dictionary*), in which some variables and operators determine the behavior of the printer. This is an historic horror that existed before page device definitions were defined. They are even more printer dependent, and are provided only for the people who don't have a level printer. In any case, refer to the documentation of your printer for supported options.

Here are some statusdict definitions in which you might be interested:

`manualfeed` *`boolean`*
: Variable which determine that the manual fed paper tray will be used. Use is '`--statusdict=manualfeed::true`'.

`setmanualfeed` *boolean*
: Idem as the previous point, but use is '`--statusdict=setmanualfeed:true`'.

`setduplexmode` *boolean*
: If *boolean*, then print Duplex. Use if '`--statusdict=setduplexmode:true`'.

8.4 Colors in PostScript

Nevertheless, here are some tips on how to design your PostScript styles. It is strongly recommended to use `gray.pro` or `color.pro` as a template.

There are two PostScript instructions you might want to use in your new PostScript prologue:

`setgray`
: this instruction must be preceded by a number between 0 (black) and 1 (white). It defines the gray level used.

`setrgbcolor`
: this instruction must be preceded by three numbers between 0 (0 %) and 1 (100%). Those three numbers are related to red, green and blue proportions used to designate a color.

`a2ps` uses two higher level procedures, `BG` and `FG`, but both use an argument as in `setrgbcolor`. So if you wanted a gray shade, just give three times the same ratio.

8.5 `a2ps` PostScript Files

`a2ps` uses several types of PostScript files. Some are standards, such as font files, and others are meant for a2ps only.

All a2ps files have two parts, one being the comments, and the other being the content, separated by the following line:

```
% code follows this line
```

8.6 Designing PostScript Prologues

It is pretty known that satisfying the various human tastes is an NEXPTIME-hard problem, so `a2ps` offers ways to customize its output through the *prologue files*. But since the authors feel a little small against NEXPTIME, they agreed on the fact that **you** are the one who will design the look you like.

Hence in this section, you will find what you need to know to be able to customize `a2ps` output.

Basically, `a2ps` uses *faces* which are associated to their "meaning" in the text. `a2ps` let's you change the way the faces look.

8.6.1 Definition of the faces

There are three things that define a face:

Its font
: You should never call the font by yourself, because sometimes `a2ps` may decide that another font would be better. This is what happens for instance if a font does not support the encoding you use.

Hence, never set the font by yourself, but ask `a2ps` to do it. This is done through a line:

```
%Face: face real-font-name size
```

This line tells `a2ps` that the font of *face* is *real-font-name*. It will replace this line by the correct PostScript line to call the needed font, and will do everything needed to set up the font.

The size of the text body is `bfs`.

Its background color

There are two cases:

1. You want a background color, then give the *RGB* (see Section 8.4 [Colors in PostScript], page 80) ratio and `true` to `BG`:

   ```
   0.8 0.8 0 true BG
   ```

2. You don't want a background color, then call `BG` with `false`:

   ```
   false BG
   ```

Its foreground color

As `BG`, call `FG` with an *RGB* ratio:

```
0 0.5 0 FG
```

Its underlining

`UL` requires a boolean argument, depending whether you want or not the current face to be underlined.

```
true UL
```

Its boxing Requiring a boolean, `BX` let's a face have a box drawn around.

8.6.2 Prologue File Format

Prologue files for `a2ps` must have '`pro`' as suffix. Documentation (reported with '`--list-prologues`') can be included in the comment part:

```
Documentation
This prologue is the same as the prologue code(pb)code, but using
the bold version of the fonts.
EndDocumentation
% code follows this line
```

See Section 5.1 [Documentation Format], page 38, for more on the format.

8.6.3 A step by step example

We strongly suggest our readers not to start from scratch, but to copy one of the available styles (see the result of '`a2ps --list=prologues`'), to drop it in one of `a2ps` directories (say '`$HOME/.a2ps`', and to patch it until you like it.

Here, we will start from `color.pro`, trying to give it a funky look.

Say you want the keywords to be in Helvetica, drawn in a flashy pink on a light green. And strong keywords, in Times Bold Italic in brown on a soft Hawaiian sea green (you are definitely a fine art *amateur*).

Then you need to look for 'k' and 'K':

```
/k {
  false BG
  0 0 0.9 FG
%Face: Keyword Courier bfs
  Show
} bind def

/K {
  false BG
  0 0 0.8 FG
%Face: Keyword_strong Courier-Bold bfs
  Show
} bind def
```

and turn it into:

```
/k {
  0.2 1 0.2 true BG
  1 0.2 1 FG
%Face: Keyword Helvetica bfs
  Show
} bind def

/K {
  0.4 0.2 0 true BG
  0.5 1 1 FG
%Face: Keyword_strong Times-BoldItalic bfs
  Show
} bind def
```

Waouh! It looks great!

A bit trickier: let change the way the line numbers are printed.

First, let's look for the font definition:

```
%%BeginSetup
% The font for line numbering
/f# /Helvetica findfont bfs .6 mul scalefont def
%%EndSetup
```

Let it be in Times, twice bigger than the body font.

```
%%BeginSetup
% The font for line numbering
/f# /Times-Roman findfont bfs 2 mul scalefont def
%%EndSetup
```

How about its foreground color?

```
% Function print line number (<string> # -)
/# {
  gsave
```

```
      sx cw mul 2 div neg 0 rmoveto
      f# setfont
      0.8 0.1 0.1 FG
      c-show
    grestore
  } bind def
```

Let it be blue. Now you know the process: just put '`0 0 1`' as `FG` arguments.

9 Contributions

This chapter documents the various shell scripts or other tools that are distributed with the `a2ps` package, but are not `a2ps` itself. The reader should also look at the documentation of `Ogonkify` (see Section "Overview" in *Ogonkify manual*), written by Juliusz Chroboczek.

9.1 `card`

Many users of `a2ps` have asked for a reference card, presenting a summary of the options. In fact, something closely related to the output of '`a2ps --help`'.

The first version of this reference card was a PreScript file (see Section 7.3.2 [PreScript], page 60) to be printed by `a2ps`. Very soon a much better scheme was found: using a style sheet to pretty print directly the output of '`a2ps --help`'! A first advantage is then that the reference cards can be printed in the tongue you choose.

A second was that this treatment could be applied to any application supporting a '`--help`'-like option.

9.1.1 Invoking `card`

`card [`*`options`*`] `*`applications`*` [-- -`*`options`*`]`

`card` is a shell script which tries to guess how to get your *applications*' help message (typically by the options '`--help`' or '`-h`'), and pretty prints it thanks to `a2ps` (or the content of the environment variable '`A2PS`' if it is set). *-options* are passed to `a2ps`.

Supported options are:

`-h` [Option]
`--help` [Option]
: print a short help message and exit successfully.

`-V` [Option]
`--version` [Option]
: report the version and exit successfully.

`-q` [Option]
`--quiet` [Option]
`--silent` [Option]
: Run silently.

`-D` [Option]
`--debug` [Option]
: enter in debug mode.

`-l `*`language`* [Option]
`--language=`*`language`* [Option]
: specify the language in which the reference card should be printed. *language* should be the symbol used by `LC_ALL` etc. (such as '`fr`', '`it`' etc.).

  If the *applications* don't support internationalization, English will be used.

`--command=`*command* [Option]

Don't try to guess the *applications*' way to report their help message, but rather use the call *command*. A typical example is

```
card --command="cc -flags"
```

It is possible to give options to `a2ps` (see Section 3.1 [Options], page 11) by specifying them after '`--`'. For instance

```
card gmake gtar --command="cc -flags" -- -Pdisplay
```

builds the reference card of `GNU make`, `GNU tar` (automatic detection of '`--help`' support), and `cc` thanks to '`-flags`'.

9.1.2 Caution when Using `card`

Remember that `card` runs the programs you give it, and the commands you supplied. Hence if there is a silly programs that has a weird behavior given the option '`-h`' etc., beware of the result.

It is even clearer using '`--command`': avoid running '`card --command="rm -rf *"`', because the result will be exactly what you think it will be!

9.2 `fixps`

The shell script `fixps` tries its best to fix common problems in PostScript files that may prevent post processing, using `ghostscript`. It is a good idea to use `fixps` in the PostScript delegations.

9.2.1 Invoking `fixps`

```
fixps [options] [file]
```

sanitize the PostScript *file* (or of the standard input if no *file* is given, or if *file* is '`-`').

Supported options are:

`-h` [Option]
`--help` [Option]

Print a short help message and a list of the fixes that are performed. Exit successfully.

`-V` [Option]
`--version` [Option]

report the version and exit successfully.

`-D` [Option]
`--debug` [Option]

enter in debug mode.

`-q` [Option]
`--quiet` [Option]
`--silent` [Option]

Run silently.

`-o file` [Option]
`--output=file` [Option]

specify the *file* in which is saved the output.

`-c` [Option]
`--check` [Option]
`--dry-run` [Option]
: Don't actually fix the *file*: just report the diagnostics. Contrary to the option '`fixps -qc`' does absolutely nothing (while it does take some time to do it nicely).

`-f` [Option]
`--force` [Option]
: Ask `ghoscript` for a full rewrite of the *file*. The output file is really sane, but can be much longer than the original. For this reason and others, it is not always a good idea to make a full rewrite. This option should be used only for files that give major problems.

9.3 `pdiff`

The shell script `pdiff` aims to pretty print diffs between files. It basically uses GNU `diff` (see Section "Overview" in *Comparing and Merging Files*) or GNU `wdiff` (see Section "The word difference finder" in *GNU wdiff*) to extract the diff, then calls `a2ps` with the correct settings to get a nice, printed contextual diff.

9.3.1 Invoking `pdiff`

`pdiff [`*`options`*`] `*`file-1 file-2`*` [-- `*`-options`*`]`

make a pretty comparison between *file-1* and *file-2*. *-options* are passed to `a2ps`.

Supported options are:

`-h` [Option]
`--help` [Option]
: print a short help message and exit successfully.

`-V` [Option]
`--version` [Option]
: report the version and exit successfully.

`-q` [Option]
`--quiet` [Option]
`--silent` [Option]
: Run silently.

`-D` [Option]
`--debug` [Option]
: enter in debug mode.

`-w` [Option]
`--words` [Option]
: Look for words differences (default). White space differences are not considered.

`-l` [Option]
`--lines` [Option]
: Look for lines differences.

It is possible to give options to `a2ps` (see Section 3.1 [Options], page 11) by specifying them after '`--`'. For instance

```
pdiff COPYING COPYING.LIB -- -1 -P display
```

Compares the files `COPYING` and `COPYING.LIB`, and prints it on the printer `display` (usually `Ghostview` or `gv`).

9.4 lp2

This program simplifies printing documents double-sided (duplex) or a single-sided (simplex) printer. The idea is simply first to print the odd pages, then the even in reversed order.

9.4.1 Invoking lp2

```
lp2 [options] [file]
```

print the given file using `lp`. First print the odd pages, then prompt the user to put the printed pages back in, then print the even pages in reverse order.

Supported options are:

`-h` [Option]
`--help` [Option]
print a short help message and exit successfully.

`-V` [Option]
`--version` [Option]
report the version and exit successfully.

`-f` [Option]
`--front` [Option]
Print only the front pages.

`-b` [Option]
`--back` [Option]
Print only the back pages.

Typical use is

```
lp2 file.ps
```

or can be put into `a2ps`' printer commands (see Section 4.5 [Your Printers], page 31).

10 Frequently asked questions

Please, before sending us mail, make sure the problem you have is not known, and explained. Moreover, avoid using the mailing list for asking question about the options, etc. It has been built for announces and suggestions, not to contact the authors.

10.1 Why Does...?

Error related questions.

10.1.1 Why Does it Print Nothing?

works OK, but the printer prints nothing.

There are two ways that printing can fail: silently, or with a diagnostic.

First, **check that the printer received what you sent**. `a2ps` may correctly do its job, but have the printer queue fail to deliver the job. In case of doubt, please check that the printer's leds blink (or whatever is its way to show that something is being processed).

If the printer does receive the job, but prints nothing at all, check that you did not give exotic options to an old printer (typically, avoid printing on two sides on a printer that does not support it). Avoid using '`-S`', '`--setpagedevice`' (see Section 8.2 [Page Device Options], page 79) and '`--statusdict`' (see Section 8.3 [Statusdict Options], page 79).

If the trouble persists, please try again but with the option '`--debug`' (a PostScript error handler is downloaded), and then send us:

1. the input file that gives problems
2. the output file created by `a2ps` **with the option '`--debug`'**
3. the error message that was printed.

10.1.2 Why Does it Print in Simplex?

Though I ask `a2ps` *to print Duplex via '`--sides`', the job is printed Simplex.*

If your printer is too old, then `a2ps` will not be able to send it the code it needs when '`-s2`' is specified. This is because your printer uses an old and not standardized interface for special features.

So you need to

1. specify that you want Duplex mode: '`-s2`',
2. remove by hand the standardized call to the Duplex feature: '`-SDuplex`',
3. add the non standard call to Duplex. Try '`--statusdict=setduplexmode:true`'.

Since this is painful to hit, a User Option (see Section 4.6 [Your Shortcuts], page 32) should help.

10.1.3 Why Does it Print in Duplex?

Though I ask `a2ps` *to print Simplex via '`--sides`', the job is printed Duplex.*

Actually when you require Simplex, `a2ps` issues nothing, for portability reasons. Hence, if your printer is defaulted to Duplex, the job will be Duplexed. So you have to force `a2ps` to issue the Simplex request with '`-SDuplex:false`'. The user options '`-=s1`' and '`-=simplex`' have names easier to remember.

In the next version of `a2ps` this kind of portability problems will be fixed in a user friendly way.

10.1.4 Why Does it Not Fit on the Paper?

When I print text files with `a2ps`*, it prints beyond the frame of the paper.*

You are most probably printing with a bad medium, for instance using A4 paper within `a2ps`, while your printer uses Letter paper. Some inkjet printers have a small printable area, and `a2ps` may not expect it. In both case, read Section 3.1.3 [Sheet Options], page 15, option '`--medium`' for more.

10.1.5 Why Does it Print Junk?

What I get on the printer is long and incomprehensible. It does not seem to correspond to what I wanted to print.

You are probably printing a PostScript file or equivalent. Try to print with '`-Z`': `a2ps` will try to do his best to find what is the program that can help you (see Section 4.10 [Your Delegations], page 35). In case of doubt, don't hesitate to save into a file, and check the content with `Ghostview`, or equivalent:

```
$ a2ps my_weird_file -Z -o mwf.ps
$ gv mwf.ps
```

If your `a2ps` is correctly installed, you can use the '`display`' fake-printer:

```
$ a2ps my_weird_file -Z -P display
```

If it is incorrect, ask for help around you.

10.1.6 Why Does it Say my File is Binary?

complains that my file is binary though it is not.

There are several reasons that can cause `a2ps` to consider a file is binary:

- there are many non printable characters in the file. Then you need to use the option '`--print-anyway`'.
- the file is sane, composed of printable characters. Then it is very likely that `file(1)` said the type of the file is '`data`', in which case `a2ps` prefers not to print the file. Then you can either:
  - specify the type of the file, for instance '`-Eplain`';
  - specify to print in any case, '`--print-anyway`';
  - remove the annoying rule from the system's `sheets.map`:

    ```
    binary: <data*>
    ```

  - insert in your own `~/.a2ps/sheets.map` a rule that overrides that of the system's `sheets.map`:

    ```
    # Load the system's sheets.map
    include(/usr/local/share/a2ps/sheets/sheets.map)
    ```

```
# Override the rule for files with type 'data' according to file(1)
plain: <data*>
```

But this is not very good, since then this rule is always the first tested, which means that any file with type '`data`' according to `file(1)` will be printed in '`plain`' style, even if the file is called `foo.c`.

– if your files can be recognized, insert a new rule in a `sheets.map`, such as

```
# file(1) says it's data, but it's pure text
plain:   /*.txx/
```

10.1.7 Why Does it Refuse to Change the Font Size

`a2ps` *does not seem to honor* `--font-size` *(or '*`--lines-per-page`*', or '*`--chars-per-line`*').*

This is probably because you used '`-1`'..'`-9`' after the '`--font-size`'. This is wrong, because the options '`-1`'..'`-9`' set the font size (so that there are 80 characters per lines), and many other things (See Section 3.1.4 [Page Options], page 16, option '`--font-size`').

Hence '`a2ps --font-size=12km -4`' is exactly the same thing as '`a2ps -4`', but is different from '`a2ps -4 --font-size=12km`'. Note that the 'pure' options (no side-effects) to specify the number of virtual pages are '`--columns`' and '`--rows`'.

10.2 How Can I ...?

A mini how-to on `a2ps`.

10.2.1 How Can I Leave Room for Binding?

The option '`--margin[=`*size*`]`' is meant for this. See Section 3.1.3 [Sheet Options], page 15.

10.2.2 How Can I Print `stdin`?

`a2ps` prints the standard input if you give no file name, or if you gave '`-`' as file name. Automatic style selection is of course much weaker: without the file name, `a2ps` can only get `file(1)`'s opinion (see Section 5.4 [Style Sheet Files], page 41). In general it means most delegations are safe, but there will probably be no pretty-printing.

You can supply a name to the standard input ('`--stdin=`*name*') with which it could guess the language.

10.2.3 How Can I Change the Fonts?

See Section 8.6 [Designing PostScript Prologues], page 80, for details. Make sure that all the information `a2ps` needs is available (see Section 5.3 [Font Files], page 40).

10.2.4 How Can I Simulate the Old Option '`-b`'?

By the past, `a2ps` had an option '`-b`' with which the fonts were bold. Since now the fonts are defined by prologues (see Section 8.6 [Designing PostScript Prologues], page 80) this option no longer makes sense. A replacement prologue is provided: '`bold`'. To use it, give the option '`--prologue=bold`'.

10.2.5 How Can I Pass Options to 'lpr'

How can I tell a2ps *to ask* lpr *no to print the banner?*

How can I pass specific options to lp*?*

If your 'Printer:' fields in the configuration files were properly filled (see Section 4.5 [Your Printers], page 31), you can use the variable 'lp.options' to pass options to lpr (or lp, depending on your environment):

```
a2ps -Dlp.options="-h -s" -P printer
```

You can also define 'lp.options' once for all (see Section 4.9.1 [Defining Variables], page 33).

Finally, you can use 'Printer:' several times to reach a printer with different lpr options.

10.2.6 How Can I Print Man Pages with Underlines

By the past, when I printed a man page with a2ps*, it used underlines, but now it uses italics. I want underlines back!*

Use 'a2ps --pro=ul'.

10.3 Please tell me...

Wondering something?

10.3.1 Is a2ps Y2K compliant?

The famous Y2K[1] problem...

Yes, a2ps is Y2K compliant since version 4.10.3.

Nevertheless, please note that you can still use two digit years. **You** are responsible for your choice of date format (see Section 3.2 [Escapes], page 24).

10.3.2 Why not having used yacc and such

There are several reasons why we decided not to use grammars to parse the files. Firstly it would have made the design of the style sheets much more tricky, and today a2ps would know only 4 or 5 languages.

Secondly, it limits the number of persons who could build a style sheet.

Thirdly, we did not feel the need for such a powerful tool: handling the keywords and the sequences is just what the users expect.

Fourthly, any extension of a2ps would have required to recompile.

And last but not least, using a parser requires that the sources are syntactic bug free, which is too strong a requirement.

Nevertheless, PreScript gives the possibility to have on the one hand a syntactic parser which would produce PreScript code, and on the other hand, a2ps, which would make it PostScript. This schema seems to us a good compromise. If it is still not enough for you, you can use the library.

[1] Year 2000.

Appendix A Glossary

This section settles some terms used through out this document, and provides the definitions of some terms you probably want to know about.

Adobe Adobe is the firm who designed and owns the PostScript language. The patent that printer manufacturers must pay to Adobe is the main reason why PostScript printers are so expansive.

AFM file AFM stands for Adobe Font Metrics. These files contain everything one needs to know about a font: the width of the characters, the available characters etc.

Charset
Code Set Cf. Encoding.

Delegate Another filter (application) which `a2ps` may call to process some files. This feature is especially meant for page description files (see Section 4.10 [Your Delegations], page 35).

DSC
Document Structuring Conventions

Because PostScript is a language, any file describing a document can have an arbitrary complexity. To ease the post-processing of PostScript files, the document should follow some conventions. Basically there are two kinds of conventions to follow:

Page Independence

Special comments state where the pages begin and end. With these comments (and the fact that the code describing a page starts and ends somewhere, which is absolutely not necessary in PostScript), very simple programs (such as `psnup`, `psselect` etc.) can post process PostScript files.

Requirements

Special features may be needed to run correctly the file. Some comments specify what services are expected from the printer (e.g., fonts, duplex printing, color etc.), and other what features are provided by the file itself (e.g., fonts, procsets etc.), so that a print manager can decide that a file cannot be printed on that printer, or that it is possible if the file is slightly modified (e.g., adding a required font not known by the printer) etc.

The DSC are edited by Adobe. A document which respects them is said to be *DSC conformant*.

`a2ps` follows all the DSC.

Duplex
DuplexTumble
DuplexNoTumble

To print *Duplex* is to print double-sided. There are two ways to print Duplex depending whether the second face is printed upside-down or not:

DuplexTumble
: DuplexTumble is suitable when (if it were to be bound) the document would be bound along the short edge (for instance when you are printing booklets).

DuplexNoTumble
: DuplexNoTumble corresponds to binding along the long edge of the medium. A typical case is when printing one-up.

Encoding
: Association of human readable characters, and computers' internal numbered representation. In other words, they are the alphabets, which are different according to your country/mother tongue. E.g.: ASCII, Latin 1, corresponding to Western Europe etc.

  To know more about encodings, see Section 6.1 [What is an Encoding], page 43.

`Ghostscript`
`gs`
: `Ghostscript`[1], `gs` for short, is a full PostScript interpreter running under many various systems (Unices, MS-DOS, Mac etc.). It comes with a large set of output formats allowing many different applications:

  Displaying
  : It can be used either to view PostScript files (in general thanks to a graphic interface such as `Ghostview` or `gv` ...).

  Converting
  : It can convert to other languages/formats, e.g. portable PostScript, Encapsulated PS etc.

  Translating
  : to a printer dedicated language, e.g., PCL. In particular, thanks to `ghostscript`, you may print PostScript files on non PostScript printers.

Face
: A virtual style given to some text. For instance, *Keyword*, *Comment* are faces.

Headings
: Everything that goes around the page and is not part of the text body. Typically the title, footer etc.

Key
: Many objects used in `a2ps`, such as encodings, have both a key and a name. The word *name* is used for a symbol, a label, which is only meant to be nice to read by a human. For instance '`ISO Latin 1`' is a name. `a2ps` never uses a name, but the key.

  A *key* is the identifier of a unique object. This is information that `a2ps` processes, hence, whenever you need to specify an object to `a2ps`, use the key, not its name. For instance '`latin1`' is the unique identifier of the '`ISO Latin 1`' encoding.

Logical page
: Cf. Virtual page.

[1] `https://www.ghostscript.com`

lhs
left hand side
: See *P-rule*.

Medium
: Official name (by Adobe) given to the output physical support. In other words, it means the description of a sheet, e.g., A4, Letter etc.

Name
: See *Key*.

Page
: A single side of a sheet.

Page Description Language
: A language that describes some text (which may be enriched with pointers, pictures etc.) and its layout. `HTML`, PostScript, LaTeX, `roff` and others are such languages. A file written in those languages is not made to be read as is by a human, but to be transformed (or compiled) into a readable form.

PCL
: FIXME:

PFA file
: PostScript Font in ASCII format. This file can be directly down loaded to provide support for another font.

PFB file
: PostScript Font in Binary format. In PFA files there are long sequences of hexadecimal digits. Here these digits are represented by their value, hence compressing 2 characters in a PFA into 1 in the PFB. This is the only advantage since a PFB file cannot be directly sent to printer: it must first be decompressed (hence turned into a PFA file) before being used.

PostScript
: *PostScript* is a page description language designed for *Raster output devices*. It is even more powerful than that: unlike to `HTML`, or `roff`, but as TeX and LaTeX, it is truly a programming language which main purpose is to draw (on sheets). Most programs are a list of instructions that describes lines, shades of gray, or text to draw on a page. This is the language that most printers understand.

  Note that the fact that PostScript is a programming language is responsible of both its success and its failure. It is a big win for the PostScript programmer who can easily implement a lot of nice visual effects. It is a big loss because the page descriptions can have an arbitrary complexity, hence rendering can be really slow (remember the first Laser you had, or even `Ghostscript`. `PDF` has been invented by Adobe to remedy these problems).

  PostScript is a trademark of Adobe Systems Incorporated.

PPD file
PostScript Printer Description file
: These files report everything one needs to know about a printer: the known fonts, the patches that should be down loaded, the available memory, the trays, the way to ask it duplex printing, the supported media, etc.

  PostScript has pretended to be a device independent page description language, and the PPD files are here to prove that device independence was a failure.

ProcSet
: Set of (PostScript) procedures.

Prologue PostScript being a language, a typical PostScript program (i.e. a typical PostScript file) consists of two parts. The first part is composed of resources, such as fonts, procsets, etc. and the second part of calls to these procedures. The first part is called the *prologue*, and the second, the *script*.

P-rule Pretty printing rule. It is composed of a *left-hand side*, (*lhs* for short), and a *right-hand side*, (*rhs*). The lhs describes when the rule is triggered (i.e., the pattern of text to match), and the rhs specifies the pretty printed output. See Section 7.5.5 [P-Rules], page 65, for more semantical details, and see Section 7.6.5 [Syntax for the P-Rules], page 69, for implementation.

`psutils` The *psutils*[2] is a set of tools for PostScript post processing. They let you resize the frame into which the page is drawn, reorder or select pages, put several pages onto a single sheet, etc. To allow the `psutils` to run correctly, the PostScript files must be DSC conformant, and the bad news is that many PostScript drivers produce files which are not. `fixps` uses `ghostscript` to fix non-conformant files.

Raster Image Processor
RIP The hardware and/or software that translates data from a high-level language (e.g., PostScript) into dots or pixels in a printer or image setter.

Raster Output Device
Behind these words is hidden the general class of devices which have Pixels that can be addressed individually: Laser, Ink or Dot printers, but also regular screens etc. It is typically opposed to the class of devices which *plot*, i.e., have a pen that they move on the paper.

rhs
right hand side
See *P-rule*.

RIP See *Raster Image Processor*.

Script See *Prologue*.

Sheet The physical support of the printing: it may support one or two pages, depending on your printing options.

Style sheet
Set of rules used by `a2ps` to give a face to the strings of a file. In `a2ps`, each programming language which is supported is defined via one style-sheet.

Tumble See *Duplex*.

Virtual page
Area on a physical page in which `a2ps` draws the content of a file. There may be several virtual pages on a physical page. ("virtual page" is the name recommended by Adobe).

[2] `https://github.com/rrthomas/psutils`

Appendix B Genesis

Here are some words on `a2ps` and its history.

B.1 History

The initial version was a shell program written by Evan Kirshenbaum. It was very slow and contained many bugs.

A new version was written in `C` by Miguel Santana to improve execution speed and portability. Many new features and improvements have been added since this first version. Many contributions (changes, fixes, ideas) were done by `a2ps` users in order to improve it.

From the latest version from Miguel Santana (4.3), Emmanuel Briot implemented bold faces for keywords in `Ada`, `C` and `C++`.

From that version, Akim Demaille generalized the pretty-printing capabilities, implemented more languages support, and other features.

Masayuki Hatta maintained a2ps for several years. Later, David Seifert modernized and cleaned up the code considerably. Latterly, Reuben Thomas did more clean-up and bug-fixing, and released version 4.15, the first release for many years.

B.2 Thanks

Patrick Andries and Roman Czyborra provided us with important information on encodings.

Juliusz Chroboczek worked a lot on the integration of the products of Ogonkify (such as Latin 2 etc. fonts) in `a2ps`. Without his help, and the time is devoted to both `a2ps` and `ogonkify`, many non west-European people would still be unable to print easily texts written in their mother tongue.

Denis Girou brought a constant and valuable support through out the genesis of pretty-printing `a2ps`. His comments on both the program and the documentation are the origin of many pleasant features (such as '`--prologue`').

Alexander Mai provided us with invaluable help in the development. He spotted several times subtle bugs in `a2ps` and the contributions, he keeps a vigilant eye on portability issues, he checks and improves the style sheets, and he maintains a port of `a2ps` for OS/2.

Graham Jenkins, with an extraordinary regularity, tortures `a2ps` on weird systems that nobody ever heard of '`:)`'. Graham is usually the ultimate test: if he says I can release `a2ps`, I rest reassured that, yes, this time it **will** compile! If `a2ps` works today on your system, you should thank Graham too!

Of course this list is not up to date, and never will be. We would like to thank everybody that helped us, talked to us, and even criticized us with the intention to help us to improve `a2ps`. In particular, many thanks to all those who wrote style sheets, translated `a2ps`, reported and fixed bugs, and gave us other feedback over the years.

Appendix C Copying

The subroutines and source code in the `a2ps` package are "free"; this means that everyone is free to use them and free to redistribute them on a free basis. The `a2ps`-related programs are not in the public domain; they are copyrighted and there are restrictions on their 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 these programs that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the programs that relate to `a2ps`, that you receive source code or else can get it if you want it, that you can change these programs or use pieces of them 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 the `a2ps`-related code, 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 the programs that relate to `a2ps`. If these programs are 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 not reflect on our reputation.

The precise conditions of the licenses for the programs currently being distributed that relate to `a2ps` are found in the General Public Licenses that accompany them.

Concept Index

M

O

P

R

S

T

U

V

W

Table of Contents